Final project in computer embedded systems

Object Tracker with Ultrasonic Sensor

JERUSALEM COLLEGE OF TECHNOLOGY
LEV ACADEMIC CENTER

Madar Eliaou 1954753

Baryohay Uzan 916781

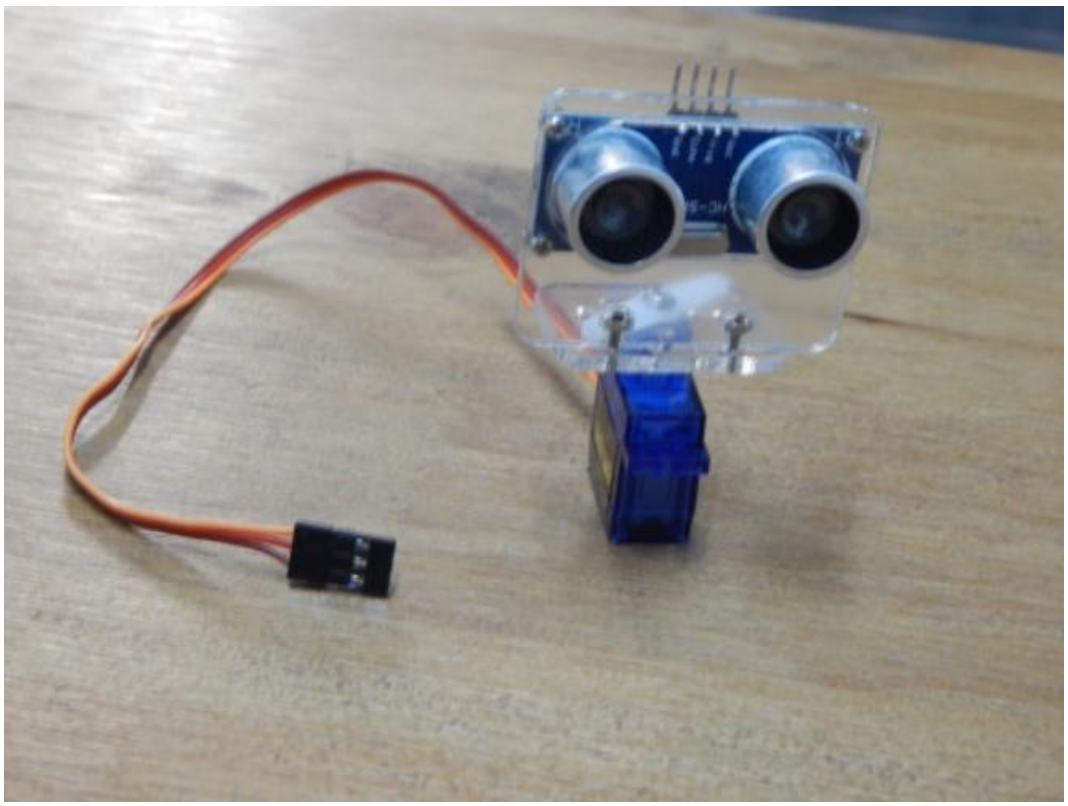


Figure 1: ROBOTIC ULTRASONIC SENSOR

Explanation of the System and Its Objectives

The presented system is a microcontroller-based device that uses an ultrasonic sensor and a servo motor to detect and track a moving object. It is designed for applications requiring precise tracking or detection in environments where the distance to an object is a key factor.

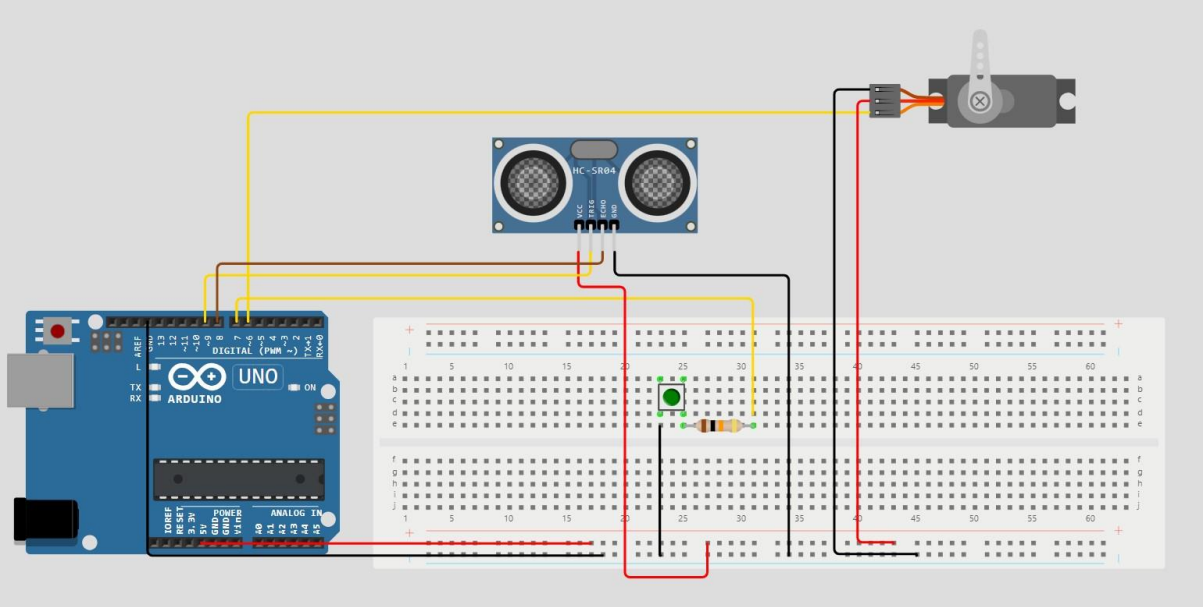
Main Objectives:

- **Accurate Distance Measurement**: Uses an ultrasonic sensor to measure the distance to an object.
- **Object Tracking**: Adjusts the position of a servo motor to center the detected object.
- **Interactivity**: Allows the user to manually set a target distance using a push button.
- **Adaptability**: Handles lateral movements of the object using a scanning system.

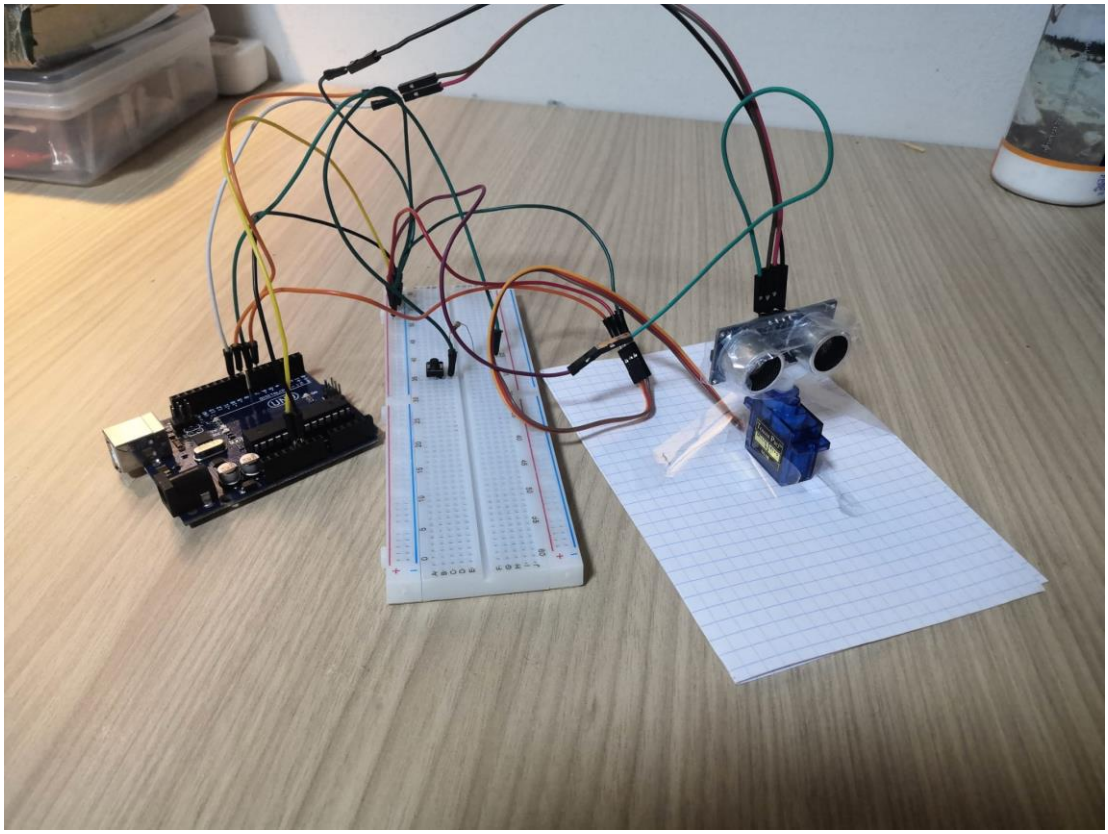
Potential Applications:

- **Zoom Conference Speaker Tracking**: The system can automatically track and center a professor or presenter during virtual conferences, ensuring they stay in focus for the audience.(and place a camera on the tracker)
- **Surveillance Systems**: Detect and follow moving objects for security purposes.
- **Industrial Automation**: Precise object tracking in automated production lines.

The shema:



Link to the video: https://drive.google.com/file/d/13ss_CGxRmFAJLUoowa4gB-aYodaUp7zX/view?usp=drivesdk



Explanation of Technical Challenges and Their Solutions

Technical Challenges:

1. **Stability of Distance Measurements**:

    a. Ultrasonic sensors may return incorrect measurements due to interferences or reflections.

    b. **Solution**: Add a delay between measurements to reduce noise and use a tolerance range to ignore minor variations.

2. **Managing Lateral Movement**:

    a. Detecting if the object moves out of the sensor's field of view without causing erratic servo movements.
    b. **Solution**: Introduce a threshold (`maxDistanceChange`) to detect significant lateral movement and initiate a controlled scanning process with the servo motor.

3. **Synchronizing Components (Servo and Sensor)**:

    a. The servo motor and ultrasonic sensor operate at different speeds, which may cause response delays.

    b. **Solution**: Add a short delay after each servo movement to ensure the sensor accurately measures the distance.

4. **Handling the Push Button**:

    a. Avoiding debounce issues when pressing the button, which may lead to incorrect distance readings.

    b. **Solution**: Implement a debounce delay (`delay(500)`) after detecting a button press.

Brief Explanation of Each Function

1. `setup()`

- **Purpose**: Configures the hardware components at system startup.
- **Importance**:
  - Initializes pins for the sensor, button, and servo motor.
  - Centers the servo motor to start with a reference position.
  - Begins serial communication to display data.

2. `loop()`

- **Purpose**: The main loop that handles the system's real-time behavior.
- **Importance**:
  - Reads the button state to set the target distance.
  - Continuously measures the current distance to the object.
  - Adjusts the servo motor to track the object or initiates a search when necessary.

3. `getDistance()`

- **Purpose**: Measures the distance to the object using the ultrasonic sensor.
- **Importance**:
  - Provides the core data required to detect and track the object.
  - Converts signal durations into a precise distance (in cm).

4. `scanForObject()`

- **Purpose**: Searches for the object if it moves out of the sensor's field of view.
- **Importance**:
  - Performs a controlled scan using the servo motor to locate the target.
  - Optimizes the scanning process by prioritizing the last known direction (left or right).

5. `scanToDirection(int startPos, int direction)`

- **Purpose**: Scans in a specific direction (left or right) to locate the object.
- **Importance**:
  - Executes a progressive scan using small increments (`scanStep`).
  - Quickly identifies if the object is within a defined range and adjusts the servo motor's position accordingly.

Here’s the detailed explanation of the most important variable :

float targetDistance = 8.0;

- This variable defines the target distance in centimeters at which the object should be positioned relative to the sensor.
- Initially set to 8 cm, it can be updated by the user when the button is pressed.
- Usage:

It serves as the reference distance. The system attempts to adjust the servo motor to maintain the object at this distance.

bool isTargetSet = false;

This variable acts as a flag to indicate whether the target distance has been set by the user (via the button).

- Before the button is pressed, the program prevents the servo motor from trying to adjust its position.
- Once the target distance is recorded, the program compares the measured distance to this target distance and adjusts the servo motor accordingly.

int servoPosition = 90;

Represents the current position of the servo motor in degrees.

- The initial value of 90 corresponds to the servo motor's center position (neutral orientation).
- The servo can move from 0° to 180°, representing its movement limits.

- This value is updated based on the detected object.
- If the object is too far or too close, the servo position is adjusted left or right to realign the sensor.

const float tolerance = 15.0;

Defines a tolerance range around the target distance (in cm).

- If the detected object is within this range, no adjustment is necessary.

- Ensures that the system doesn’t react to small variations in the measured distance, avoiding unnecessary servo motor movements.

const float maxDistanceChange = 15.0;

Defines the maximum threshold to detect lateral movement of the object.

- If the measured distance changes abruptly by more than 15 cm, it is interpreted as lateral movement of the object (rather than a simple forward/backward shift).
- Triggers a search mode where the servo motor scans the area to relocate the object.

const int scanStep = 5;

Specifies the degree increment by which the servo motor moves during a scan to relocate a lost object.

- Smaller values result in more precise scans but take longer.
- A value of 5° provides a good balance between precision and speed.

String lastDirection = "";

Stores the last direction in which the servo motor moved to adjust its position or search for the object.

- Possible values are "left" or "right", or an empty string if no direction has been recorded yet.
- Provides priority to the last known direction during an object search. If the object moves laterally, the program will begin scanning in the direction where it was last detected.

The Code:

```
#include <Servo.h>

// Define pin connections
#define TRIG_PIN 9      // Pin for ultrasonic sensor trigger
#define ECHO_PIN 8      // Pin for ultrasonic sensor echo
#define BUTTON_PIN 7    // Pin for button input
```

```
#define SERVO_PIN 6     // Pin for servo motor

Servo servo;  // Servo object to control the motor
float targetDistance = 8.0; // Target distance in cm
bool isTargetSet = false;   // Flag to check if the target distance is set
int servoPosition = 90;     // Initial position of the servo (center)
const float tolerance = 15.0; // Tolerance range for distance adjustment (in
cm)
const float maxDistanceChange = 15.0; // Threshold to detect lateral movement
const int scanStep = 5; // Degree increment for scanning
String lastDirection = ""; // Last detected direction ("left" or "right")

void setup() {
  pinMode(TRIG_PIN, OUTPUT);   // Set TRIG_PIN as output
  pinMode(ECHO_PIN, INPUT);    // Set ECHO_PIN as input
  pinMode(BUTTON_PIN, INPUT_PULLUP); // Set BUTTON_PIN as input with pull-up
resistor

  servo.attach(SERVO_PIN);    // Attach servo motor to SERVO_PIN
  servo.write(servoPosition); // Set servo to initial center position

  Serial.begin(9600); // Start serial communication
}

void loop() {
  // Check if the button is pressed
  if (digitalRead(BUTTON_PIN) == LOW) {
    targetDistance = getDistance(); // Measure and set target distance
    isTargetSet = true;             // Mark target as set
    lastDirection = "";             // Reset the last direction
    Serial.print("Target distance set: ");
    Serial.print(targetDistance);
    Serial.println(" cm");
    delay(500); // Debounce delay
  }

  if (isTargetSet) {
    float currentDistance = getDistance(); // Measure the current distance

    // Check if the object moved laterally (sudden change in distance)
    if (abs(currentDistance - targetDistance) > maxDistanceChange) {
      Serial.println("Object moved laterally, scanning...");
      scanForObject(); // Initiate scanning to locate the object
      return;
    }
```

```
    // Adjust the servo motor if the distance is slightly different
    if (currentDistance > targetDistance + tolerance) {
      // Object is farther away, move servo to the left
      servoPosition = min(servoPosition + 1, 180);
      servo.write(servoPosition);
      lastDirection = "left"; // Remember the direction
    } else if (currentDistance < targetDistance - tolerance) {
      // Object is closer, move servo to the right
      servoPosition = max(servoPosition - 1, 0);
      servo.write(servoPosition);
      lastDirection = "right"; // Remember the direction
    }

    // Print current distance to the serial monitor
    Serial.print("Current distance: ");
    Serial.print(currentDistance);
    Serial.println(" cm");
    delay(20); // Delay to smooth servo movements
  }
}

// Function to measure distance using the ultrasonic sensor
float getDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance = duration * 0.034 / 2; // Convert duration to distance in cm
  return distance;
}

// Function to scan and relocate the object
void scanForObject() {
  int scanRange = 45; // Range to scan in degrees
  int originalPosition = servoPosition;

  // Priority to last detected direction
  if (lastDirection == "right") {
    if (scanToDirection(originalPosition, 1)) return;
    if (scanToDirection(originalPosition, -1)) return;
  } else if (lastDirection == "left") {
    if (scanToDirection(originalPosition, -1)) return;
    if (scanToDirection(originalPosition, 1)) return;
  } else {
```

```
    // Default scanning in both directions
    if (scanToDirection(originalPosition, -1)) return;
    if (scanToDirection(originalPosition, 1)) return;
  }

  // Return to the original position if the object is not found
  servo.write(originalPosition);
  Serial.println("Object not found after scanning.");
}

// Function to scan in a specific direction
bool scanToDirection(int startPos, int direction) {
  for (int pos = startPos;
       (direction > 0 ? pos <= startPos + 50 : pos >= startPos - 50);
       pos += direction * scanStep) {
    if (pos < 0 || pos > 180) break; // Ensure servo position is within bounds
    servo.write(pos);
    delay(80);
    float distance = getDistance();
    if (abs(distance - targetDistance) <= tolerance) {
      servoPosition = pos; // Update the current position
      lastDirection = (direction > 0) ? "right" : "left"; // Record the
direction
      Serial.print("Object found in ");
      Serial.println(lastDirection);
      return true;
    }
  }
  return false;
}
```