Autonomous Robot with Obstacle Avoidance and Auto-Parking

Embedded Systems

SUPERVISED BY: DR. MOHAMED ELHABROUK

FACULTY OF COMPUTERS AND DATA SCIENCES,
ALEXANDRIA UNIVERSITY.
MAY 2024

Autonomous Robot with Obstacle Avoidance and Auto-Parking

Team Members

Member Name	***ID***	***Department***
Alia Medhat	20221440735	AI
Aya Abdelmoniem	20221462478	AI
Doaa		AI
Malak Mahmoud Aref	20221445867	AI
Nouran Mohamed	20221321932	AI

Table of Contents

1 Introduction

1.1 Overview of the project

This project involves designing and implementing an autonomous robot with two primary modes: obstacle avoidance and auto-parking. The robot uses an ultrasonic sensor to detect obstacles and a servo motor to maneuver around them. In the auto-parking mode, the robot will autonomously find and park in a designated area. The robot can switch between two modes: obstacle avoidance and auto-parking, controlled via Bluetooth.

1.2 Motivation

The motivation behind this project stems from the growing interest in autonomous vehicles and the potential benefits they offer in terms of safety, efficiency, and convenience. By exploring obstacle avoidance and auto-parking, this project aims to contribute to the development of intelligent systems that can operate with minimal human intervention.

1.3 Goals and Objectives

- Goal 1: Develop an obstacle avoidance system that allows the robot to navigate around obstacles autonomously.
- Goal 2: Implement an auto-parking system that enables the robot to park itself in a designated area.
- Goal 3: Ensure reliable performance in varied environments.
- Goal 4: Implement Bluetooth communication for mode switching.

1.4 Short description of the project

The project involves an Arduino-based robot that uses ultrasonic sensors to detect obstacles and navigate autonomously. The robot also features an autoparking system, allowing it to park itself when a suitable space is detected. The robot can be controlled via Bluetooth to switch between obstacle avoidance and autoparking modes.

1.5 Problem Statement

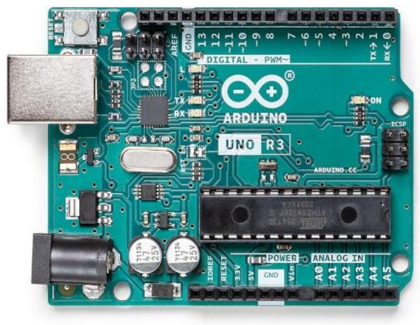
Developing a reliable and efficient system for obstacle avoidance and auto-parking poses several challenges. The main problem is to create an autonomous robot that can accurately detect and avoid obstacles while also being able to identify and park in a designated area. Ensuring the robot's decisions are both timely and accurate is critical for effective operation.

2 System Requirements

2.1Hardware Requirements:

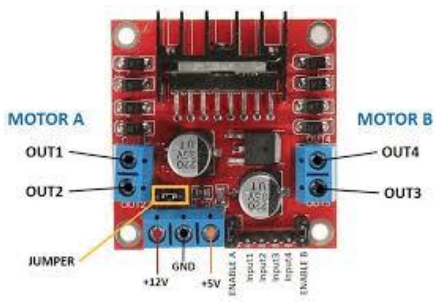
2.1.1 Arduino Uno microcontroller

The Arduino Uno is a popular open-source microcontroller board



2.1.2 H-bridge

A dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time.



2.1.3 DC gear motors

A DC gear motor is a type of electric motor that is designed to provide movement for the robot.

2.1.4 Servo motor

Servo motors are great devices that can turn to a specified position. Usually, they have a servo arm that can turn 180 degrees.

2.1.5 HC-SR04 ultrasonic sensor

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves.

2.1.6 Power supply (battery)

2.1.7 Jumpers

2.1.8 Breadboard

2.1.9 Bluetooth Module

2.2 Software Requirements:

2.2.1 Arduino IDE, The Arduino Integrated Development Environment - or Arduino Software (IDE) - connects to the Arduino boards to upload programs and communicate with them.

2.2.2 Libraries: Servo, The Servo Library is a great library for controlling servo motors.

3 Building The Architecture

To construct the architecture of the autonomous robot, various hardware components need to be connected correctly to ensure proper functionality. Below are the detailed connections for the motors, ultrasonic sensors, and servo motors:

3.1 Motor Connections:

3.1.1 Gear Motors to L298N Motor Driver:

- Connect one wire from each gear motor to the "OUT1" and "OUT2" terminals on one side of the L298N Motor Driver.
- Connect one wire from each gear motor to the "OUT3" and "OUT4" terminals on the other side of the L298N Motor Driver.

3.1.2 Power Connections:

- Connect the positive (+) terminal of the power source (battery) to the terminal labeled "12V" or "VCC" on the L298N board.
- Connect the negative (-) terminal of the power source (battery) to the terminal labeled "GND" or "-" on the L298N board.

3.1.3 Arduino Connections:

- Connect the control pins of the L298N Motor Driver to digital pins on the Arduino:
  - "IN1" to Arduino digital pin 2
  - "IN2" to Arduino digital pin 3
  - "IN3" to Arduino digital pin 4
  - "IN4" to Arduino digital pin 5
- Connect GND to Arduino GND.
- Connect 12V to Arduino Vin.

3.2 Ultrasonic Sensor Connections:

3.2.1 Power Connections:

- Connect the VCC pin of the HC-SR04 Ultrasonic sensor to the 5V pin on the Arduino.
- Connect the GND pin of the HC-SR04 Ultrasonic sensor to the GND pin on the Arduino.

3.2.2 Signal Connections:

- Connect the TRIG pin of the HC-SR04 Ultrasonic sensor to digital pin 7 on the Arduino.
- Connect the ECHO pin of the HC-SR04 Ultrasonic sensor to digital pin 6 on the Arduino.

3.3 Servo Motor Connections:

3.3.1 Power Connections:

- Connect the VCC pin of the servo motor to the 5V pin on the Arduino.
- Connect the GND pin of the servo motor to the GND pin on the Arduino.

3.3.2 Signal Connection:

- Connect the signal pin of the servo motor to digital pin 9 on the Arduino.

3.4 Auto-parking Enhancements:

For auto-parking functionality, additional ultrasonic sensors and servo motor connections are required:

3.4.1 Ultrasonic Sensor Connections:

3.4.1.1 Back Sensor:

- Connect VCC to 5V on the Arduino.
- Connect GND to GND on the Arduino.
- Connect TRIG to digital pin 13 on the Arduino.
- Connect ECHO to digital pin 8 on the Arduino.

3.4.1.2 Left Sensor:

- Connect VCC to 5V on the Arduino.
- Connect GND to GND on the Arduino.
- Connect TRIG to digital pin 12 on the Arduino.
- Connect ECHO to digital pin 11 on the Arduino.

These connections ensure proper communication and control between the Arduino microcontroller, sensors, and actuators, enabling the autonomous robot to navigate its environment effectively and perform auto-parking maneuvers with precision.

4 Experimental Setup

To conduct experiments and validate the functionality of the autonomous robot, the following setup is recommended:

4.1 Hardware Setup:

4.1.1 **Autonomous Robot:** Construct the robot according to the provided architecture, ensuring all connections are secure and components are properly attached.

4.1.2 **Power Source:** Connect a suitable power source, such as a battery pack, to power the robot's motors, Arduino microcontroller, and other electronic components.

4.1.3 **Obstacle Course:** Set up an obstacle course with various obstacles such as walls, obstacles of different heights, and objects with irregular shapes. Ensure the course provides sufficient space for the robot to maneuver.

4.1.4 **Parking Area:** Designate a parking area within the testing environment where the robot can autonomously park itself.

4.1.5 **Measurement Tools:** Prepare measurement tools such as a measuring tape or ruler to accurately measure distances and assess the robot's performance.

4.2 Software Setup:

4.2.1 **Arduino IDE:** Install the Arduino Integrated Development Environment (IDE) on your computer. Ensure the necessary libraries for servo motor control and ultrasonic sensor interfacing are installed.

4.2.2 **Upload Code:** Upload the provided Arduino code for obstacle avoidance and auto-parking functionalities to the Arduino microcontroller.

4.3 Experiment Procedure:

4.3.1 **Power On:** Turn on the power source to activate the robot. Ensure that all connections are secure and there are no loose wires or components.

4.3.2 **Obstacle Avoidance Testing:**

4.3.2.1 Place the robot at the starting point of the obstacle course.

4.3.2.2 Activate the obstacle avoidance mode by toggling the on/off switch.

4.3.2.3 Observe the robot's behavior as it navigates through the obstacle course. Note any instances of successful obstacle avoidance or collisions.

4.3.3 **Auto-parking Testing:**

4.3.3.1 Reset the robot to the starting position.

4.3.3.2 Activate the auto-parking mode by toggling the on/off switch.

4.3.3.3 Monitor the robot as it searches for and parks in the designated parking area. Evaluate the accuracy and efficiency of the parking maneuver.

4.3.4 **Performance Evaluation:**

4.3.4.1 Measure the time taken for the robot to complete the obstacle course and successfully park.

4.3.4.2 Record any instances of errors or malfunctions during the testing process.

4.3.4.3 Assess the overall performance of the robot in terms of speed, accuracy, and reliability.

4.3.5 **Data Analysis:**

4.3.5.1 Analyze the collected data to identify areas for improvement and optimization.

4.3.5.2 Compare the experimental results with the expected outcomes outlined in the project objectives.

4.3.6 **Iterative Refinement:**

4.3.6.1 Based on the analysis, make necessary adjustments to the hardware or software components to enhance the robot's performance.

4.3.6.2 Repeat the testing process to validate the effectiveness of the modifications.

By following this experimental setup, researchers and enthusiasts can evaluate the performance of the autonomous robot in various scenarios and gather valuable insights for further development and refinement of the system.

5 Appendix

5.1 Obstacle Avoidance and Auto-parking Code and Explanation

```
#include <Servo.h>

#include <SoftwareSerial.h>

// SoftwareSerial BT(1,0);

// Motor Pins

const int in1 = 2;

const int in2 = 3;

const int in3 = 4;

const int in4 = 5;

// Ultrasonic Sensor Pins

const int trigFront = 7;

const int echoFront = 6;

const int trigBack = 13;

const int echoBack = 8;

const int trigRight = 12;

const int echoRight = 11;

// Servo Motor Pin

const int servoPin = 9;

Servo myServo;

// Switch Pin

const int switchPin = 10;
```

```
// Variables

long duration, distance;

const long ULTRASONIC_TIMEOUT = 30000L; // Timeout in microseconds (30ms)

const float SOUND_SPEED = 0.0343 / 2;   // Speed of sound in cm/us (divided by 2 for one-
way travel)

// Flag to track if the system is on or off

bool systemOn = false;

// Flag to track the previous state of the switch

bool previousSwitchState = true; // Assuming the switch is initially in the OFF state

char reading; // data type used to store a character value.

 void setup( )

 {

  pinMode(in1, OUTPUT);

  pinMode(in2, OUTPUT);

  pinMode(in3, OUTPUT);

  pinMode(in4, OUTPUT);

  pinMode(trigFront, OUTPUT);

  pinMode(echoFront, INPUT);

  pinMode(trigBack, OUTPUT);

  pinMode(echoBack, INPUT);

  pinMode(trigRight, OUTPUT);

  pinMode(echoRight, INPUT);

  pinMode(switchPin, INPUT_PULLUP); // Ensure the switch pin is properly configured
  myServo.attach(servoPin);

  Serial.begin(9600);

  // Serial.begin(9600); //begin serial communications at 9600 bits per second (baud)

 }

 void loop( ) {

  if(Serial.available()) {

    reading =Serial.read();

    Serial.println(reading);

    if (reading == 'A'){

       while (digitalRead(switchPin) == LOW) { // Switch pressed (assuming pull-up
configuration)

          Serial.println("Switch pressed, starting parking sequence");

          // Move forward until space is detected

          if(getDistance(trigFront, echoFront)>30 && getDistance(trigBack, echoBack)>10 &&
getDistance(trigRight, echoRight)>20){

            moveForward();

          }

          else if(getDistance(trigFront, echoFront)<30 && getDistance(trigBack, echoBack)>10
&& getDistance(trigRight, echoRight)<20){

            stopCar();

            delay(200);

          // Start parking sequence

            Serial.println("Parking space detected, starting parking maneuver");

          // Move backward diagonally

            moveBackwardRight();
            delay(2150); // Adjust time for the desired angle and position
            moveBackward();
            delay(500);
          }
          else if(getDistance(trigBack, echoBack)>10){
            moveBackward();
          }
          else{
              stopCar();
            }
       break;
          }
    }
      else if (reading == 'O'){
       bool currentSwitchState = digitalRead(switchPin) == LOW; // LOW when pressed
       // If the switch state has changed
       if (currentSwitchState != previousSwitchState) {
          // Toggle the system state
          systemOn = !systemOn;
          // Update the previous switch state
          previousSwitchState = currentSwitchState;
          // Add a short delay to debounce the switch
          delay(50);
      }
      // If the system is off, stop motors and return from the loop
      if (!systemOn) {
        stopCar();
        return;
      }
      // If obstacle detected within 30cm, stop and turn
      if (getDistance(trigFront, echoFront) <= 30) {
        stopCar();
        delay(500); // Delay to stop the robot
        turnLeft(); // Turn left to avoid obstacle
        delay(1000); // Delay for turning left
      } else {
        // If no obstacle detected, move forward
        moveForward();
      }
    }
    else {
      stopCar();
    }
  }
}
```

```
void moveForward() {
  Serial.println("Moving forward");
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}
void moveBackward() {
  Serial.println("Moving backward");
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}
void moveBackwardRight() {
  Serial.println("Moving backward right");
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}
```

```
void stopCar() {

  Serial.println("Stopping car");

  digitalWrite(in1, LOW);

  digitalWrite(in2, LOW);

  digitalWrite(in3, LOW);

  digitalWrite(in4, LOW);

}

int getDistance(int trigPin, int echoPin) {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);

  distance = duration * 0.034 / 2;

  Serial.print("Distance: ");

  Serial.print(distance);

  Serial.println(" cm");

  return distance;

}
```

```
void turnLeft() {

  digitalWrite(in1, LOW);

  digitalWrite(in2, HIGH);

  digitalWrite(in3, LOW);

  digitalWrite(in4, HIGH);

}
```

The provided code includes two main functionalities for the robot: obstacle avoidance and auto-parking. The robot is controlled using an Arduino board, with commands received via Bluetooth to switch between the two modes.

- **Libraries:** The code utilizes the Servo library for controlling the servo motor.
- **Pin Definitions:** Pins are defined for motor control, ultrasonic sensors, servo motor, and on/off switch.
- **Constants:** Constants are defined for the ultrasonic sensor, including timeout and speed of sound.
- **Variables:** Variables are declared to store distance readings from the ultrasonic sensor and to track obstacle detection.
- **Setup Function:** Initializes pins, attaches the servo motor, sets its initial position, and starts serial communication.
- **Loop Function:** The loop() function in the Arduino program is the core of the robot's behavior. It continuously executes and handles two primary modes of operation: autoparking and obstacle avoidance.

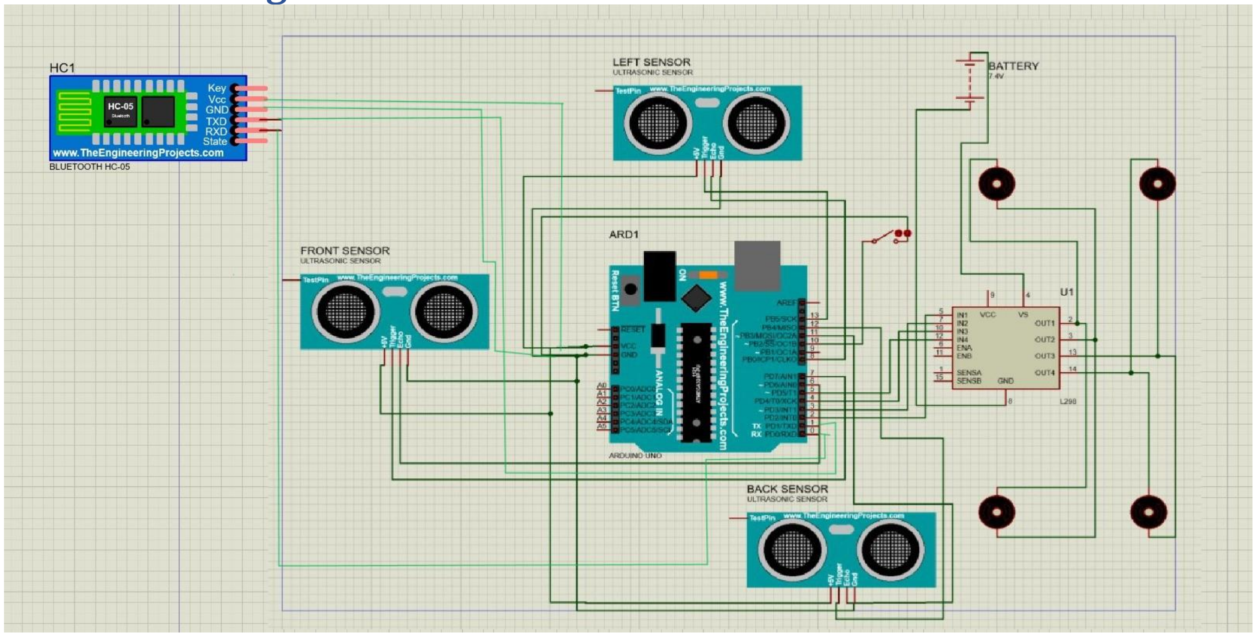
    - The function first checks if there is any data available from the serial port, which indicates a command from the Bluetooth module.
    - If data is available, it reads the data into the reading variable and prints it to the Serial Monitor for debugging purposes.
    - When the Bluetooth command 'A' is received, the robot enters autoparking mode.
    - It continuously checks if the parking switch is pressed.
    - The robot moves forward until it detects a suitable parking space. This is determined by the readings from the front, back, and right ultrasonic sensors.
    - Upon detecting a parking space, the robot stops and begins a parking maneuver by moving backward and diagonally to align itself into the parking space.
    - If the parking space is not fully clear, it continues to move backward until it is properly parked.
    - When the Bluetooth command 'O' is received, the robot enters obstacle avoidance mode.
    - The robot checks the state of the parking switch to toggle the system on or off.
    - If the system is toggled off, the robot stops all motors and exits the loop.

- If the system is on, it continuously checks for obstacles in front of the robot using the front ultrasonic sensor.
- If an obstacle is detected within 30 cm, the robot stops, waits for a short period, and then turns left to avoid the obstacle.
- If no obstacle is detected, the robot continues to move forward.
- If the received Bluetooth command is neither 'A' nor 'O', the robot defaults to stopping all motors, ensuring it remains stationary.

- **Functions:**

  - **Measure Distance:** Sends ultrasonic pulses, measures the duration for the echo, and calculates the distance based on the speed of sound.
  - **Stop Car:** Stops all motor movements.
  - **Move Forward:** Moves the robot forward by setting in1 and in3 to high.
  - **Move Backward:** Moves the robot backward by setting in2 and in4 to high.
  - **Move BackwardRight:** Moves the robot backward and to the right by setting in1,in3,in4 to low and in2 to high.
  - **Turn Left:** Turns the robot left to avoid obstacles by setting in2 and in4 high.

6 Schematic Diagram


HC1
HC-05
Key
Vcc
GND
TXD
RXD
State
www.TheEngineeringProjects.com
BLUETOOTH HC-05
LEFT SENSOR
ULTRASONIC SENSOR
BATTERY
7.4V
ARD1
FRONT SENSOR
ULTRASONIC SENSOR
U1
L298
BACK SENSOR
ULTRASONIC SENSOR
ARDUINO UNO