Some Notes

On

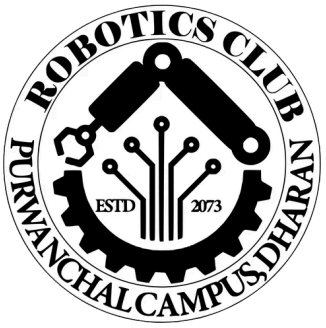
Day 4: SENSORS

By

Aditya
PUR081BAG002

To

RObotics club



Tribhuwan University

Institute of Engineering

Robotics Club

Purwanchal Campus

Dharan, Nepal

2082-3-11

Arduino Sensors

1. Introduction

Arduino microcontrollers have revolutionized the world of electronics prototyping and IoT development. One of the key aspects that makes Arduino so powerful is its ability to interface with a wide variety of sensors. Sensors are electronic components that detect and respond to physical inputs from the environment, converting them into electrical signals that can be processed by the Arduino.

This document provides a comprehensive overview of commonly used sensors with Arduino, including detailed code examples and practical applications. Each sensor section includes wiring diagrams descriptions, code implementations, and real-world use cases.

2. Temperature and Humidity Sensors

2.1. DHT22 Temperature and Humidity Sensor

The DHT22 sensor is a digital sensor that measures both temperature and humidity with high accuracy. It communicates with Arduino using a single-wire digital interface.

Specifications:

- Temperature range: −40°C to +125°C
- Humidity range: 0% to 100% RH
- Accuracy: ±0.5°C for temperature, ±2% RH for humidity
- Operating voltage: 3.3V to 5V

Wiring:

- VCC ⟶ 5V
- GND ⟶ GND
- Data ⟶ Digital Pin 2

```
#include <DHT.h>

#define DHTPIN 2
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
  Serial.println("DHT22 Sensor Initialized");
}

void loop() {
  delay(2000);

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

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

  Serial.print("Humidity: ");
```

```
  Serial.print(humidity);
  Serial.print("%  Temperature: ");
  Serial.print(temperature);
  Serial.println("°C");

  // Calculate heat index
  float heatIndex = dht.computeHeatIndex(temperature, humidity, false);
  Serial.print("Heat Index: ");
  Serial.print(heatIndex);
  Serial.println("°C");
}
```

2.2. DS18B20 Waterproof Temperature Sensor

The DS18B20 is a digital temperature sensor that provides high-precision temperature readings. It uses the OneWire protocol and can operate in parasitic power mode.

Specifications:

- Temperature range: −55°C to +125°C
- Accuracy: ±0.5°C
- Resolution: 9 to 12 bits (configurable)
- Multiple sensors can share the same data line

Wiring:

- Red wire ⟶ 5V
- Black wire ⟶ GND
- Yellow/Blue wire ⟶ Digital Pin 2
- 4.7kΩ resistor between data line and VCC

```
#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 2
#define TEMPERATURE_PRECISION 12

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

void setup() {
  Serial.begin(9600);
  sensors.begin();
  sensors.setResolution(TEMPERATURE_PRECISION);
  Serial.println("DS18B20 Temperature Sensor Ready");
}

void loop() {
  sensors.requestTemperatures();

  float temperature = sensors.getTempCByIndex(0);

  if (temperature == DEVICE_DISCONNECTED_C) {
    Serial.println("Error: Could not read temperature data");
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println("°C");
```

```
  delay(1000);
}
```

3. Motion and Distance Sensors

3.1. HC-SR04 Ultrasonic Distance Sensor

The HC-SR04 is an ultrasonic ranging module that provides non-contact distance measurements from 2cm to 4m with ranging accuracy of 3mm.

Specifications:

- Operating voltage: 5V DC
- Operating current: 15mA
- Ranging distance: 2cm to 400cm
- Measuring angle: 15°
- Ultrasonic frequency: 40kHz

Wiring:

- VCC → 5V
- GND → GND
- Trig → Digital Pin 9
- Echo → Digital Pin 10

```
#define TRIG_PIN 9
#define ECHO_PIN 10

long duration;
float distance;

void setup() {
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  Serial.println("HC-SR04 Ultrasonic Sensor Ready");
}

void loop() {
  // Clear the trigger pin
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);

  // Send ultrasonic pulse
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Read the echo pin
  duration = pulseIn(ECHO_PIN, HIGH);

  // Calculate distance in cm
  distance = duration * 0.034 / 2;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Object detection alert
  if (distance < 10) {
    Serial.println("WARNING: Object detected nearby!");
```

```
  }

  delay(500);
}
```

3.2. PIR Motion Sensor

The PIR (Passive Infrared) sensor detects motion by measuring changes in infrared radiation. It's commonly used in security systems and automatic lighting.

Specifications:

- Operating voltage: 5V to 12V
- Detection range: up to 7 meters
- Detection angle: 120°
- Delay time: adjustable (0.3s to 18s)

Wiring:

- VCC → 5V
- GND → GND
- OUT → Digital Pin 2

```
#define PIR_PIN 2
#define LED_PIN 13

int pirState = LOW;
int val = 0;

void setup() {
  Serial.begin(9600);
  pinMode(PIR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  Serial.println("PIR Motion Sensor Ready");
  Serial.println("Warming up sensor...");
  delay(30000); // Allow sensor to stabilize
}

void loop() {
  val = digitalRead(PIR_PIN);

  if (val == HIGH) {
    if (pirState == LOW) {
      Serial.println("Motion detected!");
      digitalWrite(LED_PIN, HIGH);
      pirState = HIGH;
    }
  } else {
    if (pirState == HIGH) {
      Serial.println("Motion ended!");
      digitalWrite(LED_PIN, LOW);
      pirState = LOW;
    }
  }

  delay(100);
}
```

4. Light and Color Sensors

4.1. LDR (Light Dependent Resistor)

An LDR is a passive component whose resistance decreases with increasing light intensity. It's commonly used in automatic lighting systems and light meters.

Specifications:

- Resistance in darkness: 1MΩ
- Resistance in light: 10-20kΩ
- Response time: 20-30ms
- Peak sensitivity: 540nm (green light)

Wiring:

- One leg ⟶ 5V
- Other leg ⟶ A0 and 10kΩ resistor to GND

```
#define LDR_PIN A0
#define LED_PIN 13

int ldrValue = 0;
int threshold = 500;

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  Serial.println("LDR Light Sensor Ready");
}

void loop() {
  ldrValue = analogRead(LDR_PIN);

  Serial.print("Light Level: ");
  Serial.print(ldrValue);

  if (ldrValue < threshold) {
    Serial.println(" - Dark, LED ON");
    digitalWrite(LED_PIN, HIGH);
  } else {
    Serial.println(" - Bright, LED OFF");
    digitalWrite(LED_PIN, LOW);
  }

  delay(500);
}
```

4.2. BH1750 Digital Light Sensor

The BH1750 is a digital light sensor that provides accurate lux measurements using I2C communication protocol.

Specifications:

- Measurement range: 1 to 65535 lux
- High resolution: 1 lux
- Operating voltage: 2.4V to 3.6V
- I2C interface

Wiring:

- VCC ⟶ 3.3V
- GND ⟶ GND

- SCL → A5
- SDA → A4

```
#include <Wire.h>
#include <BH1750.h>

BH1750 lightMeter;

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

  if (lightMeter.begin()) {
    Serial.println("BH1750 Light Sensor Ready");
  } else {
    Serial.println("Error initializing BH1750");
  }
}

void loop() {
  float lux = lightMeter.readLightLevel();

  Serial.print("Light Level: ");
  Serial.print(lux);
  Serial.println(" lux");

  // Classify light conditions
  if (lux < 10) {
    Serial.println("Condition: Dark");
  } else if (lux < 100) {
    Serial.println("Condition: Dim");
  } else if (lux < 1000) {
    Serial.println("Condition: Bright");
  } else {
    Serial.println("Condition: Very Bright");
  }

  delay(1000);
}
```

5. Environmental Sensors

5.1. MQ-2 Gas Sensor

The MQ-2 sensor detects various combustible gases including LPG, propane, methane, alcohol, hydrogen, and smoke.

Specifications:
- Operating voltage: 5V
- Detection range: 200-10000ppm
- Response time: 10s
- Preheat time: 20s

Wiring:
- VCC → 5V
- GND → GND
- A0 → A0
- D0 → Digital Pin 2

```
#define MQ2_ANALOG_PIN A0
#define MQ2_DIGITAL_PIN 2
#define BUZZER_PIN 8

int gasLevel = 0;
int gasThreshold = 400;

void setup() {
  Serial.begin(9600);
  pinMode(MQ2_DIGITAL_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  Serial.println("MQ-2 Gas Sensor Ready");
  Serial.println("Preheating sensor...");
  delay(20000); // Preheat time
}

void loop() {
  gasLevel = analogRead(MQ2_ANALOG_PIN);
  int digitalValue = digitalRead(MQ2_DIGITAL_PIN);

  Serial.print("Gas Level: ");
  Serial.print(gasLevel);
  Serial.print(" | Digital: ");
  Serial.println(digitalValue);

  if (gasLevel > gasThreshold || digitalValue == HIGH) {
    Serial.println("WARNING: Gas detected!");
    digitalWrite(BUZZER_PIN, HIGH);
    delay(1000);
    digitalWrite(BUZZER_PIN, LOW);
  }

  delay(1000);
}
```

5.2. BME280 Environmental Sensor

The BME280 is a high-precision environmental sensor that measures temperature, humidity, and barometric pressure.

Specifications:

- Temperature range: −40°C to +85°C
- Humidity range: 0% to 100% RH
- Pressure range: 300 to 1100 hPa
- I2C and SPI interface

Wiring:

- VCC ⟶ 3.3V
- GND ⟶ GND
- SCL ⟶ A5
- SDA ⟶ A4

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme;

void setup() {
```

```
  Serial.begin(9600);

  if (!bme.begin(0x76)) {
    Serial.println("Could not find BME280 sensor!");
    while (1);
  }

  Serial.println("BME280 Environmental Sensor Ready");
}

void loop() {
  float temperature = bme.readTemperature();
  float humidity = bme.readHumidity();
  float pressure = bme.readPressure() / 100.0F;
  float altitude = bme.readAltitude(1013.25);

  Serial.println("--- Environmental Data ---");
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println("°C");

  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println("%");

  Serial.print("Pressure: ");
  Serial.print(pressure);
  Serial.println(" hPa");

  Serial.print("Altitude: ");
  Serial.print(altitude);
  Serial.println(" m");

  Serial.println();
  delay(2000);
}
```

6. Accelerometer and Gyroscope Sensors

6.1. MPU6050 6-Axis Motion Sensor

The MPU6050 combines a 3-axis gyroscope and 3-axis accelerometer with a Digital Motion Processor (DMP).

Specifications:

- 3-axis gyroscope: ±250, ±500, ±1000, ±2000°/sec
- 3-axis accelerometer: ±2g, ±4g, ±8g, ±16g
- I2C interface
- Built-in temperature sensor

Wiring:

- VCC → 3.3V
- GND → GND
- SCL → A5
- SDA → A4

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;
```

```
void setup() {
  Serial.begin(9600);
  Wire.begin();

  mpu.initialize();

  if (mpu.testConnection()) {
    Serial.println("MPU6050 connection successful");
  } else {
    Serial.println("MPU6050 connection failed");
  }
}

void loop() {
  int16_t ax, ay, az;
  int16_t gx, gy, gz;

  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  Serial.println("--- Motion Data ---");
  Serial.print("Accelerometer: ");
  Serial.print("X="); Serial.print(ax);
  Serial.print(", Y="); Serial.print(ay);
  Serial.print(", Z="); Serial.println(az);

  Serial.print("Gyroscope: ");
  Serial.print("X="); Serial.print(gx);
  Serial.print(", Y="); Serial.print(gy);
  Serial.print(", Z="); Serial.println(gz);

  // Calculate tilt angles
  float accelX = ax / 16384.0;
  float accelY = ay / 16384.0;
  float accelZ = az / 16384.0;

  float pitch = atan2(accelY, sqrt(accelX * accelX + accelZ * accelZ)) * 180.0 / PI;
  float roll = atan2(-accelX, accelZ) * 180.0 / PI;

  Serial.print("Pitch: "); Serial.print(pitch);
  Serial.print("°, Roll: "); Serial.print(roll); Serial.println("°");

  Serial.println();
  delay(500);
}
```

7. Soil and pH Sensors

7.1. Soil Moisture Sensor

Soil moisture sensors measure the water content in soil, essential for automated irrigation systems.

Specifications:

- Operating voltage: 3.3V to 5V
- Output: Analog and Digital
- Probe length: typically 6cm
- Corrosion resistant

Wiring:

- VCC → 5V
- GND → GND
- A0 → A0
- D0 → Digital Pin 2

```
#define SOIL_MOISTURE_PIN A0
#define SOIL_DIGITAL_PIN 2
#define PUMP_PIN 7

int soilMoisture = 0;
int dryThreshold = 400;
int wetThreshold = 800;

void setup() {
  Serial.begin(9600);
  pinMode(SOIL_DIGITAL_PIN, INPUT);
  pinMode(PUMP_PIN, OUTPUT);
  Serial.println("Soil Moisture Sensor Ready");
}

void loop() {
  soilMoisture = analogRead(SOIL_MOISTURE_PIN);
  int digitalValue = digitalRead(SOIL_DIGITAL_PIN);

  Serial.print("Soil Moisture: ");
  Serial.print(soilMoisture);
  Serial.print(" | Digital: ");
  Serial.println(digitalValue);

  if (soilMoisture < dryThreshold) {
    Serial.println("Soil is DRY - Watering needed!");
    digitalWrite(PUMP_PIN, HIGH); // Turn on water pump
  } else if (soilMoisture > wetThreshold) {
    Serial.println("Soil is WET - No watering needed");
    digitalWrite(PUMP_PIN, LOW); // Turn off water pump
  } else {
    Serial.println("Soil moisture is OPTIMAL");
    digitalWrite(PUMP_PIN, LOW);
  }

  delay(1000);
}
```

8. Multi-Sensor Integration Project

8.1. Weather Station with Data Logging

This example combines multiple sensors to create a comprehensive weather monitoring system.

```
#include <DHT.h>
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <SoftwareSerial.h>

// Sensor pins and initialization
#define DHT_PIN 2
#define DHT_TYPE DHT22
#define LDR_PIN A0
#define RAIN_PIN A1
```

```
DHT dht(DHT_PIN, DHT_TYPE);
Adafruit_BMP280 bmp;

// Data structure
struct WeatherData {
  float temperature;
  float humidity;
  float pressure;
  int lightLevel;
  int rainLevel;
  unsigned long timestamp;
};

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

  // Initialize sensors
  dht.begin();

  if (!bmp.begin()) {
    Serial.println("BMP280 sensor not found!");
    while (1);
  }

  Serial.println("Multi-Sensor Weather Station Ready");
  Serial.println("Time,Temp(C),Humidity(%),Pressure(hPa),Light,Rain");
}

void loop() {
  WeatherData data;

  // Read all sensors
  data.temperature = dht.readTemperature();
  data.humidity = dht.readHumidity();
  data.pressure = bmp.readPressure() / 100.0F;
  data.lightLevel = analogRead(LDR_PIN);
  data.rainLevel = analogRead(RAIN_PIN);
  data.timestamp = millis();

  // Validate readings
  if (isnan(data.temperature) || isnan(data.humidity)) {
    Serial.println("Error reading DHT sensor!");
    delay(2000);
    return;
  }

  // Display data
  Serial.print(data.timestamp);
  Serial.print(",");
  Serial.print(data.temperature);
  Serial.print(",");
  Serial.print(data.humidity);
  Serial.print(",");
  Serial.print(data.pressure);
  Serial.print(",");
  Serial.print(data.lightLevel);
  Serial.print(",");
  Serial.println(data.rainLevel);
```

```
  // Weather analysis
  analyzeWeather(data);

  delay(5000); // Read every 5 seconds
}

void analyzeWeather(WeatherData data) {
  Serial.print("Weather Analysis: ");

  if (data.temperature > 30) {
    Serial.print("Hot ");
  } else if (data.temperature < 10) {
    Serial.print("Cold ");
  } else {
    Serial.print("Moderate ");
  }

  if (data.humidity > 80) {
    Serial.print("Humid ");
  } else if (data.humidity < 30) {
    Serial.print("Dry ");
  }

  if (data.pressure < 1000) {
    Serial.print("Low Pressure ");
  } else if (data.pressure > 1020) {
    Serial.print("High Pressure ");
  }

  if (data.lightLevel < 100) {
    Serial.print("Dark ");
  } else if (data.lightLevel > 800) {
    Serial.print("Bright ");
  }

  if (data.rainLevel < 500) {
    Serial.print("Wet ");
  }

  Serial.println();
}
```

9. Best Practices and Troubleshooting

9.1. Sensor Calibration

Many sensors require calibration for accurate readings. Here's a general calibration approach:

```
// Generic sensor calibration example
void calibrateSensor() {
  Serial.println("Starting sensor calibration...");

  float readings[10];
  float sum = 0;

  // Take multiple readings
  for (int i = 0; i < 10; i++) {
    readings[i] = analogRead(A0);
```

```
    sum += readings[i];
    delay(500);
  }

  float average = sum / 10.0;
  float offset = 512 - average; // Assuming 512 is the target value

  Serial.print("Calibration offset: ");
  Serial.println(offset);

  // Store offset in EEPROM for permanent storage
  EEPROM.write(0, offset);
}
```

9.2. Error Handling and Validation

Implement robust error handling for sensor readings:

```
bool validateSensorReading(float value, float minVal, float maxVal) {
  if (isnan(value) || isinf(value)) {
    Serial.println("Error: Invalid sensor reading (NaN/Inf)");
    return false;
  }

  if (value < minVal || value > maxVal) {
    Serial.println("Error: Sensor reading out of range");
    return false;
  }

  return true;
}
```

9.3. Power Management

For battery-powered projects, implement power-saving techniques:

```
#include <avr/sleep.h>
#include <avr/power.h>

void enterSleepMode() {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  power_adc_disable();
  power_spi_disable();
  power_timer0_disable();
  power_timer1_disable();
  power_timer2_disable();
  power_twi_disable();

  sleep_mode();

  // Wake up here
  sleep_disable();
  power_all_enable();
}
```

10. Conclusion

This covers the most commonly used sensors with Arduino,code examples and implementation details. Each sensor has its specific applications and considerations:

- **Temperature sensors** are essential for climate monitoring and control systems

- **Motion sensors** enable security and automation applications
- **Environmental sensors** provide data for weather stations and air quality monitoring
- **Light sensors** enable responsive lighting and photography applications
- **Gas sensors** are crucial for safety and air quality applications

When working with sensors, always consider factors such as accuracy requirements, environmental conditions, power consumption, and data logging needs. Regular calibration and proper error handling ensure reliable and accurate sensor readings.

Figure 1: Logo of Tribhuvan University.