LEARNLAB 2.0

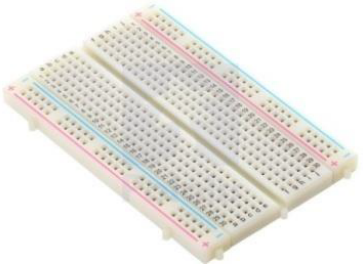
IEEE SB GCEK 2K24-25

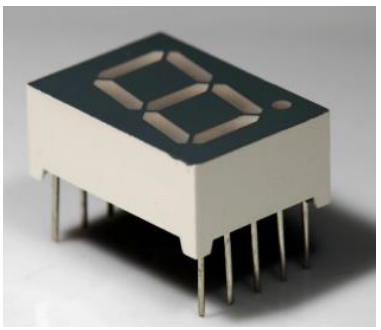
REACH US AT : AJAY E .K.

CONNECT: ajaysurendran@ieee.org

Table Of Contents

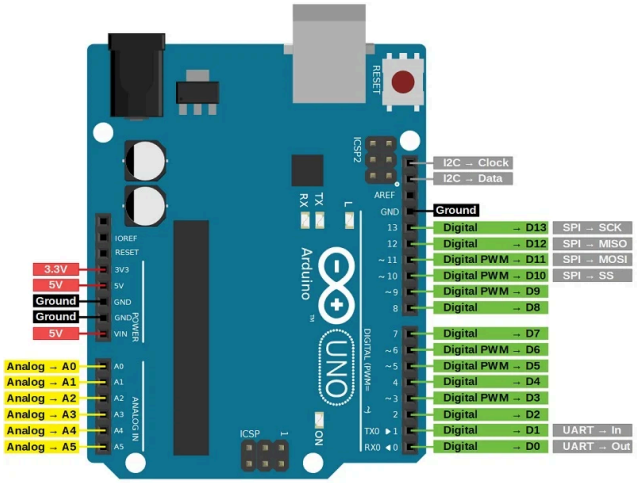
Get to Know Your Components:

SI NO	NAME	IMAGE
1	**BREAD BOARD**	
2	**JUMPER WIRES**	
3	**RESISTORS**	
4	**LED**	
5	**BUZZER**	

6	**PUSH BUTTONS**	
7	**IR SENSOR**	
8	**7-SEGMENT DISPLAY**	
9	**ULTRASONIC SENSOR**	
10	**RFID TAG & READER RC522**	

Discover the Power of Arduino Uno: A Beginner's Guide

Arduino Uno: Your Gateway to Innovation



Pin out Diagram of an Arduino

- **What is Arduino Uno?**
  Arduino is a compact, open-source electronics platform designed for creating interactive projects. It features user-friendly hardware and software, making it perfect for prototyping, DIY projects, and STEM education. Its simplicity and versatility make it an excellent choice for both beginners and advanced users.
- **Key Components:**
  - **ATmega328P Microcontroller:** The brain of the Arduino, responsible for processing inputs and controlling outputs.
  - **Digital I/O Pins:** 14 pins that can be used as either inputs or outputs for digital signals.
  - **Analog I/O Pins:** 6 pins for reading analog signals and converting them into digital values.
  - **Power Jack:** Allows the board to be powered from an external power source.
  - **USB Port:** Used for programming the Arduino and powering it from a computer.
  - **Reset Button:** Resets the microcontroller to restart the program.
- **Additional Features:**

    - **Built-in LED:** Connected to digital pin 13, useful for testing and debugging.
    - **PWM Pins:** 6 digital pins that provide Pulse Width Modulation output for controlling devices like motors and LEDs.
- **Applications:**
    - **Prototyping:** Quickly build and test electronic circuits and systems.
    - **Education:** Teach and learn fundamental concepts in electronics and programming.
    - **DIY Projects:** Create custom gadgets and interactive installations.
    - **Hobby Projects:** Explore and experiment with a wide range of electronic components and sensors.
- **Why Choose Arduino Uno?**
  With a large and active community, extensive documentation, and a plethora of tutorials and resources, Arduino Uno is accessible and well-supported. It offers a straightforward entry into the world of electronics and programming, making it an ideal tool for innovation and learning.

Project 1: LED Blink Project: A Beginner's Approach

This project provides a fundamental introduction to using an Arduino board by demonstrating a basic LED blinking example. This hands-on experience will help you understand how to set up and program your Arduino, making it an ideal starting point for beginners.

For a visual guide, you can watch the introductory video here:

Arduino LED Blink Tutorial.

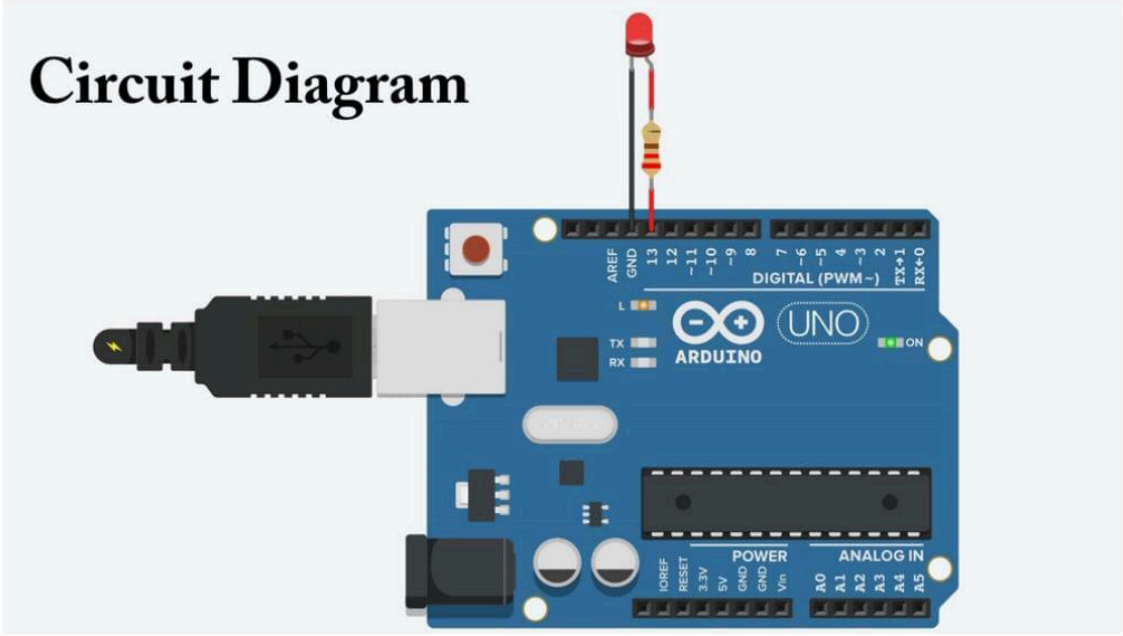
Arduino Blink Code:

Below is the standard code used in the Arduino IDE to blink an LED:

```
# Arduino LED Blink Code

# Pin number for the LED

const int ledPin = 13;



void setup() {

  // Initialize the LED pin as an output

  pinMode(ledPin, OUTPUT);

}



void loop() {

  // Turn the LED on

  digitalWrite(ledPin, HIGH);
```

```
  # Wait for 1000 milliseconds (1 second)

  delay(1000);

  // Turn the LED off

  digitalWrite(ledPin, LOW);

  # Wait for 1000 milliseconds (1 second)

  delay(1000);

}
```

You can follow these steps to ensure your code is easily readable and visually appealing in Google Docs. Let me know if you need more help with this!



Click here to download The code And try this for your self

Project 2: Tachometer - Measuring Rotational Speed

2.1 Introduction

This project focuses on creating a tachometer using an Arduino Uno and an IR sensor module. A tachometer is an instrument used to measure the rotational speed of a shaft or disk, usually expressed in revolutions per minute (RPM). This project provides hands-on experience with sensor integration, data processing, and real-time speed measurement. You will learn how to interface an IR sensor with the Arduino, process the signals, and display the RPM on a suitable output.

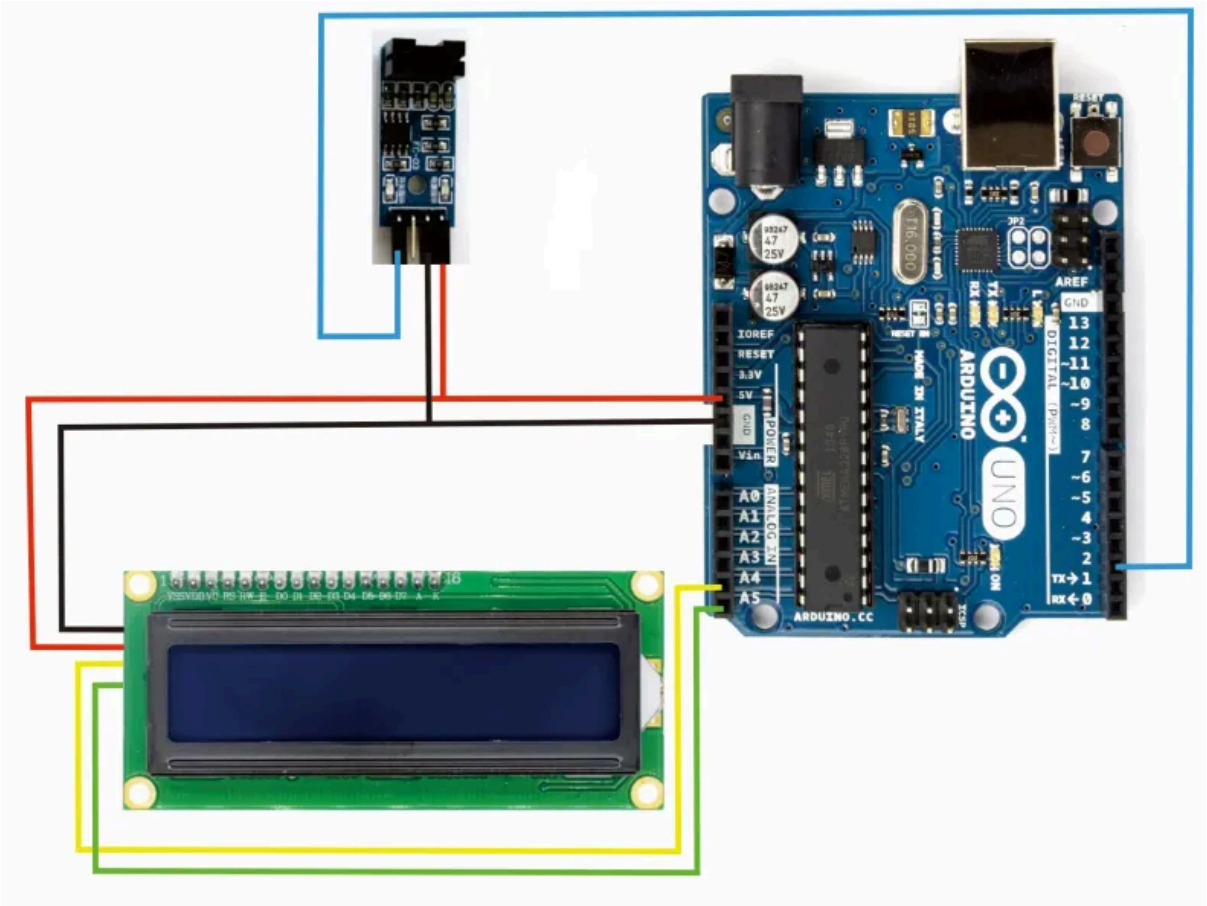
For a visual guide and further explanation, watch the introductory video here: Tachometer Project Overview.

2.2 Components Required

1. **Arduino Uno**: The microcontroller board is used to process the IR sensor data and control the project.
2. **IR Sensor Module**: Detects the rotation of the shaft and provides the necessary input signals to the Arduino.
3. **Breadboard**: Used for connecting the components without soldering.
4. **Jumper Wires**: Connect the components to the breadboard and Arduino.

2.3 Circuit

The circuit setup involves connecting the IR sensor to the Arduino via the breadboard. The IR sensor's output is connected to one of the digital input pins on the Arduino. For detailed circuit connections and setup, refer to the video tutorial available at the provided link.



2.4 Code

The code for the tachometer processes the signals from the IR sensor to calculate the RPM. It reads the sensor's output, counts the pulses, and then calculates the rotational speed. You can download the complete code and instructions from the following link: Download Tachometer Code.

2.5 Additional Information

- **Calibration:** Ensure proper calibration of the IR sensor to get accurate RPM readings. Adjustments may be needed based on the sensor's position and the rotating object.
- **Display Options:** Consider adding an LCD or serial monitor to display the RPM readings in real time.
- **Safety:** Always take necessary precautions while working with electronic components and rotating machinery to ensure safety.

Project 3: Ultrasonic Distance Measurement System

3.1 Introduction

This project focuses on developing a distance measuring system using an Arduino Uno and an ultrasonic sensor. Ultrasonic sensors use sound waves to measure the distance between the sensor and an object by calculating the time it takes for the sound waves to bounce back. This project will teach you how to interface an ultrasonic sensor with the Arduino, process the sensor's data, and display the distance measurements. It's a practical application useful for various automation and monitoring tasks.

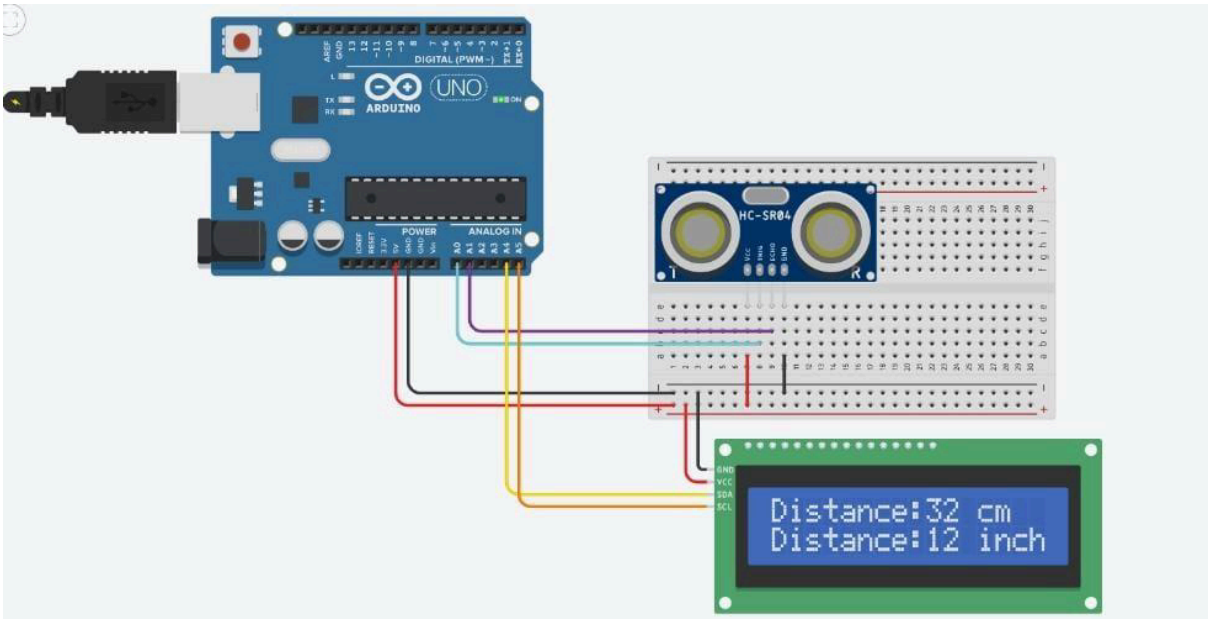
For a visual guide and detailed explanation, you can watch the introductory video here: Distance Measuring System Overview.

3.2 Components Required

1. **Arduino Uno**: The microcontroller board used to control the ultrasonic sensor and process the data.
2. **Ultrasonic Sensor**: Measures the distance to an object using ultrasonic waves.
3. **Breadboard**: Provides a platform for connecting the components without soldering.
4. **Jumper Wires**: Connects the ultrasonic sensor to the Arduino and the breadboard.

3.3 Circuit

To set up the circuit, connect the ultrasonic sensor to the Arduino using the breadboard and jumper wires. The ultrasonic sensor typically has four pins: VCC, Trig, Echo, and GND. Connect these pins to the corresponding pins on the Arduino for power and signal transmission. For detailed circuit connections, refer to the code link provided below.



3.4 Code

The Arduino code for the distance measuring system reads data from the ultrasonic sensor and calculates the distance based on the time it takes for the sound waves to return. You can view and download the complete code from the following link: Distance Measurement Code.

3.5 Additional Information

- **Calibration:** Ensure the ultrasonic sensor is properly calibrated for accurate distance measurements. The sensor's range and accuracy can vary based on the environment and the object being measured.
- **Display Options:** Consider integrating an LCD or serial monitor to display distance readings in real-time.
- **Safety Precautions:** While working with electronic components and sensors, follow standard safety practices to avoid any hazards.

Project 4: Advanced Door Security System

4.1 Introduction

This project involves designing an advanced door security system using an Arduino. The system utilizes an IR sensor to detect motion or presence near the door and activates an alert using a buzzer. This project aims to enhance security by providing an immediate audio alert when unauthorized access is detected, offering a practical solution for home or office security.

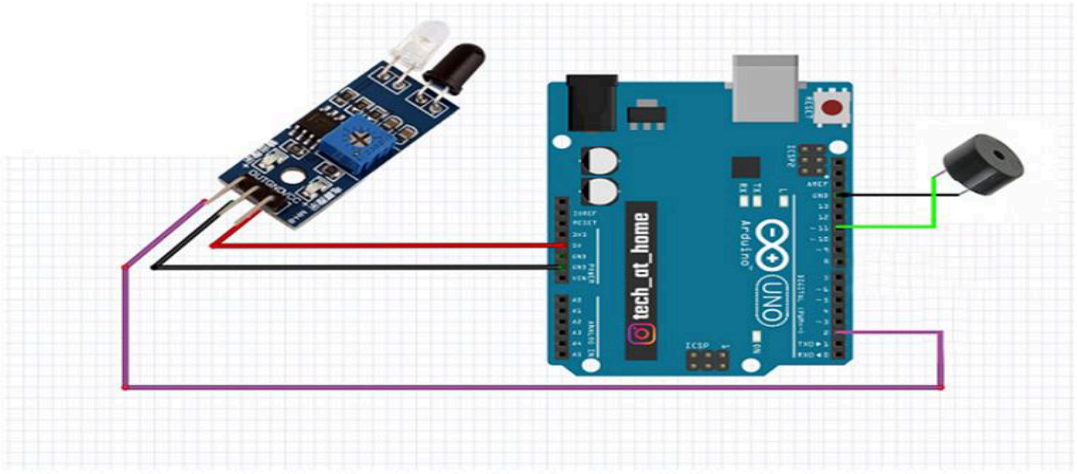
For a comprehensive visual guide on the setup and implementation, watch the introductory video here: Advanced Door Security System Overview.

4.2 Components Required

1. **Arduino Uno**: The microcontroller that processes the sensor data and controls the alert system.
2. **IR Sensor**: Detects movement or presence and sends a signal to the Arduino.
3. **Buzzer**: Emits an audible alarm when motion is detected by the IR sensor.
4. **Jumper Wires**: Connects the components to the Arduino and breadboard for proper operation.

4.3 Circuit

The circuit involves connecting the IR sensor and buzzer to the Arduino board. The IR sensor's output is connected to a digital input pin on the Arduino, while the buzzer is connected to a digital output pin. For detailed wiring instructions and connections, please refer to the provided code link.

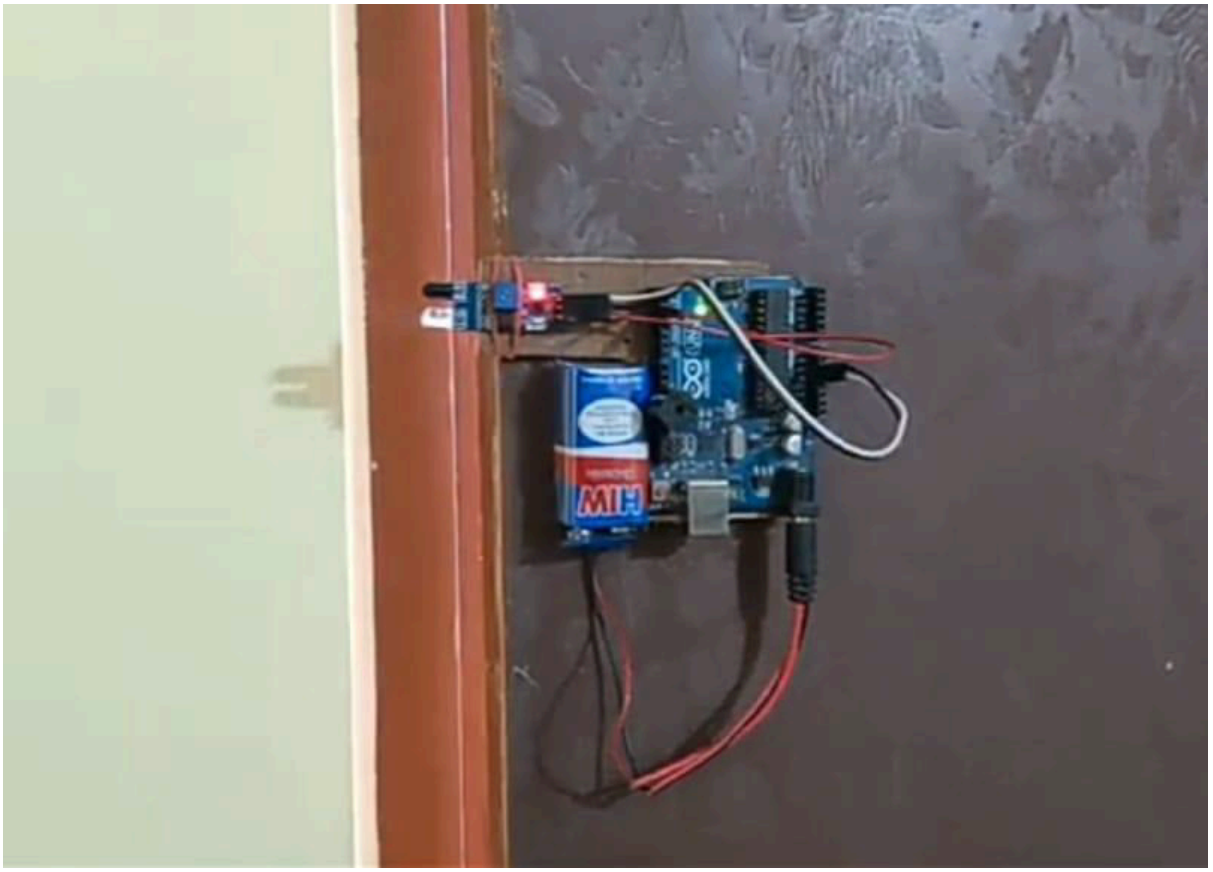


4.4 Code

The Arduino code for this project handles the IR sensor input and triggers the buzzer to sound an alarm when movement is detected. Download the complete code and detailed instructions from the following link: Download Security System Code.

4.5 Additional Information

- **Calibration:** Adjust the IR sensor's sensitivity to ensure effective detection and accurate alerts.
- **Power Requirements:** Ensure adequate power supply for the Arduino and components to maintain system reliability.
- **Installation:** Properly mount the IR sensor and buzzer to maximize detection range and alarm effectiveness.

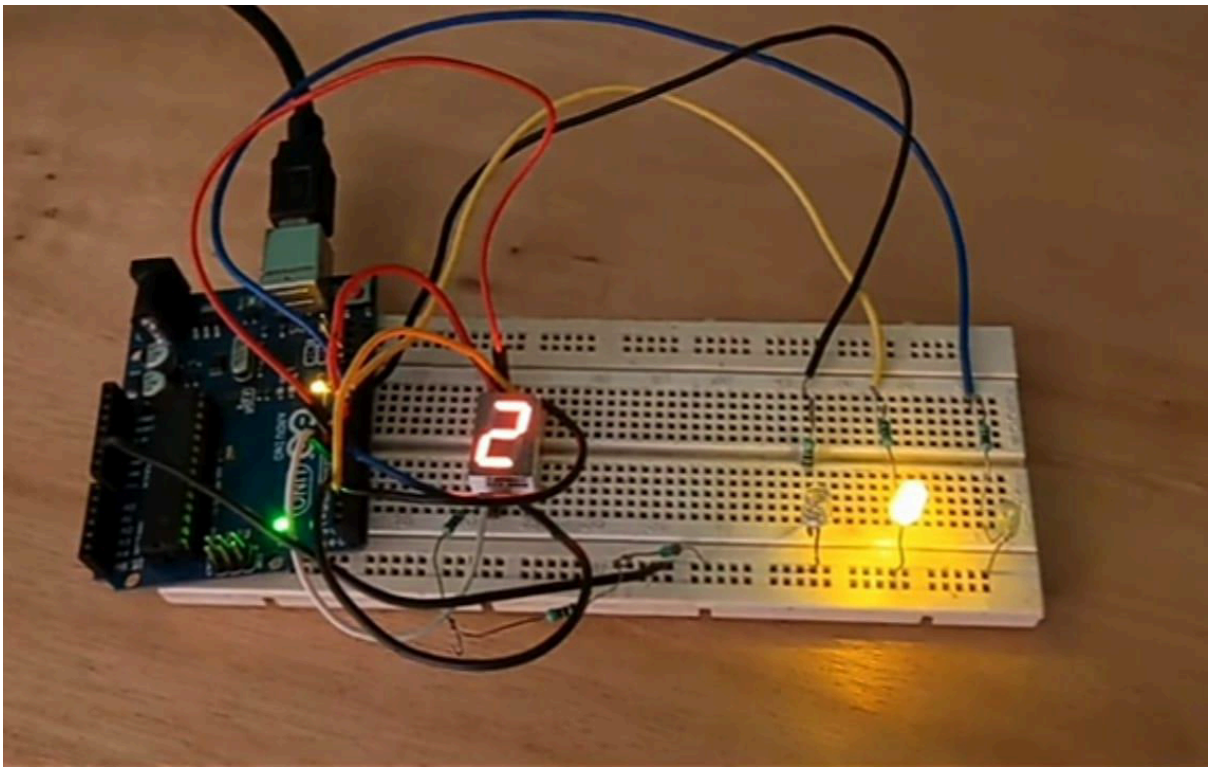


Project 5: Intelligent Traffic Light Controller

5.1 Introduction

This project involves creating an intelligent traffic light controller using an Arduino and a 7-segment display. The system simulates a traffic light system, cycling through red, yellow, and green lights to control the flow of traffic. This project provides insights into traffic management and display control, making it a practical and educational application of Arduino.

For a detailed visual guide and setup instructions, watch the video here: Traffic Light Controller Overview.



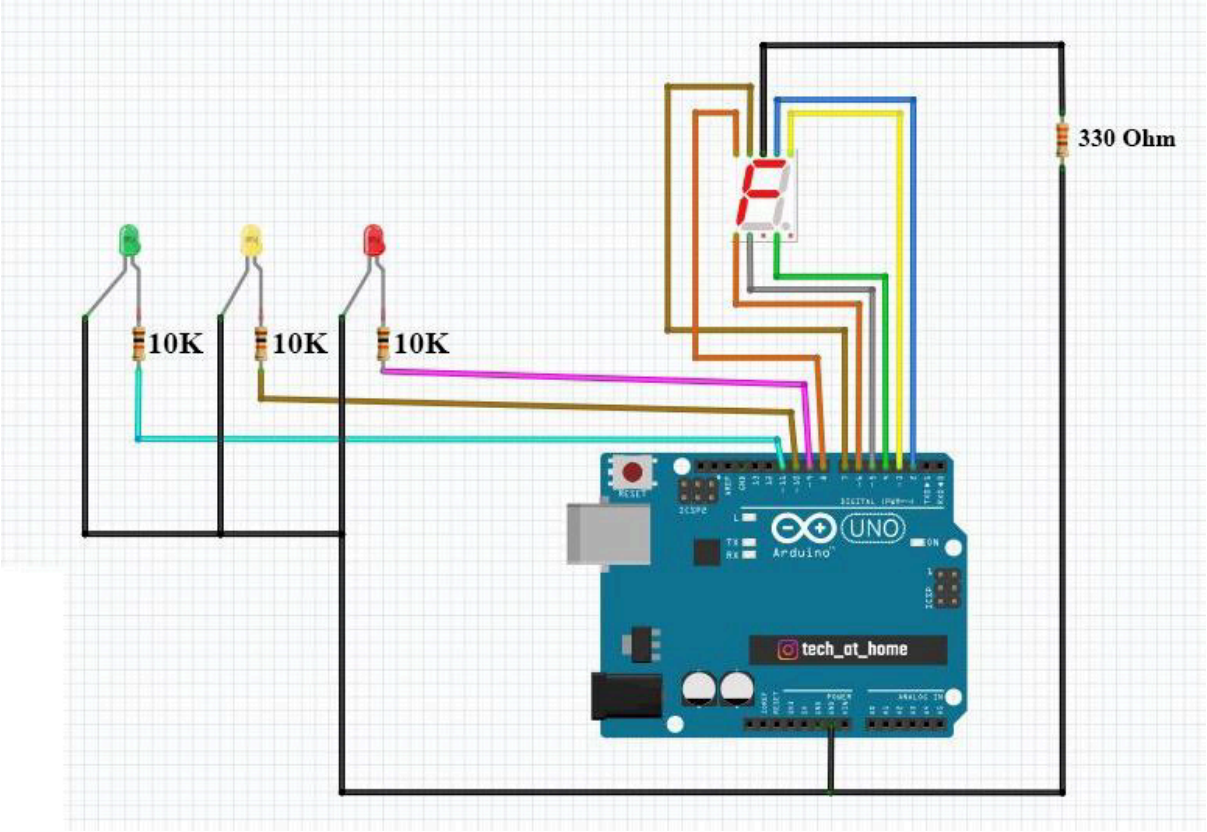
5.2 Components Required

1. **Arduino Uno**: The microcontroller board used to control the traffic light sequences.
2. **Seven Segment Display**: Displays the traffic light status.
3. **Breadboard**: A platform for connecting and arranging electronic components.
4. **Jumper Wires**: Connects the components to the Arduino and breadboard.
5. **LEDs**: Represent the traffic lights (red, yellow, and green).

6. **Resistors**: 330Ω and 10kΩ for current limiting and pull-up functions.

5.3 Circuit

The circuit involves connecting the seven-segment display and LEDs to the Arduino. The LEDs will be used to simulate traffic light signals, while the seven-segment display shows the light status. Detailed circuit diagrams and wiring instructions are provided in the linked code resource.



5.4 Code

The Arduino code controls the traffic light sequence, including the timing for red, yellow, and green lights, and displays the current status on the seven-segment display. Download the complete code and implementation details from the following link: Download Traffic Light Controller Code.

5.5 Additional Information

- **Timing Adjustments:** Modify the timing intervals in the code to simulate different traffic light durations as needed.
- **Power Considerations:** Ensure sufficient power supply for the Arduino and connected components to maintain stable operation.

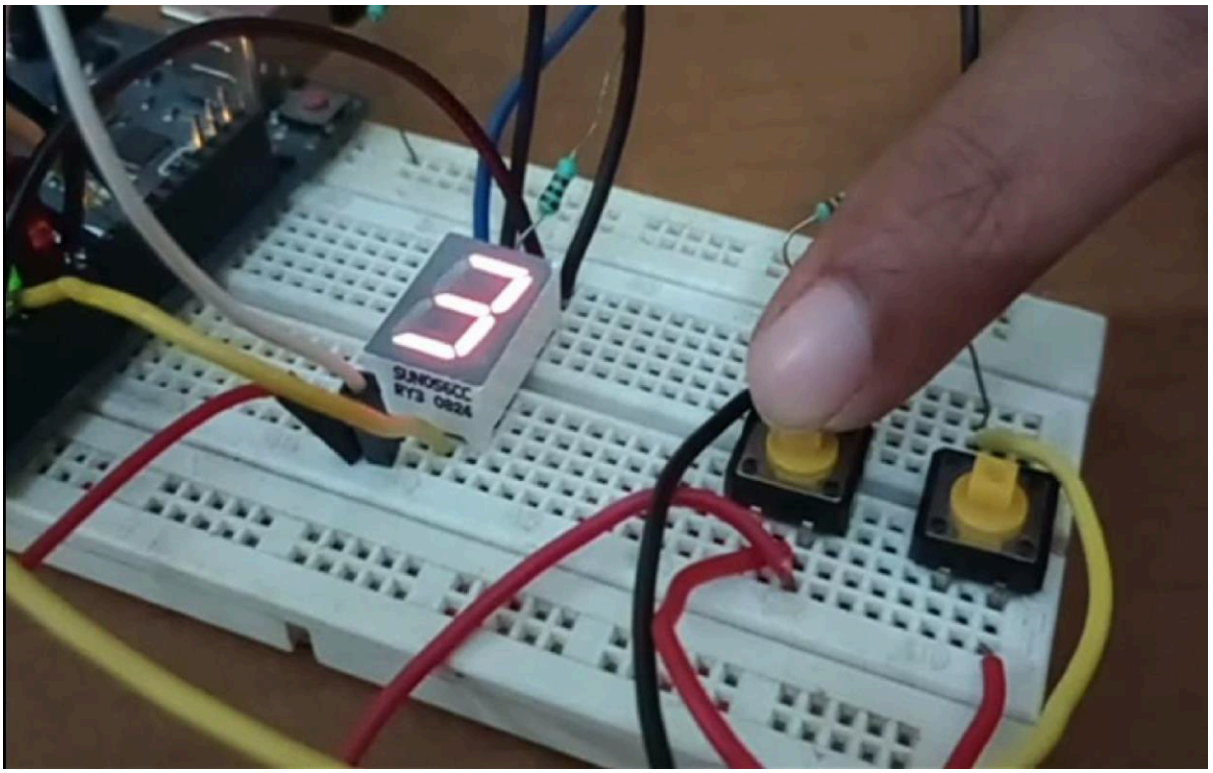
- **Component Placement:** Properly position the LEDs and seven-segment display on the breadboard to avoid wiring errors and ensure correct functioning.

🧊Project 6: Digital Counter with Arduino

6.1 Introduction

This project aims to build a digital counter using an Arduino, a 7-segment display, and push buttons. The counter will increment or decrement based on button presses, providing a simple yet effective demonstration of digital counting and display control. This project is ideal for learning about user inputs, digital outputs, and basic interfacing with display components.

For a detailed walkthrough and setup instructions, watch the video here: Digital Counter Project Overview.

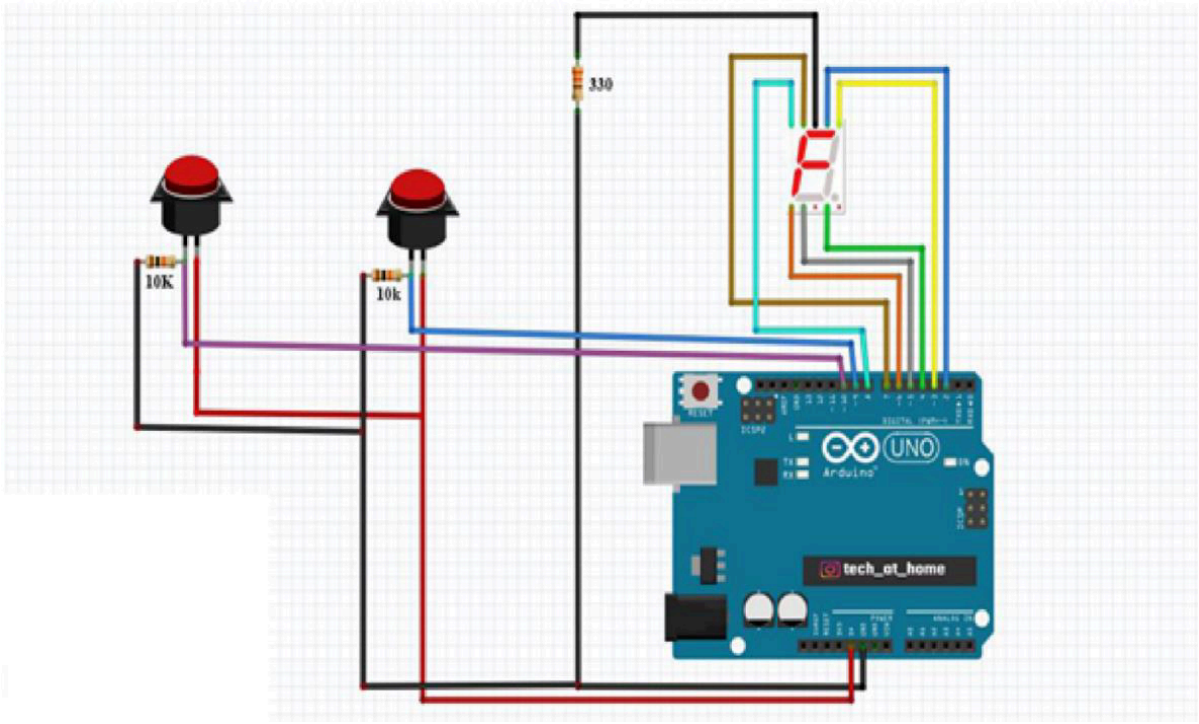


6.2 Components Required

1. **Arduino Uno**: The microcontroller board that processes button inputs and controls the 7-segment display.
2. **Breadboard**: A platform for assembling the circuit without soldering.
3. **Push Buttons**: Used to increment or decrement the counter value.
4. **Seven Segment Display**: Displays the current count.
5. **Resistors**: 330Ω and 10kΩ for current limiting and pull-up purposes.

6.3 Circuit

The circuit involves connecting the 7-segment display and push buttons to the Arduino. The push buttons are wired to digital input pins, while the 7-segment display is connected to the Arduino's output pins. Detailed wiring instructions and connections are provided in the code link below.



6.4 Code

The Arduino code manages the counting functionality, updates the 7-segment display, and responds to button presses. Download the complete code and additional setup details from the following link: Download Counter Code.

6.5 Additional Information

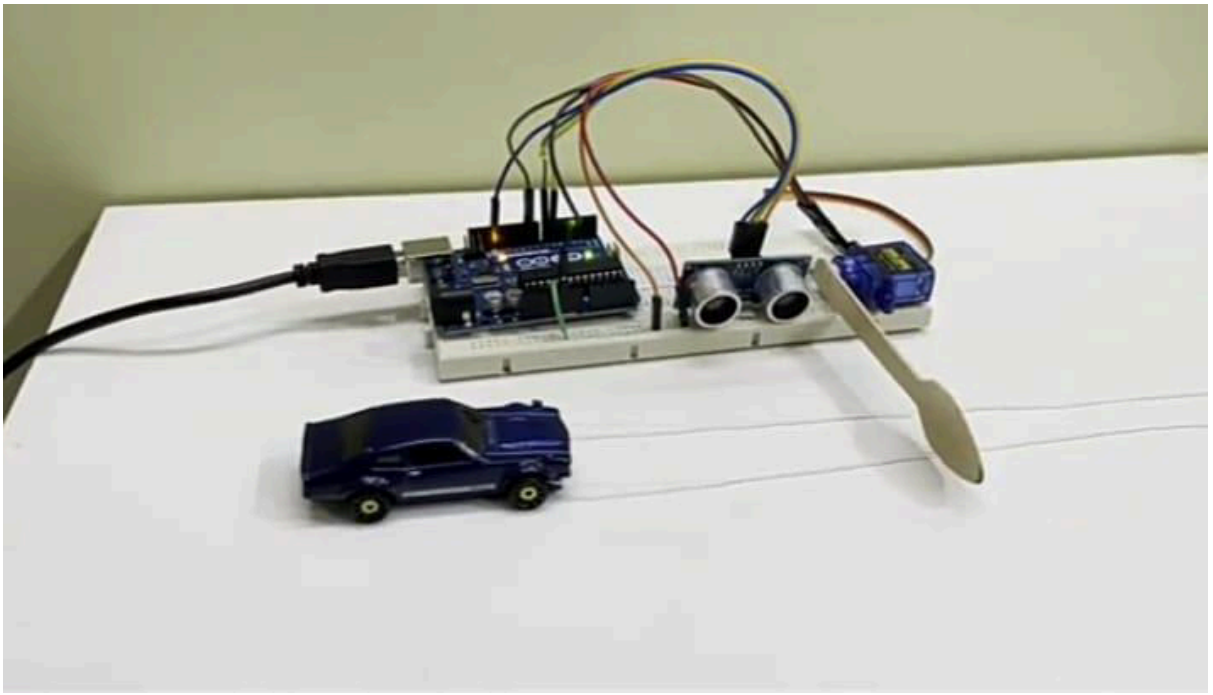
- **Debouncing:** Implement debouncing in the code to ensure accurate button presses and prevent false triggers.
- **Power Supply:** Ensure the Arduino and components have a stable power supply to avoid operational issues.
- **Component Arrangement:** Properly position the 7-segment display and buttons on the breadboard to ensure correct functionality and avoid wiring mistakes.

Project 7: Automatic Car Toll Gate System

7.1 Introduction

This project focuses on creating an automatic car toll gate system using an Arduino Uno. The system utilizes a servo motor and an ultrasonic sensor to manage and control vehicle access. When a vehicle approaches, the ultrasonic sensor detects its presence, and the servo motor operates to open or close the gate, simulating a real-world toll gate system.

For a visual guide and setup instructions, check out the video here: Automatic Car Toll Gate Overview.



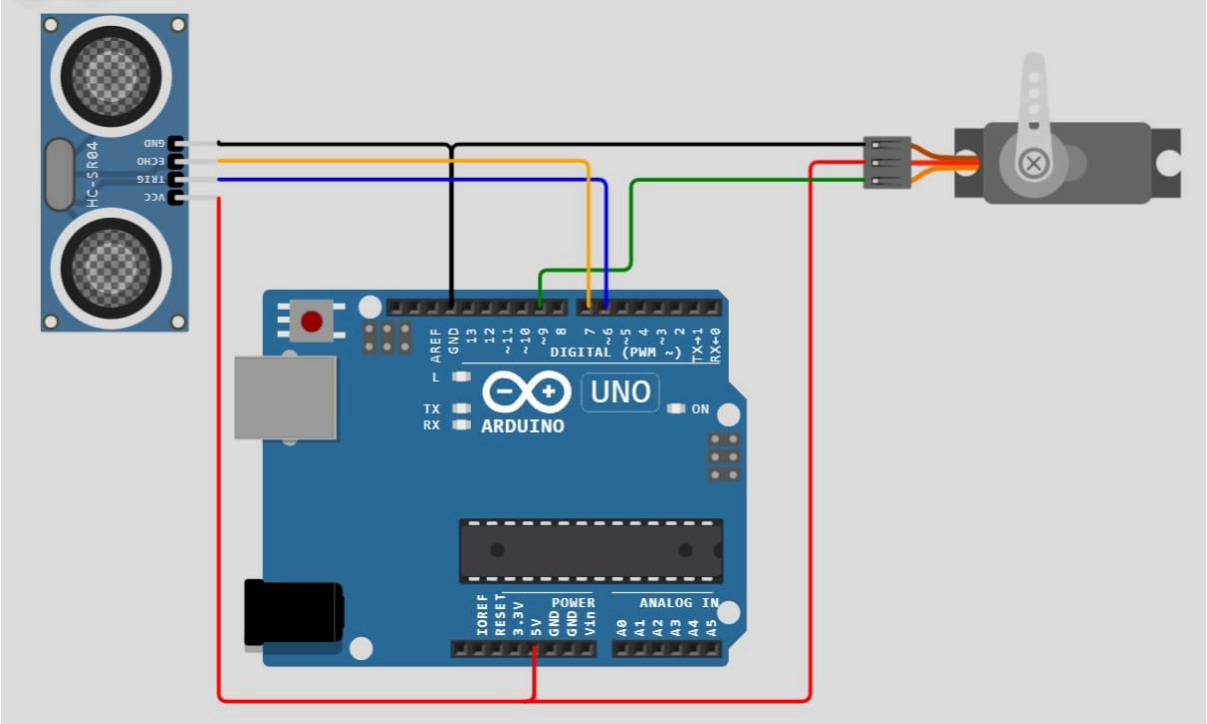
7.2 Components Required

1. **Arduino Uno**: The microcontroller that controls the servo motor and processes input from the ultrasonic sensor.
2. **Breadboard**: Used for connecting and organizing the components without soldering.
3. **Jumper Wires**: Connect the components to the Arduino and breadboard.
4. **Servo Motor**: Controls the movement of the toll gate.
5. **Ultrasonic Sensor**: Detects the presence of vehicles and triggers the gate mechanism.

7.3 Circuit

The circuit involves connecting the ultrasonic sensor and servo motor to the

Arduino. The ultrasonic sensor is used to detect the vehicle's distance, and the servo motor is used to open or close the gate based on the sensor input. Detailed wiring instructions and circuit diagrams can be found in the code link below.



7.4 Code

The Arduino code manages the toll gate system by controlling the servo motor based on the distance readings from the ultrasonic sensor. Download the complete code and additional setup information from the following link: [Download Toll Gate Code](#).

7.5 Additional Information

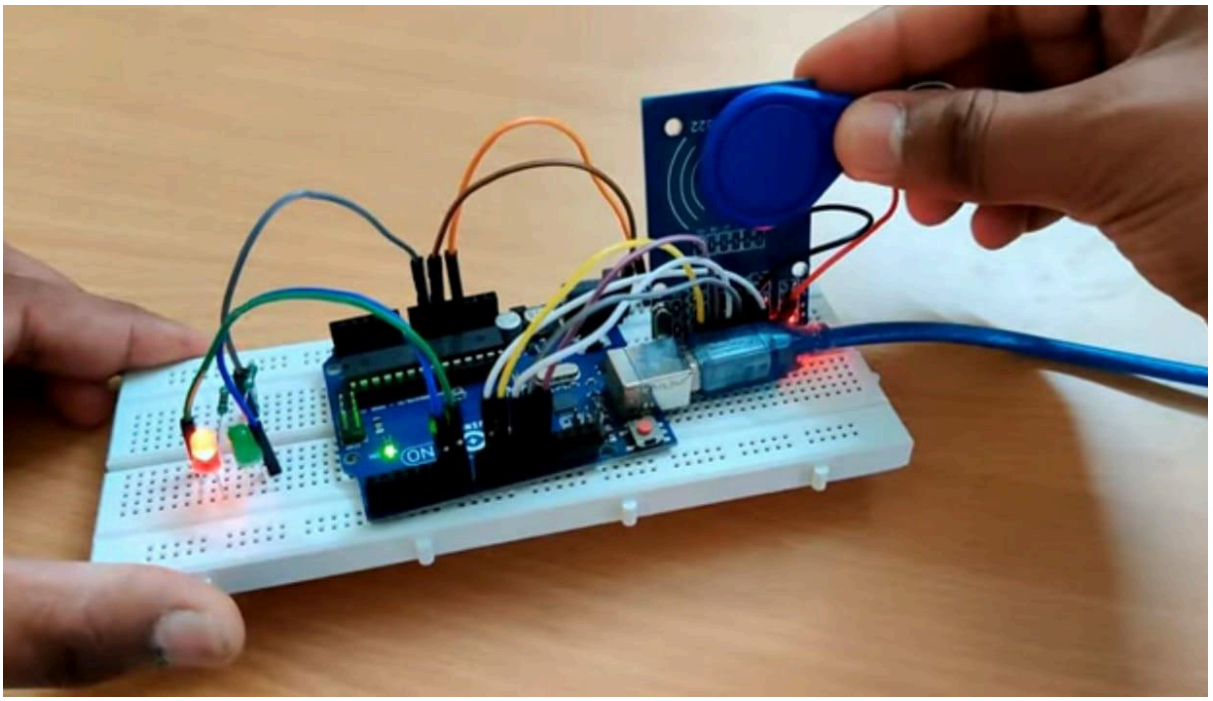
- **Calibration:** Properly calibrate the ultrasonic sensor to ensure accurate vehicle detection and gate operation.
- **Power Supply:** Ensure a stable power supply for the Arduino and servo motor to maintain reliable performance.
- **Mounting:** Securely mount the servo motor and ultrasonic sensor to achieve effective gate control and vehicle detection.

Project 8: RFID-Based Access Control System

8.1 Introduction

This project focuses on integrating RFID technology with an Arduino Uno to create an access control system. The system uses an RFID tag and reader (RC522) to authenticate access. When an RFID tag is scanned, the system can trigger various responses, such as unlocking a door or activating other devices, making it a practical demonstration of RFID technology in security and automation.

For a detailed visual guide and setup instructions, watch the video here: RFID-Based Access Control Overview.

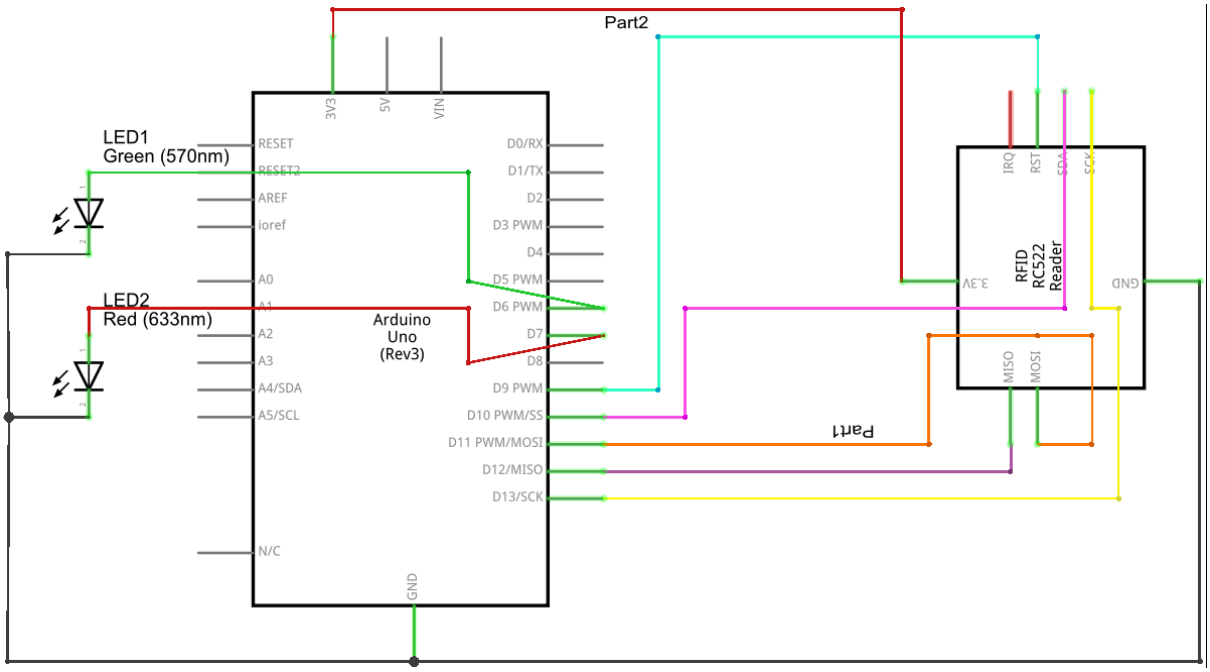


8.2 Components Required

1. **Arduino Uno**: The microcontroller that handles RFID tag reading and control logic.
2. **RFID Tag & Reader (RC522)**: The RFID system for scanning and reading tags.
3. **Push Buttons**: Optional components for additional control or user interface.
4. **Breadboard**: For assembling the circuit without soldering.
5. **Jumper Wires**: Connects components to the Arduino and breadboard.

8.3 Circuit

The circuit connects the RFID reader to the Arduino, allowing the microcontroller to read data from the RFID tag. Push buttons can be included for additional functionality or control. Detailed wiring instructions and connections are provided in the linked code resource.



8.4 Code

The Arduino code handles RFID tag reading and processes the data to control access. Download the complete code and additional setup details from the following link: Download RFID Code.

8.5 Additional Information

- **RFID Tag Management:** Ensure proper handling and programming of RFID tags for secure access control.
- **Power Supply:** Verify adequate power for the Arduino and RFID reader to avoid operational issues.
- **Component Placement:** Properly position and connect the RFID reader and push buttons on the breadboard for accurate functionality.

Project 9: RFID Access Control System with Arduino and Servo

9.1 Introduction

This project implements an RFID-based access control system using an Arduino Uno. The system reads an RFID tag using the MFRC522 RFID reader module and checks if the tag's unique identifier (UID) matches a predefined authorized UID. If the tag is authorized, a servo motor simulates unlocking, and an LED indicates access. If unauthorized, different LED lights up, signalling access denial.

Purpose

The project can be used in various applications, such as secure access systems for doors, safes, or restricted areas.

9.2 Components Required

1. **Arduino Uno**: The microcontroller board.
2. **MFRC522 RFID Module**: Reads the RFID tags.
3. **Servo Motor**: Simulates a physical lock or barrier.
4. **LEDs (2)**: One indicates authorized access, the other for denied access.
5. **Jumper Wires**: To connect components.
6. **Breadboard**: For circuit setup.

9.3 Circuit Connections

- **MFRC522 RFID Module**:
  - **SS_PIN**: Pin 10
  - **RST_PIN**: Pin 9
  - **SDA, MOSI, MISO, SCK**: Connected to the SPI pins on Arduino.
- **Servo Motor**: Connected to pin 7 of Arduino.
- **LEDs**:
  - **Green LED (Access Granted)**: Pin 6.
  - **Red LED (Access Denied)**: Pin 4.

9.4 Code Breakdown

Libraries and Definitions

```
#include <SPI.h>

#include <MFRC522.h>

#include <Servo.h>


#define SS_PIN 10

#define RST_PIN 9

#define servopin 7

#define acessLed 6

#define noacessLed 4

MFRC522 mfrc522(SS_PIN, RST_PIN);

Servo myservo;
```

- The required libraries SPI.h, MFRC522.h, and Servo.h are included to handle RFID communication and servo motor control.
- Pins for RFID readers, servo, and LEDs are defined.

Setup Function

```
void setup() {

  myservo.attach(servopin);

  myservo.write(0);

  pinMode(7,OUTPUT);

  pinMode(6,OUTPUT);
```

```
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println("Approximate your card to the reader...");
}
```

- Initializes the RFID reader and servo. The servo starts at the 0-degree position (locked state).
- Serial communication starts to print messages to the monitor.

Loop Function

```
void loop() {
  if (!mfrc522.PICC_IsNewCardPresent()) {
    return;
  }
  if (!mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  Serial.print("UID tag :");
  String userid= "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
    userid.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    userid.concat(String(mfrc522.uid.uidByte[i], HEX));
```

```
 }
 Serial.println();
 userid.toUpperCase();
```

- The code reads the RFID card, prints its UID to the serial monitor, and stores the UID as a string (userid).

Access Control Logic

```
if(userid.substring(1) == "53 FF E7 0C") {
   Serial.println("Authorised Access");
   digitalWrite(acessLed, HIGH);
   myservo.write(180);
   delay(3000);
   myservo.write(0);
   digitalWrite(acessLed, LOW);
 } else {
   Serial.println("Access Denied");
   digitalWrite(noacessLed, HIGH);
   delay(1000);
   digitalWrite(noacessLed, LOW);
 }
}
```

- If the RFID tag's UID matches the authorized UID, the servo rotates to unlock the gate, and the green LED lights up.
- If the UID doesn't match, the red LED flashes, signaling denied access.

9.5 Improvements and Expansion Ideas

- **Multiple Authorized UIDs**: Store multiple authorized UIDs for multi-user access.
- **LCD Display**: Display custom messages like "Access Granted" or "Access Denied."
- **Buzzer**: Add an audible alert for denied access attempts.
- **Data Logging**: Log access attempts to an SD card or cloud storage for security auditing.

9.6 Full Code

```
#include <SPI.h>

#include <MFRC522.h>

#include <Servo.h>


#define SS_PIN 10

#define RST_PIN 9

MFRC522 mfrc522(SS_PIN, RST_PIN);

#define servopin 7

#define acessLed 6

#define noacessLed 4

Servo myservo;


void setup() {

  myservo.attach(servopin);

  myservo.write(0);
```

```
  pinMode(7,OUTPUT);
  pinMode(6,OUTPUT);
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println("Approximate your card to the reader...");
  Serial.println();
}

void loop() {
  if (!mfrc522.PICC_IsNewCardPresent()) {
    return;
  }
  if (!mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  Serial.print("UID tag :");
  String userid = "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
    userid.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    userid.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
```

```
  Serial.println();
  userid.toUpperCase();

  if(userid.substring(1) == "53 FF E7 0C") {
    Serial.println("Authorised Access");
    digitalWrite(acessLed, HIGH);
    myservo.write(180);
    delay(3000);
    myservo.write(0);
    digitalWrite(acessLed, LOW);
  } else {
    Serial.println("Access Denied");
    digitalWrite(noacessLed, HIGH);
    delay(1000);
    digitalWrite(noacessLed, LOW);
  }
}
```

Project 10: Keypad Input System Using Arduino

10.1 Introduction

This project demonstrates how to interface a 4x4 keypad with an Arduino to capture keypresses. The keypad allows users to enter characters, which can be used in applications such as password input, menu selection, or other data entry tasks. The key presses are detected and displayed on the serial monitor.

Purpose: This project can be extended for use in various systems like security, navigation menus, or control panels that require user input.

10.2 Components Required

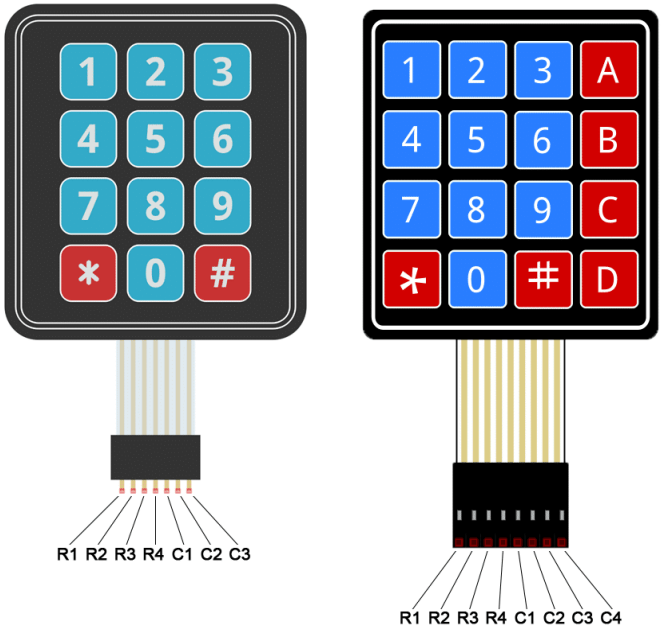
1. Arduino Uno: Microcontroller board for processing.

2. 4x4 Matrix Keypad: Input device for user interaction.

3. Jumper Wires: To connect the keypad to the Arduino.

10.3 Circuit Connections

- **Keypad Rows:**
  - Row 1 (Pin 3) → Arduino Digital Pin 3
  - Row 2 (Pin 2) → Arduino Digital Pin 2
  - Row 3 (Pin 1) → Arduino Digital Pin 1
  - Row 4 (Pin 0) → Arduino Digital Pin 0
- **Keypad Columns:**
  - Column 1 (Pin 7) → Arduino Digital Pin 7
  - Column 2 (Pin 6) → Arduino Digital Pin 6
  - Column 3 (Pin 5) → Arduino Digital Pin 5
  - Column 4 (Pin 4) → Arduino Digital Pin 4



10.4 Code Breakdown

Libraries and Definitions

```
#include <Keypad.h>
```

- The `Keypad.h` library is used to handle input from the keypad.

```
const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
```

```
};

byte rowPins[ROWS] = {3, 2, 1, 0};

byte colPins[COLS] = {7, 6, 5, 4};
```

- `ROWS` and `COLS` define the number of rows and columns in the keypad.
- `hexaKeys` stores the characters corresponding to each button on the keypad.
- `rowPins` and `colPins` arrays hold the pin numbers on the Arduino that correspond to the keypad's rows and columns.

```
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins,
ROWS, COLS);
```

- This creates an instance of the `Keypad` class to manage the input using the specified keymap and pin configurations.

Setup Function

```
void setup(){

  Serial.begin(9600);

}
```

- Serial communication is initiated to display the key presses on the serial monitor.

Loop Function

```
void loop(){

  char customKey = customKeypad.getKey();
```

```
  if (customKey){
    Serial.println(customKey);
  }
}
```

- Inside the loop, the code constantly checks if a key has been pressed using `customKeypad.getKey()`.
- If a key is pressed, it is printed to the serial monitor.

10.5 Improvements and Expansion Ideas

1. **Password Entry:** Extend the project to store and compare multi-key passwords.
2. **Menu Navigation:** Create a system where each keypress navigates through a menu.
3. **LCD Display:** Display the pressed key on an LCD screen.
4. **Timed Input:** Implement a timeout feature where the keypad will reset if no key is pressed within a specific time frame.

10.6 Full Code

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
```

```
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {3, 2, 1, 0};
byte colPins[COLS] = {7, 6, 5, 4};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins,
ROWS, COLS);

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

void loop(){
  char customKey = customKeypad.getKey();

  if (customKey){
    Serial.println(customKey);
  }
}
```

This structured format follows the same approach you provided earlier, making it easy to present future projects invariably.

Project 11: Servo motor control using Arduino and Potentiometer

---

11.1 Introduction

This project demonstrates how to interface a servo motor with an Arduino using a potentiometer to control its position. By turning the potentiometer, users can adjust the angle of the servo motor in real-time, making this setup ideal for applications such as robotics, precise positioning systems, or simple motor control tasks. The potentiometer input is mapped to the servo's rotation, and the motor's position is updated accordingly, with the values displayed on the serial monitor for feedback.

11.2 Components Required

1. Arduino Uno: Microcontroller board for processing.

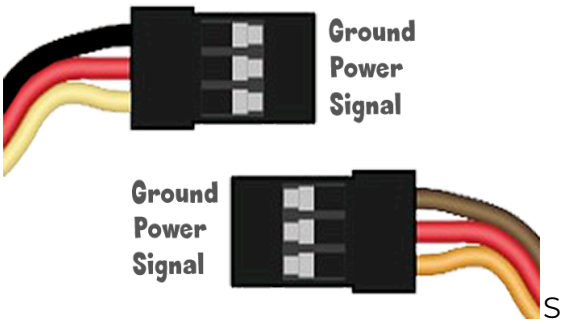
2. Servo motor: Output device for the user to see the rotation.

3. Potentiometer: input device to set the angle of rotation of the servo.

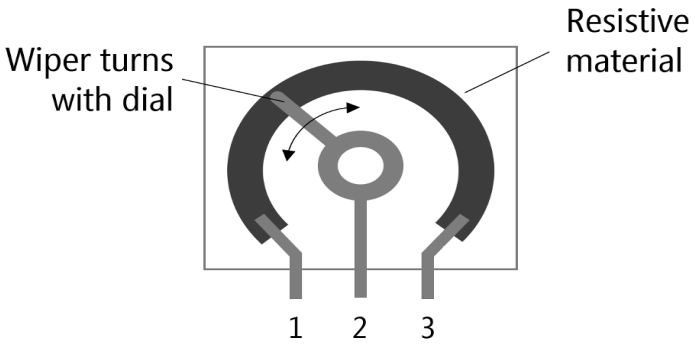
4. Jumper Wires: To connect the servo and potentiometer to the Arduino.

11.3 Circuit Connections

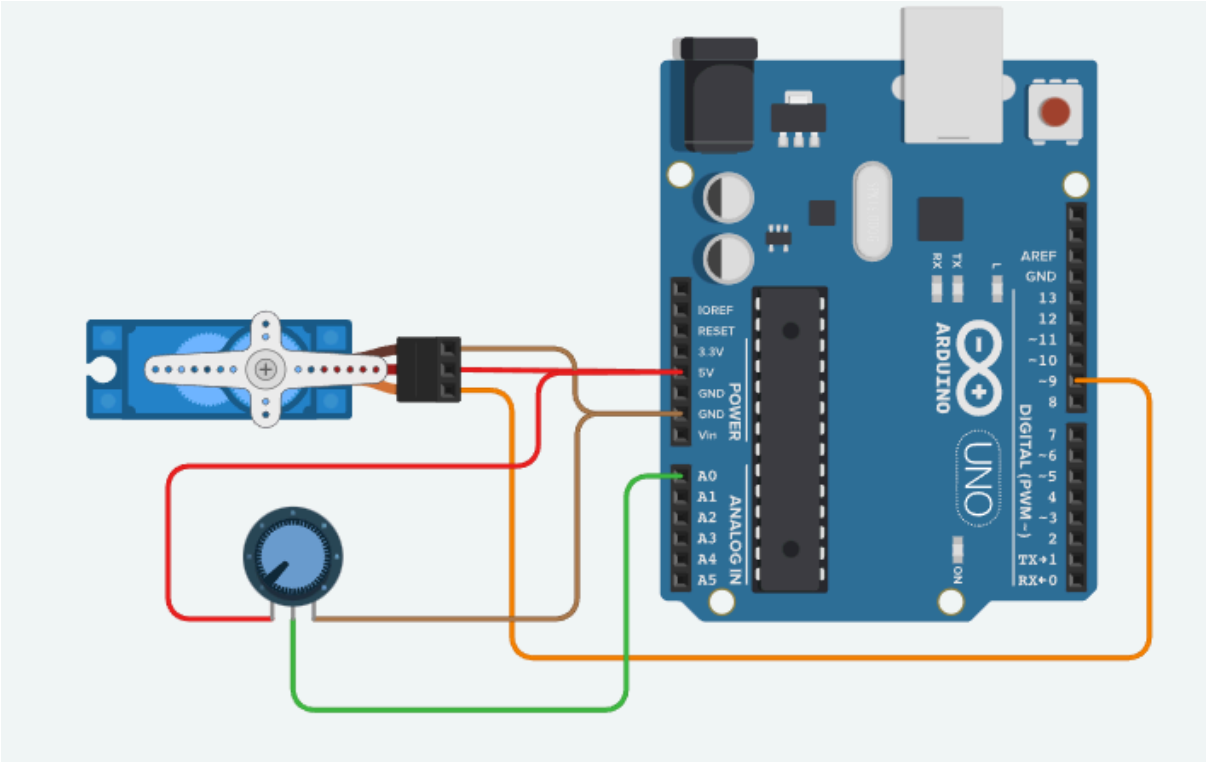
- **Servo motor:**
  - Power (Red)) → Arduino 5V out pin
  - GND (Brown)) → Arduino GND pin
  - Signal(Orange1) → Arduino Digital Pin(PWM) 9
- **Potentiometer:**
  - Fixed point 1 (Pin 1) → Arduino 5V out pin
  - Variable point (Pin 2) → Arduino Analog IN Pin A0
  - Fixed point 2 (Pin 3) → Arduino GND pin



SERVO MOTOR PINOUT



POTENTIOMETER PINOUT



11.4 Code Breakdown

Libraries and Definitions

```
#include <Servo.h>
```

- The `Servo.h` library is used to handle the servo in an easy way via Arduino.

```
Servo myServo;

int potPin = A0;

int servoPin = 9;

int potValue = 0;

int servoAngle = 0;
```

- `Servo myServo` Creates a servo object so we can use the servo in the code.
- `potPin` Stores the pin Analog pin number connected to the potentiometer.
- `servoPin` stores the pin number connected to servo signal wire.
- `potValue` stores the potentiometer values(from 0 to 1023).
- `servoAngle` stores the angle of rotation of the servo motor(from 0° to 180°)

Setup Function

```
void setup() {

  myServo.attach(servoPin);
```

```
  Serial.begin(9600);
}
```

- The `myServo.attach(servoPin);` function attaches the servo to the specified digital pin (**pin 9** in this case).
- **Serial communication** is initiated with `Serial.begin(9600);` to display the potentiometer values and servo angles on the serial monitor.

Loop Function

```
void loop() {
  potValue = analogRead(potPin);
  servoAngle = map(potValue, 0, 1023, 0, 180);
  myServo.write(servoAngle);

  Serial.print("Potentiometer Value: ");
  Serial.print(potValue);
  Serial.print(" | Servo Angle: ");
  Serial.println(servoAngle);

  delay(15);
}
```

- **Reads the potentiometer value** using `analogRead(potPin)`. This value ranges from 0 to 1023.
- **Maps the potentiometer value** to a range of 0 to 180 degrees (servo's angle) using the `map()` function.
- **Moves the servo motor** to the corresponding angle using `myServo.write(servoAngle)`.
- **Displays the potentiometer value** and the corresponding servo angle on the **serial monitor**.

11.5 Improvements and Expansion Ideas

1. **Multiple Servo Motors**: Add more potentiometers and control multiple servo motors simultaneously. This can be useful in robotic arm applications or gimbal systems where multiple axes need to be controlled.
2. **LCD Display**: Display the current potentiometer reading and corresponding servo angle on an LCD screen. This can provide a visual feedback mechanism for the user, allowing them to see the exact angle to which the servo is set.

Timed Movement: Implement a timed movement feature, where the servo gradually moves to a specified position over a set time. This could simulate smooth or delayed motion, similar to how actuators work in real-world machinery.

3. **Remote Control**: Integrate wireless communication, such as Bluetooth or RF, to control the servo remotely. You could use a smartphone or a remote control to adjust the servo position, making the system more versatile.

11.6 Full Code

```
#include <Servo.h>


Servo myServo;

int potPin = A0;
```

```
int servoPin = 9;

int potValue = 0;

int servoAngle = 0;


void setup() {
  myServo.attach(servoPin);
  Serial.begin(9600);
}


void loop() {
  potValue = analogRead(potPin);
  servoAngle = map(potValue, 0, 1023, 0, 180);
  myServo.write(servoAngle);


  Serial.print("Potentiometer Value: ");
  Serial.print(potValue);
  Serial.print(" | Servo Angle: ");
  Serial.println(servoAngle);


  delay(15);
}
```

Project 12: Remote Control of Buzzer and LED using Arduino and Infrared (IR) Sensor

---

12.1 Introduction

This project demonstrates how to control a buzzer and an LED using an IR remote and an Arduino. The IR sensor receives signals from the remote, which are then decoded by the Arduino to toggle the states of the buzzer and LED. This project can be expanded to control various electronic devices with minimal physical interaction, making it useful for home automation systems or remotely controlled devices.

12.2 Components Required

1. **Arduino Uno**: The microcontroller board for processing.
2. **IR Receiver Module**: Used to receive signals from the IR remote.
3. **IR Remote:** For sending commands to the IR receiver.
4. **Buzzer:** Output device that emits sound when triggered.
5. **LED:** Output device that lights up when activated.
6. **Jumper Wires:** To connect components to the Arduino.

12.3 Circuit Connections

- **IR Receiver:**
  - Signal Pin (Pin 1) → Arduino Analog Pin A0
  - VCC Pin (Pin 2) → Arduino 5V out pin
  - GND Pin (Pin 3) → Arduino GND pin
- **Buzzer:**
  - Positive (Red) → Arduino Digital Pin 12
  - Negative (Black) → Arduino GND pin
- **LED:**
  - Positive (Long leg) → Arduino Digital Pin 11
  - Negative (Short leg) → Arduino GND pin

12.4 Code Breakdown

Libraries and Definitions

```
#include <IRremote.h>
```

```
#define PIN A0

#define Buzz 12#include <IRremote.h>


#define PIN A0


#define Buzz 12


#define Led 11


bool buzzerstate = 0, ledstate = 0;


IRrecv irrecv(PIN);


decode_results result;

#define Led 11

bool buzzerstate = 0, ledstate = 0;

IRrecv irrecv(PIN);

decode_results result;
```

- **IRremote.h**: This library helps the Arduino decode and interpret IR signals.
- **PIN, Buzz, and Led**: These define the pin numbers for the IR receiver, buzzer, and LED, respectively.
- **buzzerstate and ledstate**: Boolean variables that store the on/off states of the buzzer and LED.
- **IRrecv irrecv(PIN)**: Initializes the IR receiver on pin A0.
- **decode_results result**: Stores the decoded results from the IR remote.

Setup Function

```
void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();
  pinMode(Buzz, OUTPUT);
  pinMode(Led, OUTPUT);
}
```

- **Serial.begin(9600)**: Initializes serial communication for debugging and feedback.
- **irrecv.enableIRIn()**: Enables the IR receiver to start decoding IR signals.
- **pinMode(Buzz, OUTPUT) and pinMode(Led, OUTPUT):** Sets the buzzer and LED as output devices.

Loop Function

```
void loop() {
  if (irrecv.decode(&result)) {
    Serial.println(result.value, HEX);
    irrecv.resume();
    delay(500);
  }
  if (result.value ==0x1FE50AF) {
    buzzerstate = !buzzerstate;
    digitalWrite(Buzz, buzzerstate);
```

```
  } else if (result.value == 0x1FED827) {

    ledstate = !ledstate;

    digitalWrite(Led,ledstate);

  }

}
```

- **irrecv.decode(&result):** Decodes the IR signal and stores the value in result.value.
- **Serial.println(result.value, HEX):** Prints the received IR signal in hexadecimal format for debugging purposes.
- **irrecv.resume():** Resets the IR receiver to continue receiving signals.
- **if (result.value == 0x1FE50AF):** Toggles the buzzer state when the specified IR code is received.
- **if (result.value == 0x1FED827):** Toggles the LED state when the specified IR code is received.

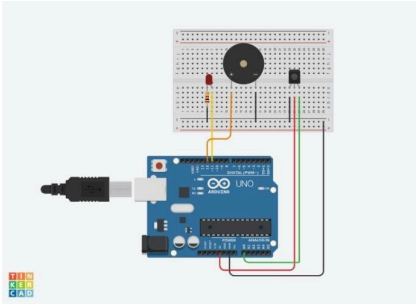
12.5 Improvements and Expansion Ideas

- **Multiple Devices:** Add more devices (such as relays, motors, or fans) controlled by different IR codes from the remote.
- **LCD Display:** Integrate an LCD screen to display the state of each device (ON/OFF) or the last IR code received.
- **IR Signal Learning:** Implement a learning function where the Arduino can learn and store new IR codes for custom devices.
- **Smart Control:** Add sensors (like a temperature sensor) that can automatically turn on/off devices like fans or heaters based on environmental conditions.
- **Remote Range Extension:** Use RF modules to extend the range of the remote control, allowing it to work over longer distances.

12.6 Full Code

```
#include <IRremote.h>

#define PIN A0

#define Buzz 12

#define Led 11

bool buzzerstate = 0, ledstate = 0;
```

```
IRrecv irrecv(PIN);
decode_results result;

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();
  pinMode(Buzz, OUTPUT);
  pinMode(Led, OUTPUT);
}

void loop() {
  if (irrecv.decode(&result)) {
    Serial.println(result.value, HEX);
    irrecv.resume();
    delay(500);
  }
  if (result.value == 0x1FE50AF) {
    buzzerstate = !buzzerstate;
    digitalWrite(Buzz, buzzerstate);
  } else if (result.value == 0x1FED827) {
    ledstate = !ledstate;
    digitalWrite(Led, ledstate);
  }
}
```



This code toggles the buzzer and LED on and off based on specific IR codes received from the remote control. The system can be expanded or modified to control additional devices or add more complex features like IR signal learning and remote communication.

Project 13: LED Control using Arduino and Joystick

13.1 Introduction

This project demonstrates how to control the brightness of four LEDs using a joystick and an Arduino. Each LED's brightness is adjusted based on the joystick's directional movements: forward, backward, left, and right. This allows for intuitive control of multiple LEDs through a single input device. The project can be expanded to include additional features such as multi-axis control or dynamic lighting effects, making it useful for interactive lighting systems, robotics, or user-controlled visual displays.

13.2 Components Required

- **Arduino Uno**: The microcontroller board to process the joystick input and control the LEDs.
- **Joystick Module**: Used to control the direction and adjust the brightness of each LED.
- **4 LEDs**: Output devices that will change brightness based on the joystick's movement.
- **Resistors (220Ω)**: To limit the current for each LED.
- **Jumper Wires**: To connect components to the Arduino.
- **Breadboard**: To assemble and connect all components.

13.3 Circuit Connections

Joystick:

- **VCC Pin** → Arduino 5V
- **GND Pin** → Arduino GND
- **X-axis Pin** → Arduino Analog Pin A0
- **Y-axis Pin** → Arduino Analog Pin A1

LED 1 (Controlled by Y-axis forward movement):

- **Positive (Long leg)** → Arduino Digital Pin 9
- **Negative (Short leg)** → Resistor (220Ω) → Arduino GND

LED 2 (Controlled by Y-axis backward movement):

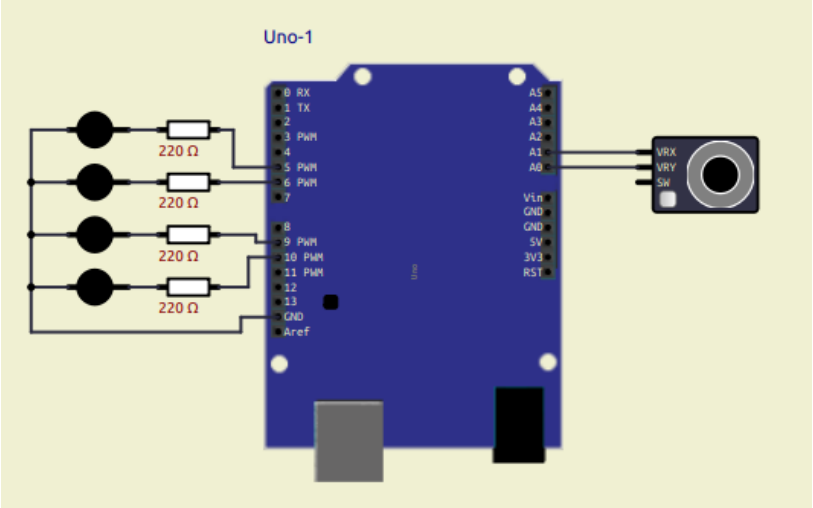
- **Positive (Long leg)** → Arduino Digital Pin 10
- **Negative (Short leg)** → Resistor (220Ω) → Arduino GND

LED 3 (Controlled by X-axis left movement):

- **Positive (Long leg)** → Arduino Digital Pin 5
- **Negative (Short leg)** → Resistor (220Ω) → Arduino GND

LED 4 (Controlled by X-axis right movement):

- **Positive (Long leg)** → Arduino Digital Pin 6
- **Negative (Short leg)** → Resistor (220Ω) → Arduino GND



13.4 Code Breakdown

Libraries and Definitions

```
const int joyXPin = A1;

const int joyYPin = A0;

const int led1 = 5;

const int led2 = 6;

const int led3 = 9;

const int led4 = 10;

int xValue = 0;

int yValue = 0;
```

- **joyXPin & joyYPin**: these defines the Analog pin inputs number from the joystick.
- **Led1,led2,led3,led3:** these defines the digital pin number that connects to the led.
- **xValue & yValue:** these variables stores the current position of the joystick.

Setup Function

```
void setup( ) {
 pinMode(led1, OUTPUT);
 pinMode(led2, OUTPUT);
 pinMode(led3, OUTPUT);
 pinMode(led4, OUTPUT);
 Serial.begin(9600);
}
```

- **Serial.begin(9600)**: Initializes serial communication for debugging and feedback.
- **pinMode(led1, OUTPUT) , pinMode(led2, OUTPUT),.....:** Sets the corresponding pins on the arduino in output mode.

Loop Function

```
void loop() {
 xValue = analogRead(joyXPin);
 yValue = analogRead(joyYPin);
 int led1Brightness = map(yValue, 511, 0, 0, 255);
 int led2Brightness = map(yValue, 512, 1023, 0, 255);
```

```
 int led3Brightness = map(xValue, 511, 0, 0, 255);
 int led4Brightness = map(xValue, 512, 1023, 0, 255);
 if(xValue>0 && xValue<=511){
   analogWrite(led3, led3Brightness);
   analogWrite(led4, 0);
 }
 if(xValue>511 && xValue<=1023){
   analogWrite(led4, led4Brightness);
   analogWrite(led3, 0);
 }
 if(yValue>0 && xValue<=511){
   analogWrite(led1, led1Brightness);
   analogWrite(led2, 0);
 }
 if(yValue>511 && xValue<=1023){
   analogWrite(led2, led2Brightness);
   analogWrite(led1, 0);
 }
//optional
  Serial.print("X: ");
 Serial.print(xValue);
 Serial.print(" Y: ");
 Serial.print(yValue);
 Serial.print(" | LED1: ");
 Serial.print(led1Brightness);
```

```
 Serial.print(" LED2: ");
 Serial.print(led2Brightness);
 Serial.print(" LED3: ");
 Serial.print(led3Brightness);
 Serial.print(" LED4: ");
 Serial.println(led4Brightness);
 delay(100);
}
```

- **`xValue = analogRead(joyXPin);` & `yValue = analogRead(joyYPin);`**: Reads the x-axis and y-axis values from the joystick.
- **`map(yValue, 511, 0, 0, 255);` & `map(xValue, 511, 0, 0, 255);`**: Maps the joystick values from the range of 511-0 to 0-255 for LED brightness control.
- **`analogWrite(led3, led3Brightness);` & `analogWrite(led4, 0);`**: If the joystick is moved left (x-axis <= 511), controls the brightness of LED 3 while turning off LED 4.
- **`analogWrite(led4, led4Brightness);` & `analogWrite(led3, 0);`**: If the joystick is moved right (x-axis > 511), controls the brightness of LED 4 while turning off LED 3.
- **`analogWrite(led1, led1Brightness);` & `analogWrite(led2, 0);`**: If the joystick is moved forward (y-axis <= 511), controls the brightness of LED 1 while turning off LED 2.
- **`analogWrite(led2, led2Brightness);` & `analogWrite(led1, 0);`**: If the joystick is moved backward (y-axis > 511), controls the brightness of LED 2 while turning off LED 1.
- **`Serial.print()`**: Outputs the current joystick values and the brightness of all LEDs for debugging.
- **`delay(100);`**: Adds a short delay to reduce the frequency of updates.

13.5 Improvements and Expansion Ideas

- **Multiple LEDs or RGB LEDs**: Expand the project by adding more LEDs or using RGB LEDs for multi-color control based on joystick movements.

- **LCD Display**: Integrate an LCD to display the joystick's position and the brightness level of each LED.
- **Multi-Axis Control**: Add another joystick for 3D movement control, allowing additional functions such as color or speed control in robotics or lighting systems.
- **Automatic Brightness Adjustments**: Implement sensors (like light or temperature sensors) to automatically adjust the brightness of LEDs based on environmental conditions.
- **Wireless Joystick Control**: Use wireless communication (like RF or Bluetooth modules) to remotely control the LEDs using a joystick from a distance.

13.6 Full Code

```
const int joyXPin = A1;

const int joyYPin = A0;

const int led1 = 5;

const int led2 = 6;

const int led3 = 9;

const int led4 = 10;

int xValue = 0;

int yValue = 0;

void setup() {

 pinMode(led1, OUTPUT);

 pinMode(led2, OUTPUT);

 pinMode(led3, OUTPUT);

 pinMode(led4, OUTPUT);

 Serial.begin(9600);
```

```
}
void loop() {
 xValue = analogRead(joyXPin);
 yValue = analogRead(joyYPin);
 int led1Brightness = map(yValue, 511, 0, 0, 255);
 int led2Brightness = map(yValue, 512, 1023, 0, 255);
 int led3Brightness = map(xValue, 511, 0, 0, 255);
 int led4Brightness = map(xValue, 512, 1023, 0, 255);
 if(xValue>0 && xValue<=511){
   analogWrite(led3, led3Brightness);
   analogWrite(led4, 0);
 }
 if(xValue>511 && xValue<=1023){
   analogWrite(led4, led4Brightness);
   analogWrite(led3, 0);
 }
 if(yValue>0 && xValue<=511){
   analogWrite(led1, led1Brightness);
   analogWrite(led2, 0);
 }
 if(yValue>511 && xValue<=1023){
   analogWrite(led2, led2Brightness);
   analogWrite(led1, 0);
 }
  Serial.print("X: ");
```

```
 Serial.print(xValue);
 Serial.print(" Y: ");
 Serial.print(yValue);
 Serial.print(" | LED1: ");
 Serial.print(led1Brightness);
 Serial.print(" LED2: ");
 Serial.print(led2Brightness);
 Serial.print(" LED3: ");
 Serial.print(led3Brightness);
 Serial.print(" LED4: ");
 Serial.println(led4Brightness);
 delay(100);
}
```

Project 14: Temperature and Humidity-Based LED Control using DHT11 and Arduino

---

14.1 Introduction

This project demonstrates how to control the brightness of an LED based on temperature and humidity readings from a DHT11 sensor using an Arduino. As the temperature increases, the LED becomes brighter, and similarly, it dims as the temperature decreases. This project is a practical way to visualize environmental conditions and can be extended to control other devices like fans or heaters based on temperature and humidity levels.

14.2 Components Required

- **Arduino Uno:** The microcontroller board to process the sensor input and control the LED.
- **DHT11 Sensor:** Measures the temperature and humidity.
- **LED:** The output device whose brightness is controlled based on the temperature readings.
- **Resistor (220Ω):** To limit the current for the LED.
- **Jumper Wires:** To connect components to the Arduino.
- **Breadboard:** To assemble and connect all components.

14.3 Circuit Connections

- **DHT11 Sensor:**
  - VCC Pin → Arduino 5V
  - GND Pin → Arduino GND
  - Data Pin → Arduino Digital Pin 2
- **LED:**
  - Positive (Long leg) → Arduino Digital Pin 9
  - Negative (Short leg) → Resistor (220Ω) → Arduino GND

14.4 Code Breakdown

Libraries and Definitions

```
#include "DHT.h"

#define DHTPIN 2       // Pin where DHT11 is connected

#define DHTTYPE DHT11   // Define DHT type as DHT11

#define LEDPIN 9       // Pin connected to LED
```

```
DHT dht(DHTPIN, DHTTYPE);  // Initialize DHT sensor

int temperature = 0;
```

- **DHTPIN & DHTTYPE:** These define the digital pin connected to the DHT11 sensor and specify the sensor type.
- **LEDPIN:** This defines the digital pin connected to the LED.
- **temperature:** Stores the temperature reading from the DHT11.

Setup Function

```
void setup() {
  pinMode(LEDPIN, OUTPUT);   // Set the LED pin as an output
  Serial.begin(9600);        // Initialize serial communication
  dht.begin();               // Start the DHT11 sensor
}
```

- **pinMode(LEDPIN, OUTPUT):** Sets the LED pin as an output.
- **Serial.begin(9600):** Initializes serial communication for debugging.
- **dht.begin():** Initializes the DHT sensor to start reading values.

Loop Function

```
void loop() {
  // Read temperature in Celsius
  float temperature = dht.readTemperature();
```

```
  // Check if reading failed
  if (isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Map the temperature to LED brightness (0-255)
  int brightness = map(temperature, 15, 40, 0, 255);

  // Limit brightness to the valid range (0-255)
  brightness = constrain(brightness, 0, 255);

  // Set LED brightness
  analogWrite(LEDPIN, brightness);

  // Print temperature and brightness for debugging
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, LED Brightness: ");
  Serial.println(brightness);

  delay(2000);  // Wait for 2 seconds before next reading
}
```

- **dht.readTemperature():** Reads the current temperature from the DHT11 sensor.
- **map():** Maps the temperature range (15°C to 40°C) to an appropriate LED brightness (0 to 255).
- **analogWrite():** Adjusts the LED brightness based on the mapped value.
- **Serial.print():** Displays the temperature and LED brightness for debugging purposes.
- **delay(2000):** Waits 2 seconds before taking the next reading.

14.5 Improvements and Expansion Ideas

- **Humidity Control:** Use the humidity readings from the DHT11 to control additional LEDs or other devices like fans or humidifiers.
- **Multiple LEDs:** Expand the project by adding more LEDs for different ranges of temperature and humidity.
- **LCD Display:** Integrate an LCD to display real-time temperature and humidity values.
- **Automation:** Add relays to control fans or other electrical appliances based on environmental conditions.

14.6 Full Code

```
#include "DHT.h"

#define DHTPIN 2       // Pin where DHT11 is connected
#define DHTTYPE DHT11   // Define DHT type as DHT11
#define LEDPIN 9       // Pin connected to LED

DHT dht(DHTPIN, DHTTYPE);  // Initialize DHT sensor

void setup() {
  pinMode(LEDPIN, OUTPUT);   // Set the LED pin as an output
```

```
  Serial.begin(9600);       // Initialize serial communication
  dht.begin();              // Start the DHT11 sensor
}

void loop() {
  // Read temperature in Celsius
  float temperature = dht.readTemperature();

  // Check if reading failed
  if (isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Map the temperature to LED brightness (0-255)
  int brightness = map(temperature, 15, 40, 0, 255);

  // Limit brightness to the valid range (0-255)
  brightness = constrain(brightness, 0, 255);

  // Set LED brightness
  analogWrite(LEDPIN, brightness);

  // Print temperature and brightness for debugging
  Serial.print("Temperature: ");
```

```
  Serial.print(temperature);
  Serial.print(" °C, LED Brightness: ");
  Serial.println(brightness);

  delay(2000);  // Wait for 2 seconds before next reading
}
```

This project controls the LED brightness using real-time temperature readings, and you can further build on it for more complex systems.

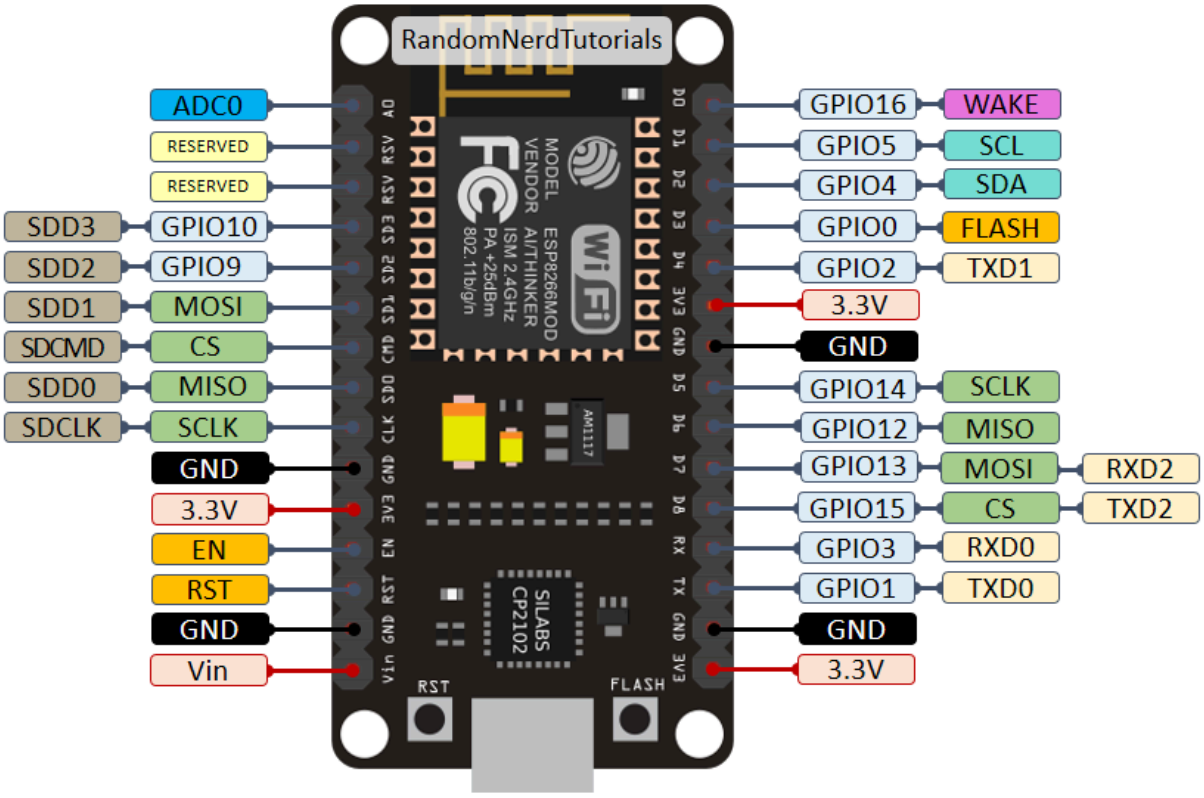
THE IOT SECTION

Discover the Power of NodeMCU: A Beginner's Guide

NodeMCU: Your Gateway to IoT Innovation

What is NodeMCU?

NodeMCU is an open-source IoT platform that integrates an ESP8266 Wi-Fi module with easy-to-use hardware and software. It's designed for IoT applications, making it perfect for connecting your projects to the internet. NodeMCU is versatile and widely used in DIY projects, IoT solutions, and smart home automation, offering a simple interface for both beginners and advanced users.



Key Components:

- **ESP8266 Microcontroller:** The core of NodeMCU, responsible for processing inputs, controlling outputs, and providing Wi-Fi capabilities.

- **Digital I/O Pins:** 16 pins that can be used as either inputs or outputs for digital signals, allowing interaction with external devices.
- **Analog Input Pin (A0):** A single analog pin for reading analog sensor values.
- **Micro-USB Port:** Used for powering and programming the NodeMCU from a computer.
- **Reset Button:** Resets the microcontroller to restart the running program.
- **Flash Button:** Used during programming and uploading code to the board.

Additional Features:

- **Wi-Fi Connectivity:** NodeMCU's built-in ESP8266 module enables easy internet access, making it ideal for IoT projects.
- **PWM Support:** NodeMCU provides Pulse Width Modulation on certain digital pins, allowing control of devices like motors, servos, and LEDs.
- **Breadboard-Friendly:** The compact design fits perfectly on a standard breadboard for easy prototyping.
- **Low Power Consumption:** Ideal for battery-powered projects, NodeMCU offers efficient power usage, essential for long-running IoT applications.

Applications:

- **IoT Projects:** Easily connect sensors and devices to the internet for remote monitoring and control.
- **Home Automation:** Build smart home systems that control lights, appliances, and more via Wi-Fi.
- **Prototyping:** Rapidly design, test, and deploy IoT prototypes with minimal hardware requirements.
- **DIY Projects:** Create custom internet-connected gadgets and interactive installations.
- **STEM Education:** Teach and learn about IoT concepts, networking, and embedded systems.

Why Choose NodeMCU?

NodeMCU offers an accessible entry into the world of IoT with its built-in Wi-Fi capabilities and robust community support. With countless tutorials, resources, and a growing user base, it's easy to find guidance for your projects. Its affordability, versatility, and ease of use make NodeMCU the go-to choice for IoT beginners and innovators alike.

Installing ESP8266 in Arduino IDE: A Step-by-Step Guide

To get started with NodeMCU on the Arduino IDE, you'll first need to install the ESP8266 board. This will allow the IDE to recognize and program your NodeMCU board.

Step 1: Open Arduino IDE

- Ensure you have the latest version of the Arduino IDE installed. You can download it from the official Arduino website (https://www.arduino.cc).

Step 2: Add ESP8266 Board URL

- Open the **Arduino IDE**.
- Go to **File > Preferences**.

In the **Additional Board Manager URLs** field, enter the following URL:

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

- Click **OK** to save.

Step 3: Open the Board Manager

- Go to **Tools > Board > Boards Manager**.
- In the search bar, type "ESP8266".
- Find the package labeled **"ESP8266 by ESP8266 Community"** and click **Install**.

Step 4: Select NodeMCU Board

- Once the installation is complete, go to **Tools > Board** and scroll down until you see **NodeMCU 1.0 (ESP-12E Module)**.
- Select this option to configure the IDE for programming the NodeMCU board.

Step 5: Connect NodeMCU to Your Computer

- Use a micro-USB cable to connect your NodeMCU to the computer.
- The Arduino IDE should automatically detect the correct port. If not, go to **Tools > Port** and select the port corresponding to your NodeMCU.

Step 6: Upload Your First Sketch

- To verify everything is working correctly, upload a basic sketch like the **Blink** example.
    - Go to **File > Examples > 01.Basics > Blink**.
    - Modify the pin to match **D0** (GPIO16) if needed.
    - Click **Upload** to program the board.

Project 15: LED Blink with NodeMCU

15.1 Introduction

This project demonstrates how to blink an LED using NodeMCU. It's a basic yet essential project for beginners to understand the fundamental operations of controlling digital pins on the NodeMCU using the Arduino IDE.

15.2 Components Required

- NodeMCU (ESP8266)
- LED (1x)
- Resistor (220 ohms)
- Breadboard
- Jumper wires
- USB cable

15.3 Circuit Connections

1. Connect the positive leg (anode) of the LED to GPIO 2 (D4) of NodeMCU.
2. Connect the negative leg (cathode) of the LED to the ground (GND) pin of the NodeMCU through a 220-ohm resistor.
3. Power the NodeMCU via USB connection to your computer.

15.4 Code Breakdown

15.4.1 Libraries and Definitions No additional libraries are required for this simple LED blink project.

15.4.2 Setup Function

cpp

Copy code

```
void setup() {

  pinMode(2, OUTPUT);  // Set GPIO 2 (D4) as an output pin
```

```
}
```

15.4.3 Loop Function

cpp

Copy code

```
void loop() {
  digitalWrite(2, HIGH);  // Turn the LED on
  delay(1000);            // Wait for 1 second
  digitalWrite(2, LOW);   // Turn the LED off
  delay(1000);            // Wait for 1 second
}
```

15.5 Improvements and Expansion Ideas

- **Multiple LEDs:** Expand the project to control multiple LEDs with different timings.
- **Button-Controlled LED:** Add a push-button to manually control when the LED should blink.
- **PWM LED Control:** Use PWM to control the brightness of the LED by modifying the duty cycle of the signal.

15.6 Full Code

cpp

Copy code

```
void setup() {
  pinMode(2, OUTPUT);  // Set GPIO 2 (D4) as output
}
```

```
void loop() {
  digitalWrite(2, HIGH);  // Turn LED on
  delay(1000);            // Wait for 1 second
  digitalWrite(2, LOW);   // Turn LED off
  delay(1000);            // Wait for 1 second
}
```

How to Set Up Arduino IoT Cloud

Arduino IoT Cloud is a platform designed for controlling and monitoring your Arduino devices from anywhere using the internet. Follow these steps to get started:

1. Create an Arduino Account

To access the Arduino IoT Cloud, you need to create an account:

- Visit Arduino IoT Cloud.
- Click on **"Get Started"** and either log in or create a new account.

2. Install Arduino Create Agent

The Arduino Create Agent is a plugin that allows your browser to communicate with the connected Arduino hardware:

- Once logged in, you'll be prompted to install the **Arduino Create Agent**.
- Follow the on-screen instructions to download and install the agent on your computer.

3. Add a New Device

Now it's time to connect your Arduino device to the cloud:

- In the Arduino IoT Cloud dashboard, click on **"Devices"** and then **"Add Device"**.
- Select **"Set up a Third Party Device"** if you are using a NodeMCU or ESP8266.
- If you have an Arduino board, select it from the list (e.g., Arduino Nano 33 IoT, MKR WiFi 1010).

4. Configure Your Device

After adding the device, configure the board settings:

- Set up a **Thing** (a virtual representation of your device in the cloud).
- Name your device and assign a unique ID to it.

- Choose the **Wi-Fi credentials** (SSID and password) that will allow the board to connect to the cloud.

5. Set Up Variables

The variables represent the data that your device will send or receive from the cloud:

- Go to the **"Variables"** section in your Thing.
- Add variables like temperature, humidity, or any other sensor data, defining their type (int, float, etc.).

6. Write and Upload the Code

Arduino IoT Cloud automatically generates a sketch for you, but you can customize it:

- In the **Sketch** section, edit the code as needed for your project.
- Click on **"Upload"** to send the code to your connected device via the Arduino IDE or Web Editor.

7. Create a Dashboard

To monitor and control your device remotely, create a custom dashboard:

- Navigate to the **"Dashboards"** section and click **"Create Dashboard"**.
- Add widgets such as buttons, sliders, or gauges to visualize your device's data.

8. Control and Monitor Your Device

Once everything is set up, you can monitor and control your Arduino device from the cloud:

- Use the **Arduino IoT Cloud** dashboard from any internet-connected device (PC, phone, tablet).
- You can turn devices on or off, check sensor readings, or receive alerts.

9. Optional: Link to Alexa/Google Assistant

For voice control integration:

- Link your Arduino IoT Cloud account to **Amazon Alexa** or **Google Assistant**.
- Set up specific voice commands to interact with your devices.

Why Choose Arduino IoT Cloud?

- **Ease of Use**: Simple drag-and-drop interface for beginners.
- **Remote Control**: Manage your devices from anywhere in the world.
- **Integration**: Works with Alexa, Google Assistant, and more for seamless smart home integration.

Project 16: Arduino IoT Cloud with NodeMCU and DHT11

In this project, we'll use **NodeMCU**, **DHT11 sensor**, and **Arduino IoT Cloud** to remotely monitor temperature and humidity.

16.1 Introduction

This project demonstrates how to connect a **DHT11 temperature and humidity sensor** to a **NodeMCU** board and send the sensor data to the **Arduino IoT Cloud**. You will be able to monitor real-time environmental data remotely through the Arduino IoT Cloud Dashboard.

16.2 Components Required

- **NodeMCU** (ESP8266)
- **DHT11 Temperature and Humidity Sensor**
- Jumper Wires
- Breadboard
- **Arduino Create Agent** installed on your PC
- Arduino IoT Cloud account

16.3 Circuit Connections

Connect the components as shown:

- **DHT11 VCC** → 3.3V (NodeMCU)
- **DHT11 GND** → GND (NodeMCU)
- **DHT11 Data Pin** → D4 (GPIO 2) on NodeMCU

16.4 Setup in Arduino IoT Cloud

Step 1: Create an Account

- Go to Arduino IoT Cloud and log in or create an account.

Step 2: Install Arduino Create Agent

- Download and install the **Arduino Create Agent** as prompted.

Step 3: Add NodeMCU to the Cloud

- In the Arduino IoT Cloud dashboard, click **Devices** → **Add Device** → **Set up a Third Party Device**.
- Select **ESP8266** and follow the steps to connect your NodeMCU.

Step 4: Create a Thing

- Go to **Things** → **Create New Thing** and name it “DHT11 Monitor.”
- Add a **DHT11 Temperature** and **Humidity** variable, setting the data type to **float**.

16.5 Code Breakdown

Arduino IoT Cloud will auto-generate code based on the variables, but you need to modify it for the **DHT11 sensor**:

cpp

Copy code

```
#include "thingProperties.h"

#include <DHT.h>


#define DHTPIN D4      // Pin where DHT11 is connected

#define DHTTYPE DHT11  // DHT11 sensor type


DHT dht(DHTPIN, DHTTYPE);


void setup() {

  // Initialize serial and cloud connection

  Serial.begin(115200);

  initProperties();

  ArduinoCloud.begin(ArduinoIoTPreferredConnection);


  // Sync with cloud

  setDebugMessageLevel(2);

  ArduinoCloud.printDebugInfo();
```

```
  // Start DHT11 sensor
  dht.begin();
}

void loop() {
  // Update cloud and sensor readings
  ArduinoCloud.update();

  // Read temperature and humidity
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Check if readings are valid
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Send values to the cloud
  temp = temperature;
  hum = humidity;
}
```

Explanation of Code:

- **thingProperties.h** handles the connection to the cloud and defines the variables (temperature and humidity).
- The **DHT library** reads temperature and humidity from the sensor.
- The values are sent to Arduino IoT Cloud by assigning them to the cloud variables.

16.6 Create Dashboard

1. Go to **Dashboards** in Arduino IoT Cloud and create a new dashboard.
2. Add widgets like **gauges** or **graphs** to display the temperature and humidity data in real-time.
3. Link these widgets to the variables you created in the cloud.

16.7 Improvements and Expansion Ideas

- Add an **alert system**: Get notifications when the temperature or humidity crosses a certain threshold.
- Integrate with **Alexa/Google Assistant**: Control or monitor your system with voice commands.
- Add more sensors (e.g., light, motion) to expand your project for a full home automation setup.

16.8 Full Code

cpp

Copy code

```
#include "thingProperties.h"

#include <DHT.h>


#define DHTPIN D4

#define DHTTYPE DHT11


DHT dht(DHTPIN, DHTTYPE);


void setup() {
```

```
  Serial.begin(115200);
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  dht.begin();
}

void loop() {
  ArduinoCloud.update();

  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  temp = temperature;
  hum = humidity;
}
```

Conclusion

This collection of Arduino-based projects offers a diverse range of practical applications and learning opportunities. From fundamental tasks like blinking LEDs to more complex systems such as RFID-based access control, each project is designed to enhance your understanding of electronics, programming, and system integration.

By exploring these projects, you'll gain valuable hands-on experience and insights into real-world applications of Arduino technology. Whether you're a beginner looking to get started with electronics or an experienced maker seeking new challenges, these projects provide a solid foundation and inspiration for further experimentation and innovation.

We encourage you to dive into each project, experiment with different components, and adapt the designs to suit your needs. As you progress, you'll develop problem-solving skills, technical knowledge, and a deeper appreciation for the possibilities that Arduino and electronics offer.

Thank you for exploring these projects, and we hope they spark your creativity and enthusiasm for electronics and programming.

Happy building!

Prepared by:

Learn Lab Team