Task 31

Deep Das

1. Implement logic to move the servo motor based on the distance measured by the ultrasonic sensor.

For example, rotate the servo to 0 degrees if the distance is greater than a threshold, and to 180 degrees if the distance is less than the threshold.

Answer :

```
#include <Servo.h>


Servo servo3;


const int sensorPin = 7;
int distance = 0;
int pos = 90;


long readUltrasonicDistance(int sensorPin) {
  pinMode(sensorPin, OUTPUT);
  digitalWrite(sensorPin, LOW);
  delayMicroseconds(2);
  digitalWrite(sensorPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sensorPin, LOW);
  pinMode(sensorPin, INPUT);
  return pulseIn(sensorPin, HIGH);
}


void setup() {
  Serial.begin(9600);
  servo3.attach(3, 500, 2500);
```

```
  servo3.write(pos);
}

void loop() {
  long duration = readUltrasonicDistance(sensorPin);
  distance = duration * 0.034 / 2;

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

  if (distance < 200) {
    if (pos < 180) {
      pos++;
      servo3.write(pos);
    }
  } else {
    if (pos > 0) {
      pos--;
      servo3.write(pos);
    }
  }

  delay(20);
```

2. Write a program that calculates a person's Body Mass Index (BMI). The program should prompt the user to enter their height in feet and inches, and their weight in pounds. It should then calculate the BMI . The program should display the calculated BMI and provide an interpretation of the result based on standard BMI ranges.

Answer :

lastcpp.cpp > main()

```cpp
int main()
{
    int feet, inches;
    float weight, heightInInches, heightInMeters, bmi;

    cout << "Enter your height:" << endl;
    cout << "Feet: ";
    cin >> feet;
    cout << "Inches: ";
    cin >> inches;

    cout << "Enter your weight in pounds: ";
    cin >> weight;

    heightInInches = (feet * 12) + inches;
    heightInMeters = heightInInches * 0.0254;
    weight = weight * 0.453592;
    bmi = weight / (heightInMeters * heightInMeters);

    cout << "Your BMI is: " << bmi << endl;

    if (bmi < 18.5)
    {
        cout << "Interpretation: Underweight" << endl;
    }
    else if (bmi >= 18.5 && bmi < 24.9)
    {
        cout << "Interpretation: Normal weight" << endl;
    }
    else if (bmi >= 25 && bmi < 29.9)
    {
        cout << "Interpretation: Overweight" << endl;
    }
    else
    {
        cout << "Interpretation: Obesity" << endl;
    }
```

3. Write embedded C program that blinks the light blue when the source of sound is within the range of 110cm ,blinks green within the range of 220cm and blinks red within the range of 330 cm.

Answer : #define SENSOR_PIN 7

#define BLUE_LED 3

#define GREEN_LED 4

#define RED_LED 5

void setup() {

pinMode(SENSOR_PIN, OUTPUT);

pinMode(BLUE_LED, OUTPUT);

```
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  Serial.begin(9600);
}

long readUltrasonicDistance() {
  pinMode(SENSOR_PIN, OUTPUT);
  digitalWrite(SENSOR_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(SENSOR_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(SENSOR_PIN, LOW);
  pinMode(SENSOR_PIN, INPUT);
  return pulseIn(SENSOR_PIN, HIGH);
}

void loop() {
  long duration, distance;
  duration = readUltrasonicDistance();
  distance = (duration / 2) * 0.0343;

  if (distance <= 110) {
    blinkLED(BLUE_LED);
  }
  else if (distance <= 220) {
    blinkLED(GREEN_LED);
  }
  else if (distance <= 330) {
    blinkLED(RED_LED);
  }
  else {
```

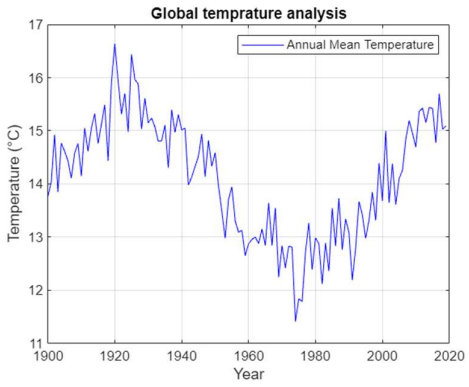
```
    turnOffLEDs();
  }
  delay(1000);
}

void blinkLED(int pin) {
  digitalWrite(pin, HIGH);
  delay(500);
  digitalWrite(pin, LOW);
  delay(500);
}

void turnOffLEDs() {
  digitalWrite(BLUE_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
  digitalWrite(RED_LED, LOW);
}
```

Python

1. Examine the temperature changes over the last century and plot the annual average temperatures.

Answer :

```
/MATLAB Drive/last.m
1  years = 1900:2019;
2  mean_temp = 14 + sin((years - 1900) * 2 * pi / 100) * 1.5 + ra
3
4  figure;
5  plot(years, mean_temp, 'b-');
6  title('Global temprature analysis');
7  xlabel('Year');
8  ylabel('Temperature (°C)');
9  grid on;
10 legend('Annual Mean Temperature');
11
```



2. Track a moving object across a video sequence. Draw the tracked object's bounding box on each frame and display the results. You may use the CamShift algorithm for object tracking.

Answer :

```
import cv2
import numpy as np

video_file = r"C:\Users\lenovo\Videos\Captures\screen recorder for windows
inbuilt - Google Search - Google Chrome 2024-05-26 13-00-41.mp4"
cap = cv2.VideoCapture(video_file)

ret, frame = cap.read()
if not ret:
    print("Error: Failed to read video file")
    exit()

# Select ROI for tracking
bbox = cv2.selectROI("Select Object to Track", frame, fromCenter=False,
showCrosshair=True)
track_window = (bbox[0], bbox[1], bbox[2], bbox[3])

# Initialize CamShift tracker
roi = frame[bbox[1]:bbox[1]+bbox[3], bbox[0]:bbox[0]+bbox[2]]
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255.,
255.)))
roi_hist = cv2.calcHist([hsv_roi], [0], mask, [180], [0, 180])
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iterations or move by at least 1
pixel
term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)

    # Apply CamShift to get the new location
    ret, track_window = cv2.CamShift(dst, track_window, term_crit)

    # Draw the tracked object as a bounding box
    pts = cv2.boxPoints(ret)
    pts = np.int0(pts)
    img = cv2.polylines(frame, [pts], True, (0, 255, 0), 2)
```

```
    # Display the frame with the tracked object
    cv2.imshow('Tracked Object', img)

    # Exit if ESC pressed
    if cv2.waitKey(30) & 0xFF == 27:
        break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

3. WAP to print the following patterns:

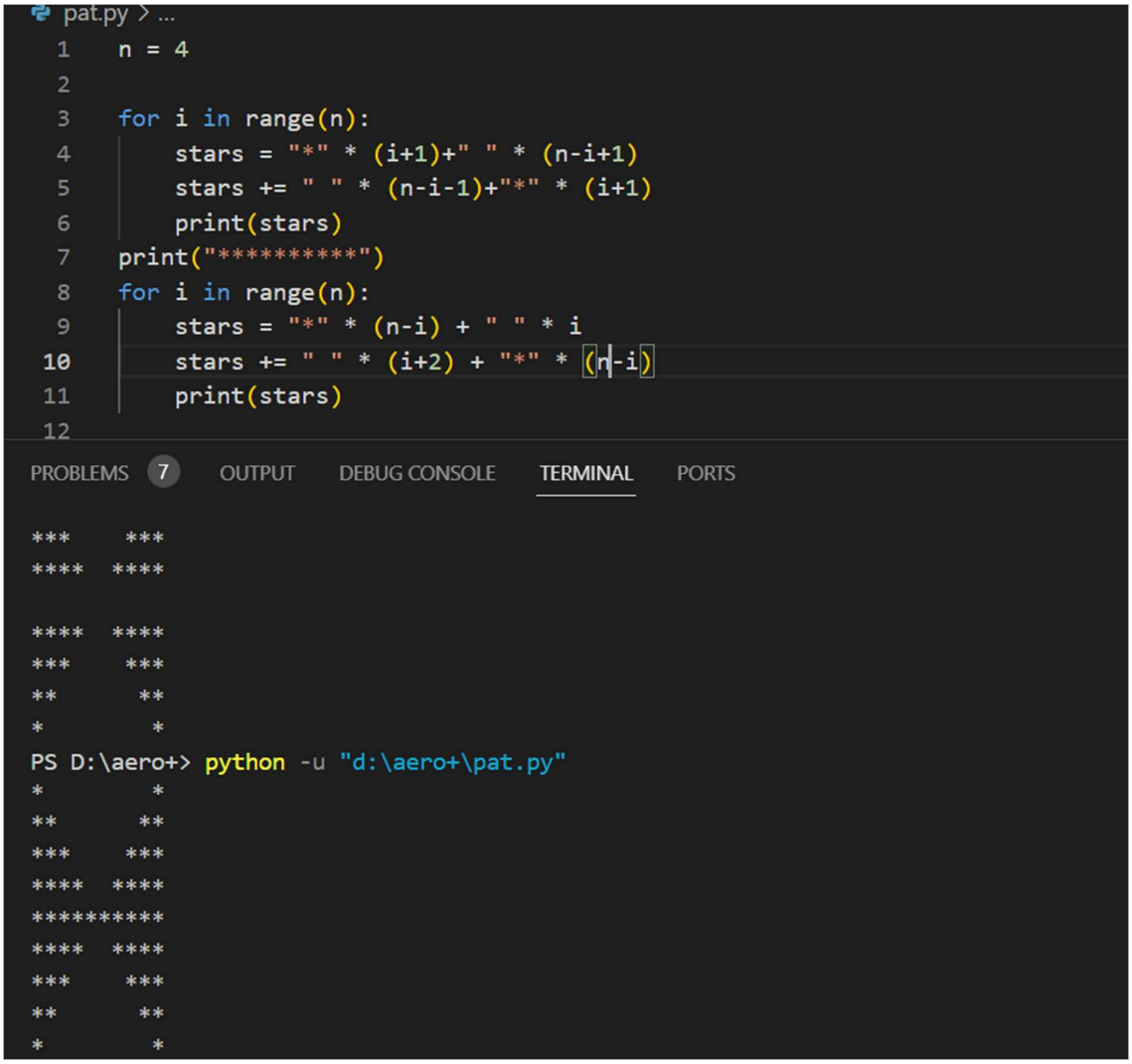
   Answer :

pat.py > ...

```python
n = 5
for i in range(n):
    spaces = " " * (i+1)
    stars = "*" * (2*(n-i-1)+1)
    print(spaces + stars)
for i in range(n):
    spaces = " " * ((n-i-1)+1)
    stars = "*" * (2*i+1)
    print(spaces + stars)
```

PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
> python -u "d:\aero+\pat.py"
 *********
  *******
   *****
    ***
     *
     *
    ***
   *****
  *******
 *********
```



Drone

1. Explain communication protocols and its types.

Answer : SPI (Serial Peripheral Interface)

Description: SPI (Serial Peripheral Interface) is a synchronous serial communication protocol developed by Motorola. It is widely used in embedded systems to transfer data between a microcontroller (master) and peripheral devices (slaves) such as sensors, SD cards, and displays.

Key Features:

- **Four Signal Lines:**
  - **MOSI (Master Out Slave In):** Sends data from master to slave.
  - **MISO (Master In Slave Out):** Sends data from slave to master.
  - **SCK (Serial Clock):** Clock signal generated by master to synchronize data transfer.
  - **SS (Slave Select):** Line to select a specific slave device.

- **Full-Duplex Communication:** Enables simultaneous transmission and reception of data.
- **Master-Slave Architecture:** One master device controls communication with one or more slave devices, selected using the SS line.
- **High Speed:** Capable of very high-speed data transfer suitable for applications requiring rapid data exchange.
- **Simple Hardware Interface:** Minimal pin connections and straightforward signal wiring simplify hardware design and implementation.

I2C (Inter-Integrated Circuit)

Description: I2C (Inter-Integrated Circuit) is a multi-master, multi-slave, packet-switched, serial communication bus developed by Philips (now NXP). It is commonly used for connecting low-speed peripherals to microcontrollers in short-distance, intra-board communication.

Key Features:

- **Two Signal Lines:**
  - **SDA (Serial Data Line):** Transmits data between master and slave devices.
  - **SCL (Serial Clock Line):** Clock signal generated by master to synchronize data transfer.
- **Addressing:** Each device on the bus has a unique address, allowing multiple devices to share communication lines.
- **Multi-Master Capability:** Multiple master devices can be connected to the same bus, with one active at a time.
- **Simple Protocol:** Uses start/stop and acknowledgment for communication, making it easy to implement in hardware and software.
- **Clock Stretching:** Allows slower devices to hold the clock line low until ready for reliable data transfer.

UART (Universal Asynchronous Receiver/Transmitter)

Description: UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol used for full-duplex serial communication. It operates in both asynchronous mode, making it versatile for various applications.

Key Features:

- **Full-Duplex Communication:** Allows simultaneous bidirectional data transfer between transmitter and receiver.
- **Asynchronous Mode:** Data is transferred without a clock signal, using start/stop bits for synchronization.
- **Configurable Baud Rate:** Transfer speed configurable to match application requirements.
- **Parity, Stop, and Data Bits:** Configurable settings for data integrity and compatibility.

CAN (Controller Area Network)

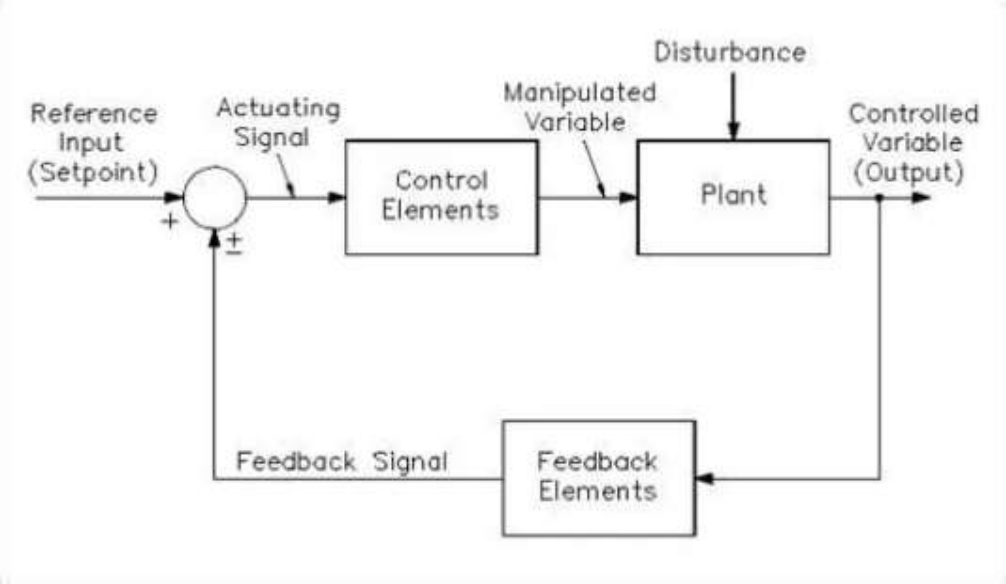
Description: CAN (Controller Area Network) is a robust vehicle bus standard designed for reliable communication between microcontrollers and devices without a host computer. Widely used in automotive and industrial applications.

Key Features:

- **Multi-Master Configuration:** Multiple devices can communicate on the same bus, allowing decentralized control.
- **Error Detection and Handling:** Includes mechanisms for detecting and handling errors to ensure reliable communication.
- **Priority-Based Bus Access:** Messages sent based on priority, ensuring critical data transmission first.
- **High Reliability:** Designed for harsh environments with differential signaling to improve noise immunity.

2. Explain flight controller with control system diagram

Answer :



1. **Reference Input (Setpoint)**:
    - **Definition**: The desired value or target for the system.
    - **Flight Controller Context**: This could be the desired altitude, heading, speed, or any other flight parameter set by the pilot or an autopilot system.
2. **Actuating Signal**:
    - **Definition**: The difference between the setpoint and the feedback signal, which drives the control elements.
    - **Flight Controller Context**: The difference between the desired flight path and the current flight path. This error signal is used to determine the necessary adjustments.
3. **Control Elements**:

    - **Definition**: Components that process the actuating signal to generate the manipulated variable.
    - **Flight Controller Context**: This includes the flight control computer or autopilot system that processes the error signal and computes the necessary control commands.
4. **Manipulated Variable**:
    - **Definition**: The output from the control elements that acts on the plant.
    - **Flight Controller Context**: These are the commands sent to the aircraft's actuators, such as servos controlling the ailerons, elevators, rudders, and throttles.
5. **Plant**:
    - **Definition**: The system to be controlled.
    - **Flight Controller Context**: The aircraft itself, including its dynamics and response to control inputs.
6. **Disturbance**:
    - **Definition**: External factors that affect the plant but are not controlled by the system.
    - **Flight Controller Context**: This could include wind gusts, turbulence, or any other environmental factors that affect the aircraft's flight.
7. **Controlled Variable (Output)**:
    - **Definition**: The actual value of the parameter being controlled.
    - **Flight Controller Context**: The actual altitude, heading, speed, etc., of the aircraft.
8. **Feedback Signal**:
    - **Definition**: The actual measured value of the controlled variable.
    - **Flight Controller Context**: The sensor data from instruments such as altimeters, gyroscopes, GPS, and airspeed indicators.
9. **Feedback Elements**:
    - **Definition**: Components that measure the controlled variable and send the feedback signal.
    - **Flight Controller Context**: The sensors and instruments that measure the aircraft's current state and send this information back to the control elements.

Explanation of the Feedback Control Loop

- The **Reference Input** sets the desired target for the aircraft, such as a specific altitude or heading.
- The **Actuating Signal** is generated by comparing the reference input with the feedback signal (the current state of the aircraft). This signal represents the error or deviation from the desired state.
- The **Control Elements** (flight control computer or autopilot) process this error signal to determine the necessary adjustments. These adjustments are sent as the **Manipulated Variable** to the aircraft's control surfaces.
- The **Plant** (aircraft) responds to these control inputs by changing its flight path.
- External **Disturbances** like wind or turbulence can affect the aircraft's flight path, causing deviations from the desired state.
- The **Controlled Variable** is the aircraft's actual flight path, which is continuously monitored.
- The **Feedback Elements** (sensors and instruments) measure the current flight path and generate the **Feedback Signal**.
- This feedback signal is sent back to the control elements to update the error signal, forming a closed-loop system that continuously adjusts the aircraft's control surfaces to maintain the desired flight path.

Flight Controller Implementation

In a real-world flight controller system:

- **Reference Input** could be set manually by the pilot or automatically by the autopilot.
- **Control Elements** include sophisticated algorithms implemented in the flight control computer.
- **Sensors and Instruments** provide real-time data for the feedback signal.
- The system continuously adjusts the aircraft's control surfaces (ailerons, elevators, rudder, and throttle) to maintain stable and accurate flight.

3. Differentiate between Kalman filter and Complimentary filter

Answer :

Feature	Kalman Filter	Complementary Filter
Algorithm Type	Optimal recursive estimator	Simple fusion filter
Application	Wide range of fields (aerospace, robotics)	Orientation estimation (IMU sensor fusion)
Mathematical Basis	Probabilistic models, Gaussian distributions	Weighted average of filtered sensor data
Advantages	Optimal performance, handles non-linearities	Simplicity, real-time performance
Disadvantages	Computational complexity	Limited accuracy in complex environments
Complexity	More complex	Simpler
Accuracy	Generally higher	Lower, especially in complex scenarios
Implementation	More resource-intensive	Easier to implement, less computational
Suitability	Versatile, precise state estimation	Orientation estimation, simpler tasks

4. Briefly explain RTK GPS.

Answer : RTK (Real-Time Kinematic) GPS is an advanced satellite navigation technique used to enhance the accuracy of standard GPS systems, particularly in applications requiring centimeter-level positioning precision. Here's a brief explanation of RTK GPS:

Overview:

1. **Principle:**
   - RTK GPS works by using a fixed ground station (base station) with a known location to send correction data to a mobile receiver (rover). The rover receiver compares its own GPS measurements with the corrections from the base station to calculate highly accurate position fixes.
2. **Operation:**

    - **Base Station:** The base station precisely measures its location using GPS and transmits correction data to nearby rovers in real-time via radio signals or cellular networks.
    - **Rover:** The rover receiver uses the correction data received from the base station to refine its own GPS measurements, compensating for errors such as atmospheric disturbances and satellite clock drift.
3. **Accuracy:**
    - RTK GPS can achieve centimeter-level accuracy in positioning, significantly better than standard GPS, which typically provides accuracy within several meters.
    - Accuracy depends on factors like the distance between the base station and rover, quality of satellite signals, and environmental conditions.
4. **Applications:**
    - **Surveying and Mapping:** RTK GPS is widely used in land surveying, construction site mapping, and precision agriculture for precise location and mapping of features.
    - **Navigation:** Used in autonomous vehicles, drones, and marine vessels where accurate positioning is critical for navigation and control.
    - **Infrastructure Monitoring:** Monitoring and managing infrastructure such as bridges, dams, and pipelines that require precise geolocation data.
5. **Challenges:**
    - **Cost and Infrastructure:** Setting up and maintaining RTK systems can be costly due to the need for base stations and continuous correction data transmission.
    - **Environmental Factors:** Accuracy may be affected by signal obstructions, multipath interference, and atmospheric conditions.

Benefits:

- **High Precision:** Provides centimeter-level accuracy, making it ideal for applications demanding precise positioning.
- **Real-Time Capability:** Corrections are applied immediately, enabling real-time monitoring and control.
- **Versatility:** Used across various industries for tasks requiring accurate spatial data acquisition and navigation.

5. Briefly explain about PWM servos and PWM signals.

Answer : **PWM Servos:**

PWM (Pulse Width Modulation) servos are actuators commonly used in robotics, RC (remote control) vehicles, and various automation applications. They are controlled using PWM signals to determine their position or speed. Here's a brief overview:

1. **Functionality:**
    - PWM servos operate based on the duration of PWM pulses sent to them. The width of these pulses determines the position or speed of the servo's output shaft.
    - Typically, PWM servos are used for angular motion, where the shaft rotates to a specific angle corresponding to the PWM signal received.
2. **Control Signal:**
    - The control signal for PWM servos is a periodic square wave signal, where the width of the high (on) pulse varies.

    - The position or speed of the servo is determined by the duty cycle of the PWM signal—the ratio of the pulse width (on time) to the total period (on + off time).
3. **Operation:**
    - When a PWM signal is applied, the servo motor moves to the position corresponding to the pulse width. Commonly, a 1.0 to 2.0 ms pulse width corresponds to a range of 0 to 180 degrees for standard servos.
    - Continuous rotation servos use PWM signals to control speed and direction, with a specific pulse width corresponding to stopped, forward, and reverse motion.
4. **Applications:**
    - Used in robotics for precise angular positioning of joints.
    - In RC vehicles for controlling steering and throttle.
    - Automation applications for actuating valves, switches, and other mechanisms.

PWM Signals:

PWM signals are a method of encoding information in the form of a varying duty cycle square wave. They are widely used for controlling servos and other devices requiring precise timing control. Key points about PWM signals include:

1. **Definition:**
    - A PWM signal consists of a periodic waveform where the duration of the high (on) state (pulse width) varies while the period (total cycle time) remains constant.
2. **Duty Cycle:**
    - The duty cycle of a PWM signal is the ratio of the pulse width to the total period expressed as a percentage.
    - It determines the average power delivered to the load or the position/speed of the servo motor.
3. **Frequency:**
    - The frequency of a PWM signal is the rate at which the waveform repeats (measured in Hertz, Hz).
    - Higher frequencies provide smoother control and reduce audible noise in motors and actuators.
4. **Applications:**
    - Apart from controlling servos, PWM signals are used in motor speed control, LED dimming, switching power supplies, and audio amplification.
    - They are versatile due to their ability to efficiently control analog devices using digital signals.