Auto Parallel Parking car

Team 2

Name	ID
Alaa Amer Mohammed	20200090
Sama Hussien Abo-Elala	20200232
Marwa Shaaban Eid	20200516
Mina Makram Fathy	20210603

Under the supervision of
Dr/Mustafa Shiple

Abstract

The purpose of this project is to create a simple prototype parking system capable of parallel parking on its own. This system would consist of a number of proximity sensors as well as a main processing unit (micro controller) that would operate the vehicle. The project is separated into two parts: the hardware (vehicle itself) and the software that operates it. Parking is handled with the help of three distance sensors and is located utilizing a Wi-Fi module. The automatic parking system gathers information about available parking spaces, processes it, and then places the automobile in a certain location. It is unavoidable for people to keep up with evolving technology, and in general, people are having difficulty parking their vehicles in parking spaces. An Arduino microprocessor, which is attached to these sensors, receives the output voltages from the sensors. The car then travels in accordance with the sensor values, and it is directed by its two front wheels.The wheels of the car are powered by a simple electric motor with a driver (H-bridge) . The H-bridge allows the motor to change direction and rotate at different speeds.Finally, the automatic car parking locates the nearest parking place for our user. This paper has provided a viable solution to the parking issue.

Background

Materials:

Arduino Uno
L298N to Motor driver
3Ultra sonic sensors
Breadboard 400 points
Battery Case Holder 4 cells
ON/OFF Switch 2 Pins Dim: 17x11mm
4Rechargeable Li-ion Battery 18650 (3.7V, 2400mAh) Full Charge 4.2V
2 wheels car with their DC-motors
10”Wires 20 Cm“ Male To Male Pins 10”Wires 20 Cm“ Male to Female Pins 10”Wires 20 Cm“ Female To Female Pins

Arduino Uno

Microprocessor board operates as the 'brain' of this project. The board is made up of the following components: a little chip, an ATmega328 microprocessor—wired in a small circuit that connects it to a USB port for uploading sketches, a battery port for receiving power, and input/output pins. The board in this project receives data from the range finders and uses pulse width modulation to operate each of the car's two wheels.THE MENLO ROUNDTABLE 61 The Arduino microprocessor is powered by a Processing language sketch written in the free and open source "Integrated Development Environment." When the sketch is downloaded to the Arduino board, it is translated into C and sent to the avr-gcc compiler, a piece of software that does the final translation into the language understood by the micro controller. The UNO board has 14 digital input/output pins (numbered 0 through 13) that can be used for both input and output, including pulse width modulation. The board is powered by a USB connection to a computer or an external power supply. It can be powered by an external power supply ranging from 6 to 20 volts; however, more than 7 volts is preferred for a consistent 5V supply from the 5V pin. The front of the Arduino UNO is shown in Figure 1 .

Figure 1

L298N to Motor driver

H-Bridge Integrated Circuit

A basic H-Bridge is made up of four switches that allow electricity to be applied in both directions across a load. These switches are frequently two PNP and two NPN, as illustrated in the diagram below.

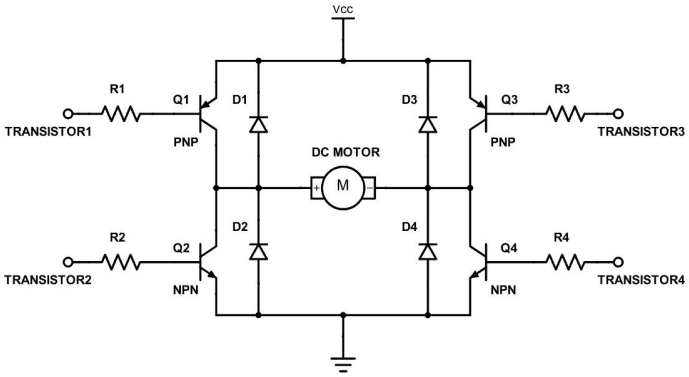

Figure 2: A transistorized H-bridge

The L298N H-Bridge chip in this project regulates both the direction and speed of the automobile engine. The H-Bridge requires a minimum of 12V and can function at voltages as high as 55V. The motor's direction is controlled by setting a digital pin on the Arduino to high or low. The motor's speed is controlled by pulse width motion.

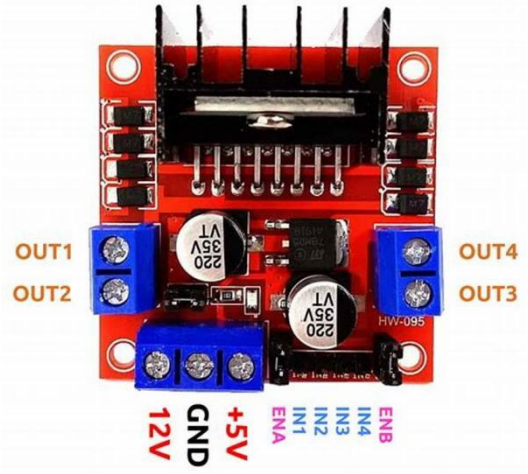

Figure 3

Ultra sonic sensor

The purpose of an ultrasonic sensor is to detect the distance of a target object by producing ultrasonic sound waves and translating the reflected sound into an electrical signal. Ultrasonic waves travel at a faster rate than audible sound. The two basic components of ultrasonic sensors are the transmitter, which generates sound using piezoelectric crystals, and the receiver, which encounters the sound after it has traveled to and from the target.

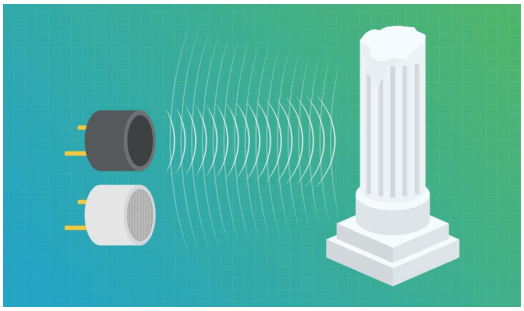

Figure4:Ultrasonic Sensors

Ultrasonic sensors work by generating a sound wave at a frequency that is higher than the human hearing range. The sensor's transducer acts as a microphone to receive and transmit ultrasonic sound. Our ultrasonic sensors, like many others, use a single transducer to send a pulse and receive the echo. By measuring the time interval between delivering and receiving the ultrasonic pulse, the sensor estimates the distance to a target. The operation of this module is simple. It releases a 40kHz ultrasonic pulse that passes through the air and bounces back to the sensor if it finds a barrier or object.You may calculate the distance by multiplying the trip time by the speed of sound.The ultrasonic sensor is shown in Figure 5.

Figure 5

Those are our main hardware components we'll be using through our project the arduino is micro controller , H-bridge to control the wheels direction and speed and the ultrasonic sensors to detect obstacles, parking slot and help park properly.

Proposed Idea

First Prototype The first prototype of the car was built as a first attempt to wire motors with transistors and control them with an Arduino. It was also a first attempt to code motors to respond according to range finder data. First prototype of model car is the schematic of each of the model car’s two wheels wired to the Arduino. And we have a switch, to control when car moves. The precise turns that a car needs to make to parallel park successfully depends largely on its own dimensions. Since parking algorithm will work in a small prototype car. The algorithm will be tailored

to it, but this does not mean that it will not work on cars with other dimensions; this only means that it will be most precise for this car size. The mechanism which control the basic motor rotation is move on wheel forward and the other wheel backward. This mechanism only allowed the wheels to turn to the most extreme left and right positions, whereas a car that can parallel park would need to have wheels that could rotate to various precise degrees. To solve this problem,we used a delay by trial and error to get a suitable delay considering the car speed.The sensor placement is shown in Figure 6.

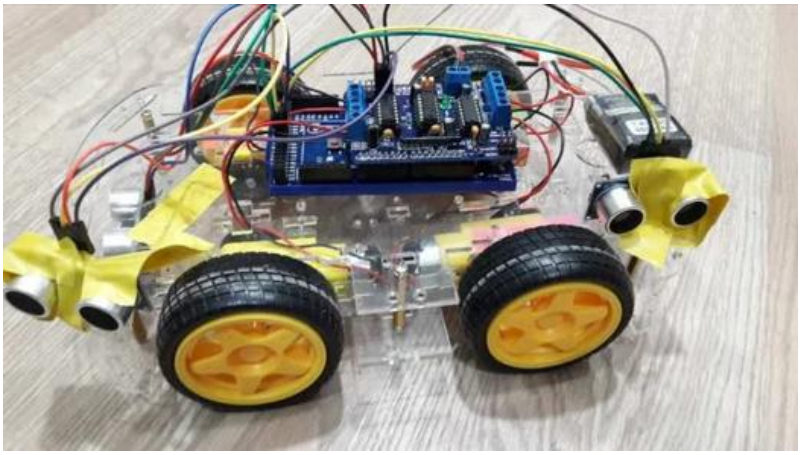

Figure6

Each sensor is connected to its own analog pin on the Arduino, so that each pin can read in specific outputs from their corresponding sensors. The rotation speed and direction of the wheels are controlled by an H-Bridge. The final circuit diagram is shown in Figure 7.

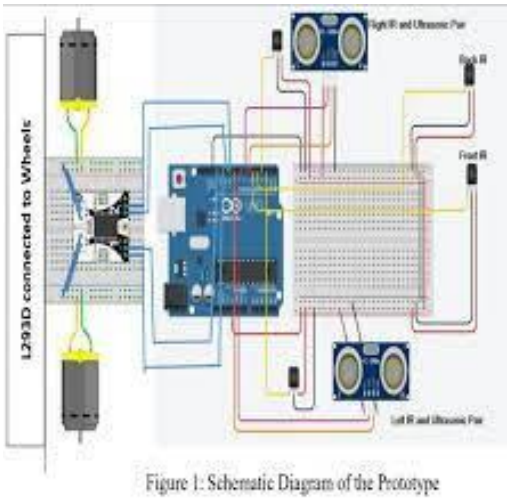

Figure 1: Schematic Diagram of the Prototype

Figure7

Once each of these elements was installed on the car, many tests were performed to ensure that the car was responding correctly to various external conditions picked up by the proximity sensors. For example, one piece of test code was written so that

if two sensors picked up distance readings greater than a certain distance, the wheels would spin one direction, and if this condition was not met, the wheels would spin the other direction. Powering both the H-Bridge and the Arduino with batteries turned out to be the biggest challenge. The H-Bridge requires a minimum of 12V and draws anywhere from 0.23A to 0.51A. The Arduino powers the sensors required 0.45A of current. The first attempt to power the H-Bridge was with a single 12V battery with about a 55mAh capacity. There was no effect. The second attempt was connecting several 12V batteries in parallel to increase the amount of current they provided. Again, no effect was observed. Finally, two 12V batteries were wired in series to create the effect of 24V shown in Figure 8.

Figure 8:2 wheels car with 4 batteries holder

And then several of these 24V pairs were wired in parallel to provide enough current. Though this setup was successful in powering, it was very inconsistent. Powering both the Arduino and H-bridge with power supplies is a much more reliable method. Finally, once the car was responding appropriately to various external conditions, the parallel parking sketch was written and then uploaded onto the Arduino. The algorithm is as follows.

Motion 1: Finding a Parking Space

a) Pre-condition:

- The car begins on the right side of both the parked cars and is parallel to the curb. The front of the parking car is between both parked cars.

b) Execution:

- The code initiates the car to move straight forward. When both the right sensor (`trigPin2`, `echoPin2`) and front right sensor (`trigPin3`, `echoPin3`) register distances less than 10 cm, the car stops.

c) Post-condition:

- The car is stopped next to the front car.

Motion 2: Reversing Toward Curb

a) Pre-condition:

- Post-condition of Motion 1.

b) Execution:

- The front wheels turn all the way to the right. The car reverses at this angle until the rear-right corner is within 15 cm of the curb.

c) Post-condition:

- The car is stopped at an angle with the rear-right corner close to the curb.

Motion 3: Straightening Out

a) Pre-condition:

- Post-condition of Motion 2.

b) Execution:

- The front wheels turn all the way to the left. The car reverses at this angle until the rear sensor (`trigPin1`, `echoPin1`) is within 5 cm of the car parked in the rear.

c) Post-condition:

- The car is stopped parallel to the curb and 5cm away from the rear car.

Conclusion in 3 main motions:-

1. Motion 1: Finding a Parking Space:

The existing code in the `loop()` function already includes a section where the car moves forward and checks for a suitable parking space using the `Check()` function. You may want to adjust the distance condition in the `Check()` function to match the specified 9cm requirement.

2. Motion 2: Reversing Toward Curb:

After Motion 1, you can add a new function or modify the existing code to implement the turning of the front wheels to the right and reversing until the rear-right corner is within 10 cm of the curb.

3. Motion 3: Straightening Out:

Following Motion 2, you can integrate code to turn the front wheels to the left and reverse until the rear sensor is within 5cm of the car parked in the rear.

Software

We programmed our car using (arduino c) programming language on an arduino IDE. Our algorithm to park a car works on a certain steps these steps as follow:

1)Initial Move Forward:

The car starts moving forward for a specified duration (300 ms in this case).

2)Check for Suitable Parking Space:

The Check() function is called to check if there is a suitable parking space available. It does this by moving forward until it detects an obstacle at a distance less than 10 units (trial and error value). The variable space is incremented during this movement to keep track of the potential parking space width.

3)Park if Suitable Space Found:

If a suitable parking space is found (Check() returns 1) and the slot found is suitable to the car size space counter=27 (trial and error value), the car stops, waits for a short duration (1 second), and then initiates the parking sequence by calling the Park() function.

4)Parking Sequence (Park() function):
The car first moves backward for 300 milliseconds (MoveBack() function).
It then stops for 1 second.
The car turns left for 450 milliseconds (MoveLeft() function).
The car stops again for 1 second.
While the right ultrasonic sensor (trigPin1, echoPin1) detects an obstacle within a certain distance (greater than 10 units), the car keeps moving backward.
The car stops for 1 second.
The car then turns right for 350 milliseconds (MoveRight() function).
After stopping for a short duration (400 milliseconds), the car moves forward for 500 milliseconds (MoveForward() function).
Finally, the car stops for 1 second.

5)Exit Program:
After successfully parking, the program exits (exit(0)).

The ultrasonic sensors (UFun() function) are used to measure distances from obstacles during the various movements. The specific distances and duration for movements have been determined through trial and error for the given application. Adjustments to these values may be needed depending on the environment.

The provided code is demonstrate these steps

```
void loop() {

  delay(3000);
  MoveForward();
  delay(300);

  if (Check()==1&& space>27){//space is measured by trial and error to
check the right value ,it's varied by the speed
    space=0;
    Stop();
    delay(1000);
    Park();
    exit(0);
    }
     else {
    if(UFun(trigPin1,echoPin1)>4&&UFun(trigPin3,echoPin3)>4)
    {
      MoveForward();
```

```
    }
    else{
      Stop();
      }
    }
}
void Park(){//Parking algorithm
  MoveBack();
  delay(300);
  Stop();
  delay(1000);
  MoveLeft();
  delay(450);
   Stop();
  delay(1000);
  while(UFun(trigPin1,echoPin1)>10){
  MoveBack();
  }
  Stop();
  delay(1000);
  MoveRight();
  delay(350);
  Stop();
  delay(400);
  MoveForward();
  delay(500);
  Stop();
  delay(1000);
 //the delay after each instruction varies depending on our car speed

  }
int Check(){//Chicking the sutable parking space for the car width
  int Flag =0;
  while(UFun(trigPin2,echoPin2)>10){
    MoveForward();
    space++;
    Flag =1;
    }
  return Flag;
  }
```

And as we don’t have a wifi in arduino we used a wifi module to utilize the localization that wifi module provides so we can know the location of the car and park it in a certain park slot.
But in case you couldn’t provide a wifi module you could just use your phone hotspot as a trial .

```
#include <Arduino.h>
#ifdef ESP32
#include <WiFi.h>
#include <AsyncTCP.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>
#include <ESP32Servo.h>

#include <Math.h>

float a1,a2,b1,b2,c1,c2,det;
void trilaterate(float x1, float y1, float d1, float x2, float y2,
float d2, float x3, float y3, float d3) {
  a1 = 2 * (x2 - x1);
  b1 = 2 * (y2 - y1);
  c1 = d1 * d1 - d2 * d2 - x1 * x1 + x2 * x2 - y1 * y1 + y2 * y2;

  a2 = 2 * (x3 - x1);
  b2 = 2 * (y3 - y1);
  c2 = d1 * d1 - d3 * d3 - x1 * x1 + x3 * x3 - y1 * y1 + y3 * y3;

  det = a1 * b2 - a2 * b1;

  float resultX = (c1 * b2 - c2 * b1) / det;
  float resultY = (a1 * c2 - a2 * c1) / det;
  Serial.println("-----------------------------------------------------");
  Serial.print("Estimated Coordinates : ");
  Serial.print(resultX);
  Serial.print(", ");
  Serial.println(resultY);

}

void setup() {
  Serial.begin(9600);
}
const char* ssidArray[] = {"car1", "car2", "car3"};
float point[3][2] = {{0, 0}, {0, 10}, {5, 5}};
int arraySize = sizeof(ssidArray) / sizeof(ssidArray[0]);

void localization()
{
  double distance[3];
  int numNetworks = WiFi.scanNetworks();
  Serial.println("-----------------------------------------------------");
  for (int i = 0; i < numNetworks; i++)
```

```
  {
    // Serial.println(WiFi.SSID(i) + " | RSSI: " + String(WiFi.RSSI(i)));
    for (int j = 0; j < arraySize; j++) {
      if(String(WiFi.SSID(i)) == String(ssidArray[j]))
      {
        double s = (-59 - WiFi.RSSI(i)) / (10.0 * 3);
        distance[j] = pow(10.0, s);
        // Serial.println("Whole Term: " + String(s));
        Serial.println(String(ssidArray[j]) + " with RSSI = " +
String(WiFi.RSSI(i)));
        Serial.println("has distance of = " + String(distance[j]) +
"\n");
      }
    }
  }

  trilaterate(point[0][0],point[0][1],distance[0],point[1][0],point[1][1]
,distance[1],point[2][0],point[2][1],distance[2]);
}

int flag=1;
void loop() {
  // Do other tasks if needed
  if(flag)localization();
  flag=0;
}
```

The provided code is for mathematical localization using esp32 .

Application

As we used esp32 at first we used a html page to control our car. But for our arduino implementation we didn't use an application to control the car.