Car controller

MCE 411: MECHATRONICS SYSTEM DESIGN II

Instructor: Dr. Noel Maalouf

15/05/2022

Iyad Baz 201901551

Reda Atoui 201905047

Elie Feghaly 201901228

Karim Kaouk 201900327

Table of Contents

Description 5
Hardware 6
Distance Controller 11
The fuzzy controller 14
Data Set-up 15
Fuzzy rules 16
Defuzzification 18
System Evaluation 19
The distance controller 21
Acceleration measurement 22
Distance measurement 25
Displaying distance and acceleration 25
Self-Parking Car 27
Car Steering 32
Stepper motor control 33
User Interface 35
Voice Control App 37
Main loop 40
Obstacles 41

Table of Figures

Figure 1 Arduino 6
Figure 2 Motor driver 6
Figure 3 MPU6050 6
Figure 4 Ultrasonic sensor 7
Figure 5 Stepper motor 7
Figure 6 Stepper motor driver 7
Figure 7 Bluetooth module 8
Figure 8 LCD Screen 8
Figure 9 DC motors 8
Figure 10 Gears 9
Figure 11 Distance Error membership functions 11
Figure 12 Acceleration Error membership functions 11
Figure 13 Acceleration output membership functions 12
Figure 14 Fuzzy rules 12
Figure 15 Output surface 12
Figure 16 Fuzzy variables set-up 14
Figure 17 Setting up the membership functions and its calculations (triangular & trapezoidal) 14
Figure 18 Fuzzy support functions 15
Figure 19 Setting up the type & number of input/output membership functions 15
Figure 20 Creating the triangular and trapezoidal membership functions 16
Figure 21 Selecting the types of membership functions for inputs and outputs 16
Figure 22 Setting all rule weights to be equal 16
Figure 23 Input rules 17
Figure 24 Output rules 17
Figure 25 Setting input and output ranges 17
Figure 26 Setting up the function that returns the output according to the membership functions 18
Figure 27 Setting up centroid defuzzification method 18
Figure 28 Fuzzy evaluation code 19
Figure 29 Fuzzy evaluation code 20
Figure 30 Fuzzy evaluation code 21
Figure 31 Distance controller 21
Figure 32 Motor controller 22
Figure 33 Accelerometer code 22
Figure 34 Accelerometer set-up 23
Figure 35 Accelerometer set-up 23
Figure 36 Accelerometer set-up 24
Figure 37 Acceleration biasing 24
Figure 38 Ultrasonic code 25
Figure 39 LCD pin initialization and library installation 25
Figure 40 LCD screen code 26
Figure 41 LCD dimensions initialization 26
Figure 42 Parallel Parking Procedure 27

Figure 43 Defining the back ultrasonic pins 27
Figure 44 Check parking slot 28
Figure 45 Check parking slot 29
Figure 46 Checking parking slot length logic 30
Figure 47 Self-parking 31
Figure 48 Self-parking 31
Figure 49 Steering code 32
Figure 50 Stepper motor initialization and pin definition 33
Figure 51 Stepper motor clockwise control 33
Figure 52 Stepper motor anticlockwise control 34
Figure 53 Graphical User Interface 35
Figure 54 GUI code 36
Figure 55 GUI code 36
Figure 56 User interface 38
Figure 57 Application Backend 39
Figure 58 Main Loop 40
Figure 59 H-bridge module 41

Description

Our project is mainly car that can control the distance between its front and the car in front of it, by varying the PWM sent to the motors. The decisions of the PWM are made by mapping the acceleration output to the value of PWM that should be given, after a fuzzy controller returns the acceleration output in m/sec^2. Our fuzzy logic is a MAMDANI interference system that uses triangular and trapezoidal membership functions for its 2 inputs (distance & acceleration), and its 1 output (desired acceleration). The car also will include an emergency mode that will stop the car in cases where the distance in front of it becomes less than 20% of the desired one, to avoid crashes.

Additionally, our car will be programmed to park on command. In this case the car will search for a suitable place to park, and parallel park based on the inputs of the ultrasonic sensors on its edges.

The whole system will be controlled using will be controlled using ARDUINO, and a GUI that will facilitate testing and usage. Moreover, the system will contain an option that will allow voice control.

Hardware

1) ARDUINO ELEGOO MEGA 2560

Figure 1 Arduino

2) UK1122 Motor driver

Figure 2 Motor driver

3) MPU6050 (accelerometer/gyroscope/temperature sensor)

Figure 3 MPU6050

4) HC-SRO4 Ultrasonic sensors

Figure 4 Ultrasonic sensor

5) Stepper motor 28BYJ-48

Figure 5 Stepper motor

6) ULN2003 Stepper motor driver

Figure 6 Stepper motor driver

7) HC-06 Bluetooth module

Figure 7 Bluetooth module

8) LCD Screen

Figure 8 LCD Screen

9) 9V DC motors

Figure 9 DC motors

10) Gears

Figure 10 Gears

Functionality

Starting with the mechanical structure of the car, the steering is done using a DC motor whose rotation is limited by plastic barriers on both sides. The car will steer right and left when given a certain polarity, and then when the voltage across its terminals is 0 (IN3 and IN4 are 0/0 or 1/1), the wheels are taken to their default straight position using springs.

As for the main motor that drives the car, it is a 9V DC motor that moves forward or backward according to the voltage polarity given by the driver. The motor rotation in both directions is transferred to the wheel shaft using a gearbox. Its speed is dependent on the PWM given by the Arduino.

The brain of the system is an ARDUINO controller. It controls all actuators and sensors according to the code deployed.

Distance Controller

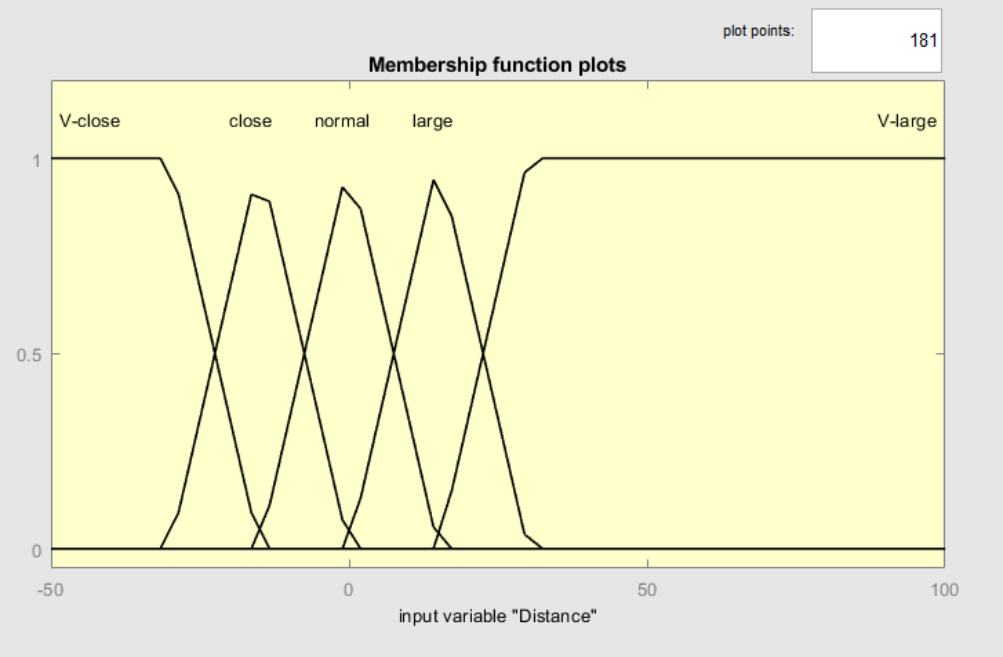

Figure 11 Distance Error membership functions

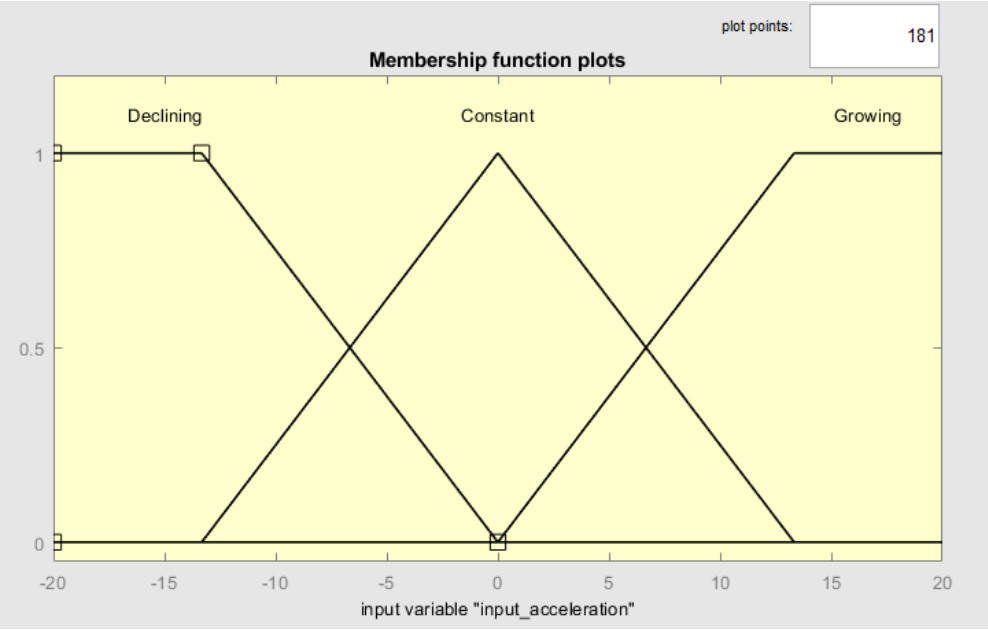

Figure 12 Acceleration Error membership functions

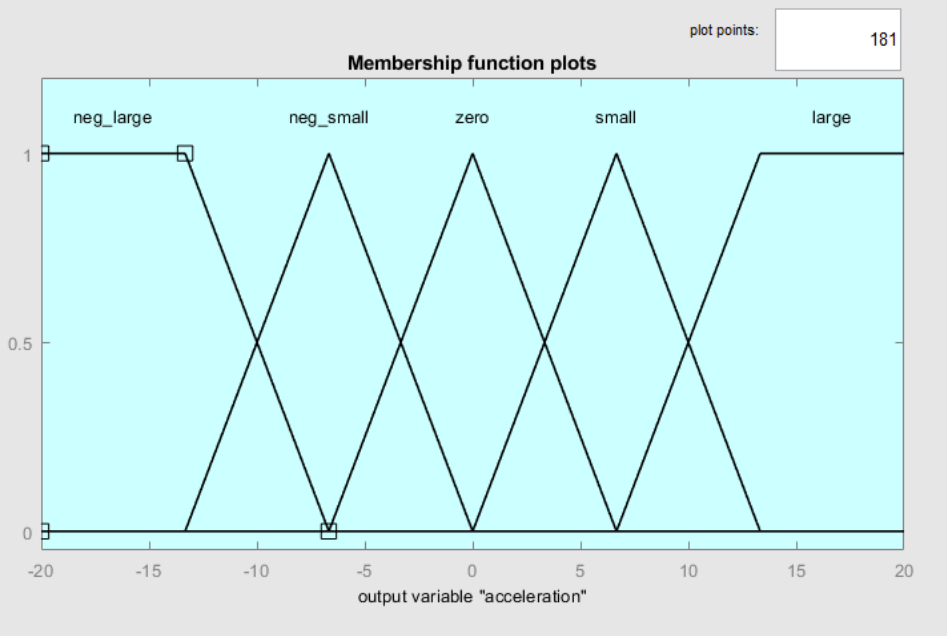

Figure 13 Acceleration output membership functions

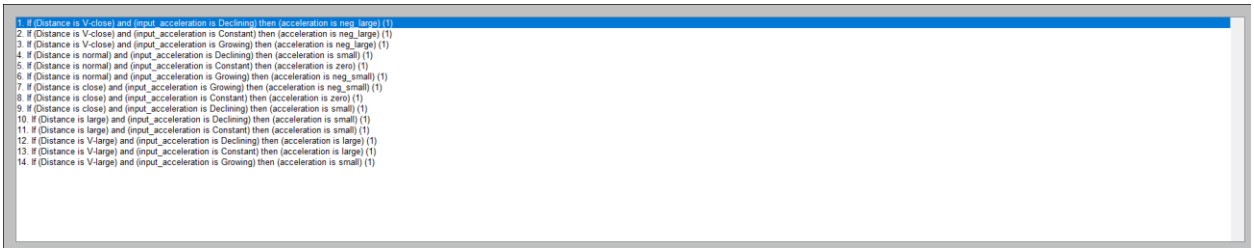

Figure 14 Fuzzy rules

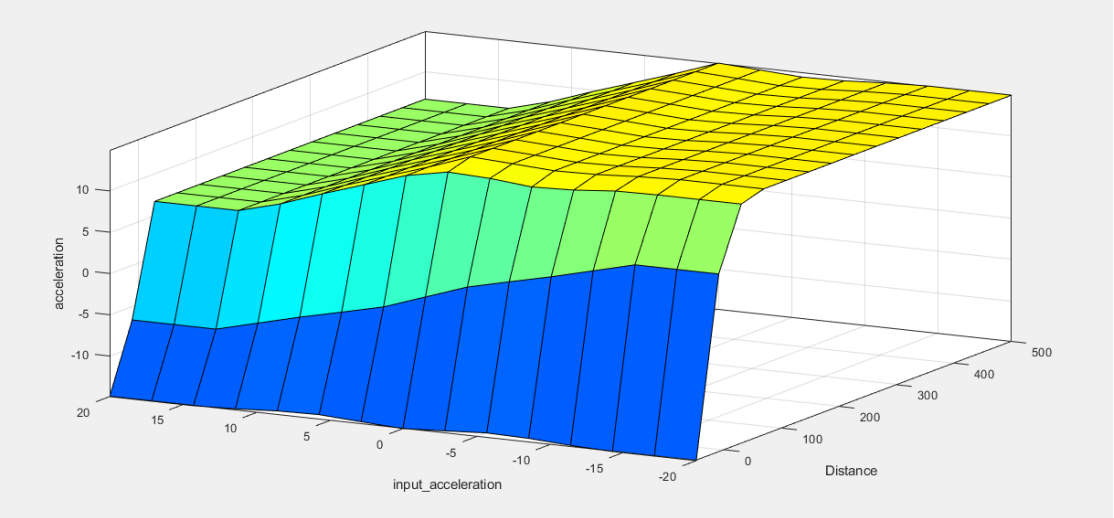

Figure 15 Output surface

Our fuzzy controller takes one input as the current acceleration of the car which is measured using MPU6050 accelerometer which is calibrated at the start of the car. The other input that the system takes is the distance error, which is the difference between the distance measured by the ultrasonic sensor set on the front of the car and the desired distance set by the user using the user interface we built.

The 2 inputs are then taken by the controller and the output is the acceleration needed to be given by the motor to achieve the desired distance. The output acceleration which has a range of -20 to 20 is then mapped to the PWM given to the motor which has a range from 0 to 255.

An additional emergency loop is accessed when the distance between the car and the car in front becomes less than 20% of the desired value. In this case, the car is given a PWM of 0, and thus stops.

Note:

The membership functions regarding the distance are varying according to the distance given through the user interface. This is done by multiplying the membership functions by a certain ratio and adding a certain offset.

The fuzzy controller

```
#define FIS_TYPE float
#define FIS_RESOLUSION 101
#define FIS_MIN -3.4028235E+38
#define FIS_MAX 3.4028235E+38
typedef FIS_TYPE(*_FIS_MF)(FIS_TYPE, FIS_TYPE*);
typedef FIS_TYPE(*_FIS_ARR_OP)(FIS_TYPE, FIS_TYPE);
typedef FIS_TYPE(*_FIS_ARR)(FIS_TYPE*, int, _FIS_ARR_OP);

// Number of inputs to the fuzzy inference system
const int fis_gcI = 2;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 1;
// Number of rules to the fuzzy inference system
const int fis_gcR = 15;

FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];
```

Figure 16 Fuzzy variables set-up

```
//************************************************************************
// Support functions for Fuzzy Inference System
//************************************************************************
// Trapezoidal Member Function
FIS_TYPE fis_trapmf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE a = p[0], b = p[1], c = p[2], d = p[3];
    FIS_TYPE t1 = ((x <= c) ? 1 : ((d < x) ? 0 : ((c != d) ? ((d - x) / (d - c)) : 0)));
    FIS_TYPE t2 = ((b <= x) ? 1 : ((x < a) ? 0 : ((a != b) ? ((x - a) / (b - a)) : 0)));
    return (FIS_TYPE) min(t1, t2);
}

// Triangular Member Function
FIS_TYPE fis_trimf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE a = p[0], b = p[1], c = p[2];
    FIS_TYPE t1 = (x - a) / (b - a);
    FIS_TYPE t2 = (c - x) / (c - b);
    if ((a == b) && (b == c)) return (FIS_TYPE) (x == a);
    if (a == b) return (FIS_TYPE) (t2*(b <= x)*(x <= c));
    if (b == c) return (FIS_TYPE) (t1*(a <= x)*(x <= b));
    t1 = min(t1, t2);
    return (FIS_TYPE) max(t1, 0);
}
```

Figure 17 Setting up the membership functions and its calculations (triangular & trapezoidal)

```
FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}

FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}

FIS_TYPE fis_array_operation(FIS_TYPE *array, int size, _FIS_ARR_OP pfnOp)
{
    int i;
    FIS_TYPE ret = 0;

    if (size == 0) return ret;
    if (size == 1) return array[0];

    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }

    return ret;
}
```

Figure 18 Fuzzy support functions

Data Set-up

```
_FIS_MF fis_gMF[] =
{
    fis_trapmf, fis_trimf
};

// Count of member function for each Input
int fis_gIMFCount[] = { 5, 3 };

// Count of member function for each Output
int fis_gOMFCount[] = { 5 };
```

Figure 19 Setting up the type & number of input/output membership functions

```
// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFI0Coeff1[] = { -Required_distance, -Required_distance, -0.6*Required_distance, -0.3*Required_distance};
FIS_TYPE fis_gMFI0Coeff2[] = { -0.3*Required_distance, 0, 0.3*Required_distance} ;
FIS_TYPE fis_gMFI0Coeff3[] = { 0.3*Required_distance, 0.6*Required_distance, 500, 500};
FIS_TYPE fis_gMFI0Coeff4[] = { -0.6*Required_distance, -0.3*Required_distance, 0 } ;
FIS_TYPE fis_gMFI0Coeff5[] = { 0, 0.3*Required_distance, 0.6*Required_distance} ;


FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2, fis_gMFI0Coeff3, fis_gMFI0Coeff4, fis_gMFI0Coeff5 };
FIS_TYPE fis_gMFI1Coeff1[] = { -20, -20, -13.33, 0 };
FIS_TYPE fis_gMFI1Coeff2[] = { -13.33, 0, 13.33 };
FIS_TYPE fis_gMFI1Coeff3[] = { 0, 13.33, 20, 20 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2, fis_gMFI1Coeff3 };
FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff };

// Coefficients for the Output Member Functions
FIS_TYPE fis_gMFO0Coeff1[] = { -20, -20, -13.33, -6.668 };
FIS_TYPE fis_gMFO0Coeff2[] = { -6.668, 0, 6.668 };
FIS_TYPE fis_gMFO0Coeff3[] = { 6.668, 13.33, 20, 20 };
FIS_TYPE fis_gMFO0Coeff4[] = { -13.33, -6.668, 0 };
FIS_TYPE fis_gMFO0Coeff5[] = { 0, 6.668, 13.33 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2, fis_gMFO0Coeff3, fis_gMFO0Coeff4, fis_gMFO0Coeff5 };
FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff };
```

Figure 20 Creating the triangular and trapezoidal membership functions

The data here in the first membership function represents that of the distance input, thus the membership functions are made according to certain percentages of the required distance for the code to be flexible and change according to the input.

```
// Input membership function set
int fis_gMFI0[] = { 0, 1, 0, 1, 1 };
int fis_gMFI1[] = { 0, 1, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1};

// Output membership function set
int fis_gMFO0[] = { 0, 1, 0, 1, 1 };
int* fis_gMFO[] = { fis_gMFO0};
```

Figure 21 Selecting the types of membership functions for inputs and outputs

Fuzzy rules

```
// Rule Weights
FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type
int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
```

Figure 22 Setting all rule weights to be equal

```
// Rule Inputs
int fis_gRI0[] = { 1, 1 };
int fis_gRI1[] = { 1, 2 };
int fis_gRI2[] = { 1, 3 };
int fis_gRI3[] = { 2, 1 };
int fis_gRI4[] = { 2, 2 };
int fis_gRI5[] = { 2, 3 };
int fis_gRI6[] = { 4, 3 };
int fis_gRI7[] = { 4, 2 };
int fis_gRI8[] = { 4, 1 };
int fis_gRI9[] = { 5, 1 };
int fis_gRI10[] = { 5, 2 };
int fis_gRI11[] = { 3, 1 };
int fis_gRI12[] = { 3, 2 };
int fis_gRI13[] = { 3, 3 };
int* fis_gRI[] = { fis_gRI0, fis_gRI1, fis_gRI2, fis_gRI3, fis_gRI4, fis_gRI5, fis_gRI6, fis_gRI7, fis_gRI8, fis_gRI9, fis_gRI10, fis_gRI11, fis_gRI12, fis_gRI13 };
```

Figure 23 Input rules

```
// Rule Outputs
int fis_gRO0[] = { 1 };
int fis_gRO1[] = { 1 };
int fis_gRO2[] = { 1 };
int fis_gRO3[] = { 5 };
int fis_gRO4[] = { 2 };
int fis_gRO5[] = { 4 };
int fis_gRO6[] = { 4 };
int fis_gRO7[] = { 2 };
int fis_gRO8[] = { 5 };
int fis_gRO9[] = { 5 };
int fis_gRO10[] = { 5 };
int fis_gRO11[] = { 3 };
int fis_gRO12[] = { 3 };
int fis_gRO13[] = { 5 };
int* fis_gRO[] = { fis_gRO0, fis_gRO1, fis_gRO2, fis_gRO3, fis_gRO4, fis_gRO5, fis_gRO6, fis_gRO7, fis_gRO8, fis_gRO9, fis_gRO10, fis_gRO11, fis_gRO12, fis_gRO13 };
```

Figure 24 Output rules

```
// Input range Min
FIS_TYPE fis_gIMin[] = { -Required_distance, -20 };

// Input range Max
FIS_TYPE fis_gIMax[] = { 500, 20 };

// Output range Min
FIS_TYPE fis_gOMin[] = { -20 };

// Output range Max
FIS_TYPE fis_gOMax[] = { 20 };
```

Figure 25 Setting input and output ranges

This step is also done according to the desired distance.

Defuzzification

```
FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
{
    FIS_TYPE mfOut;
    int r;

    for (r = 0; r < fis_gcR; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
            index = index - 1;
            mfOut = (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
            mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }

        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
    }
    return fis_array_operation(fuzzyRuleSet[0], fis_gcR, fis_max);
}
```

Figure 26 Setting up the function that returns the output according to the membership functions

```
FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
{
    FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o]) / (FIS_RESOLUSION - 1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;

    // calculate the area under the curve formed by the MF outputs
    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }

    return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) : (momentum / area));
}
```

Figure 27 Setting up centroid defuzzification method

System Evaluation

```
double fis_evaluate()
{
    FIS_TYPE fuzzyInput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyInput1[] = { 0, 0, 0 };
    FIS_TYPE* fuzzyInput[fis_gcI] = { fuzzyInput0, fuzzyInput1, };
    FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyOutput[fis_gcO] = { fuzzyOutput0, };
    FIS_TYPE fuzzyRules[fis_gcR] = { 0 };
    FIS_TYPE fuzzyFires[fis_gcR] = { 0 };
    FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
    FIS_TYPE sW = 0;

    // Transforming input to fuzzy Input
    int i, j, r, o;
    for (i = 0; i < fis_gcI; ++i)
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
            fuzzyInput[i][j] =
                (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i], fis_gMFICoeff[i][j]);
        }
    }
```

Figure 28 Fuzzy evaluation code

```
int index = 0;
for (r = 0; r < fis_gcR; ++r)
{
    if (fis_gRType[r] == 1)
    {
        fuzzyFires[r] = FIS_MAX;
        for (i = 0; i < fis_gcI; ++i)
        {
            index = fis_gRI[r][i];
            if (index > 0)
                fuzzyFires[r] = fis_min(fuzzyFires[r], fuzzyInput[i][index - 1]);
            else if (index < 0)
                fuzzyFires[r] = fis_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
            else
                fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
        }
    }
    else
    {
        fuzzyFires[r] = FIS_MIN;
        for (i = 0; i < fis_gcI; ++i)
        {
            index = fis_gRI[r][i];
            if (index > 0)
                fuzzyFires[r] = fis_max(fuzzyFires[r], fuzzyInput[i][index - 1]);
            else if (index < 0)
                fuzzyFires[r] = fis_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
            else
                fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
        }
    }
```

Figure 29 Fuzzy evaluation code

```
        fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
        sW += fuzzyFires[r];
    }

    if (sW == 0)
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
        }
    }
    else
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
        }
    }

    return g_fisOutput[0];
}
```

Figure 30 Fuzzy evaluation code

The distance controller

```
void Control_Distance() {
 // Read Input: Distance
    g_fisInput[0] = ultra_measure() - Required_distance;
    // Read Input: input_acceleration
    g_fisInput[1] = measure_acceleration() - acceleration_bias;

    g_fisOutput[0] = 0;
    Serial.print(g_fisInput[0]);
    Serial.print(" ");
    Serial.print( g_fisInput[1]);
    Serial.print(" ");
    Serial.println(fis_evaluate());


    if(ultra_measure() > 0.2*Required_distance){
      motor_control(true, fis_evaluate());
    }else {
      digitalWrite(enA, LOW);
    }
}
```

Figure 31 Distance controller

```
void motor_control(bool dir, double acceleration) {
  int potValue = acceleration; // Read potentiometer value
  int pwmOutput = map(potValue, -20, 20, 0 , 255); // Map the potentiometer value from 0 to 255
  analogWrite(enA, pwmOutput); // Send PWM signal to L298N Enable pin

  Serial.println(pwmOutput);
  // go forward
  if (dir == true  & rotDirection == 0) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    rotDirection = 1;
    delay(20);
  }
  // go backward
  if (dir == false & rotDirection == 1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    rotDirection = 0;
    delay(20);
  }
}
```

Figure 32 Motor controller

The motor control is given a Boolean to change the direction. True for forward motion and false for backward motion. This is done using the UK1122 driver model that is based on H-bridges that open and close circuits according to the inputs given (IN1, IN2, IN3, IN4), whenever the enable pins (ENA & ENB) are given a Boolean HIGH. The acceleration given is mapped to a PWM output which is given then to the motor.

The distance control function measure measures the current distance and gets the fuzzy output needed, which is then given to the motor controller.

Acceleration measurement

```
double measure_acceleration() {
  mpu.getEvent(&a, &g, &temp);
  delay(200);
  return a.acceleration.y;
}
```

Figure 33 Accelerometer code

The acceleration Is measured using the y component of the accelerometer in the MPU.

```
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);

'/accelerometer
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MPU6050 test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
```

Figure 34 Accelerometer set-up

```
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  Serial.print("Accelerometer range set to: ");
  switch (mpu.getAccelerometerRange()) {
  case MPU6050_RANGE_2_G:
    Serial.println("+-2G");
    break;
  case MPU6050_RANGE_4_G:
    Serial.println("+-4G");
    break;
  case MPU6050_RANGE_8_G:
    Serial.println("+-8G");
    break;
  case MPU6050_RANGE_16_G:
    Serial.println("+-16G");
    break;
  }
```

Figure 35 Accelerometer set-up

```
mpu.setFilterBandwidth(MPU6050_BAND_5_HZ);
Serial.print("Filter bandwidth set to: ");
switch (mpu.getFilterBandwidth()) {
case MPU6050_BAND_260_HZ:
  Serial.println("260 Hz");
  break;
case MPU6050_BAND_184_HZ:
  Serial.println("184 Hz");
  break;
case MPU6050_BAND_94_HZ:
  Serial.println("94 Hz");
  break;
case MPU6050_BAND_44_HZ:
  Serial.println("44 Hz");
  break;
case MPU6050_BAND_21_HZ:
  Serial.println("21 Hz");
  break;
case MPU6050_BAND_10_HZ:
  Serial.println("10 Hz");
  break;
case MPU6050_BAND_5_HZ:
  Serial.println("5 Hz");
  break;
}
```

Figure 36 Accelerometer set-up

The device is checked if functional before the code starts to execute, and it the system waits for its functionality to start. The accelerometer range and filter bandwidth are then set to 8GHz and 5Hz respectively, and the set specs of the accelerometer are printed to the serial to make sure everything Is functioning correctly.

```
acceleration_bias = measure_acceleration();
```

Figure 37 Acceleration biasing

In the beginning of the code, the initial acceleration is calculated, and all the other outputs are biased using this value, in case of any calibration errors.

Distance measurement

```
double ultra_measure() {
  digitalWrite(TRIGPIN,LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGPIN,HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGPIN,LOW);

  float distance = pulseIn(ECHOPIN, HIGH);
  distance = distance/58;
  // in cm
  delay(200);
  return distance;
}
```

Figure 38 Ultrasonic code

The ultrasonic trigger pin is set to LOW initially, then it's given a pulse after a delay of 2 microseconds. The pulse is then detected by the ECHO Pin and stored in a variable called distance, which is then divided by 58 to convert its output to cm.

The speed of sound is 340m/s or 29 us per cm. The Ultrasonic burst travels out & back. So to find the distance of the object we divide by 58.

Displaying distance and acceleration

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(41, 39, 37, 35, 33, 31);
```

Figure 39 LCD pin initialization and library installation

```
void display(double distance, double acceleration) {
  String Distance = "Dist: ";
  Distance.concat(distance);
  Distance.concat("cm");
  lcd.print(Distance);
  lcd.setCursor(0,1);
  String Acc = "Accel: ";
  Acc.concat(acceleration);
  Acc.concat("m/s2");
  lcd.print(Acc);
  delay(200);
  lcd.clear();
}
```

Figure 40 LCD screen code

```
lcd.begin(16, 2);
```

Figure 41 LCD dimensions initialization

The current distance with the car in front and the current acceleration are measured and displayed on the screen in cm and m/s^2 respectively.

Self-Parking Car

The parking method will be based on the following procedure:

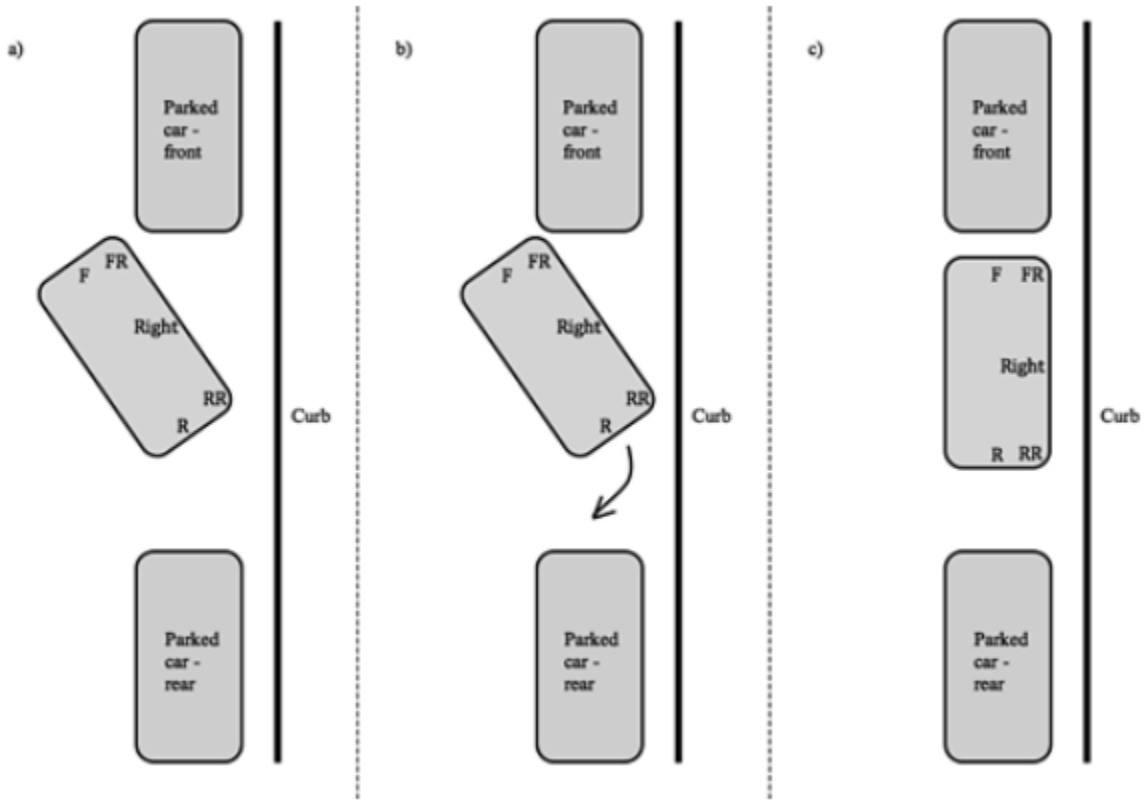

Figure 42 Parallel Parking Procedure

```
// define the pins used for ultrasonic sensors used for self parking
#define EchoBack 22;
#define TrigBack 24;
```

Figure 43 Defining the back ultrasonic pins

First, the car checks if there is enough space for it to park, through the following code:

```
// method to check if space is enough to park
bool check_park()
{
  bool check = false;

  // let the car move forward initially with a relatively slow speed
  // steer the ultrasonic to the side
  StepperCW(50);
  StepperCCW_B(50);
  motor_control(true,0);



  double distF = ultra_measure(TRIGPIN, ECHOPIN);
  double distB = ultra_measure(TrigBack, EchoBack);
  while (true){
     distF = ultra_measure(TRIGPIN, ECHOPIN);
     distB = ultra_measure(TrigBack, EchoBack);
     if(distF>9 && distB>9) {
      motor_control(true, -20);
      break;
     }
  }

  StepperCCW(44);

  int count = 0;
  int min_count;
  double current, next, d;
  next = ultra_measure(TRIGPIN, ECHOPIN);
  d = next;
```

Figure 44 Check parking slot

```
  while(count<19){
    next = ultra_measure(TRIGPIN, ECHOPIN);
    if(next < d) {
      d = next;
      min_count = count;
    }
    count = count + 1;
  }

  double angle = (min_count + 6) * 1.8;
  double l = 32; //car length

  if(d*cos(angle) > 0.5*l) {
    check = true;
  }

  StepperCCW(25); //retrun stepper to initial position
  return check;

}

// display message that the parking space is not enough
void display_error() {
  lcd.clear();
  String error = "NO PARK";
  lcd.print(error);
  delay(1000);
  lcd.clear();
}
```

Figure 45 Check parking slot

The car directs both ultrasonics 90° to the right (knowing that on Lebanese streets, car park to the right of the street). The car moves forward till both ultrasonics detect a large distance, which means there is a slot which is either equal or longer than the car itself. When the car reaches this position, it stops and directs its front ultrasonic 10° to the right from its initial position (which is directly to the front). The front ultrasonic is then rotated 35°, and the minimum distance and count are stored in variables d and min_count. The min_count variable is then used to figure out the angle theta, that will be used to calculate the total length of the slot according to the following logic:

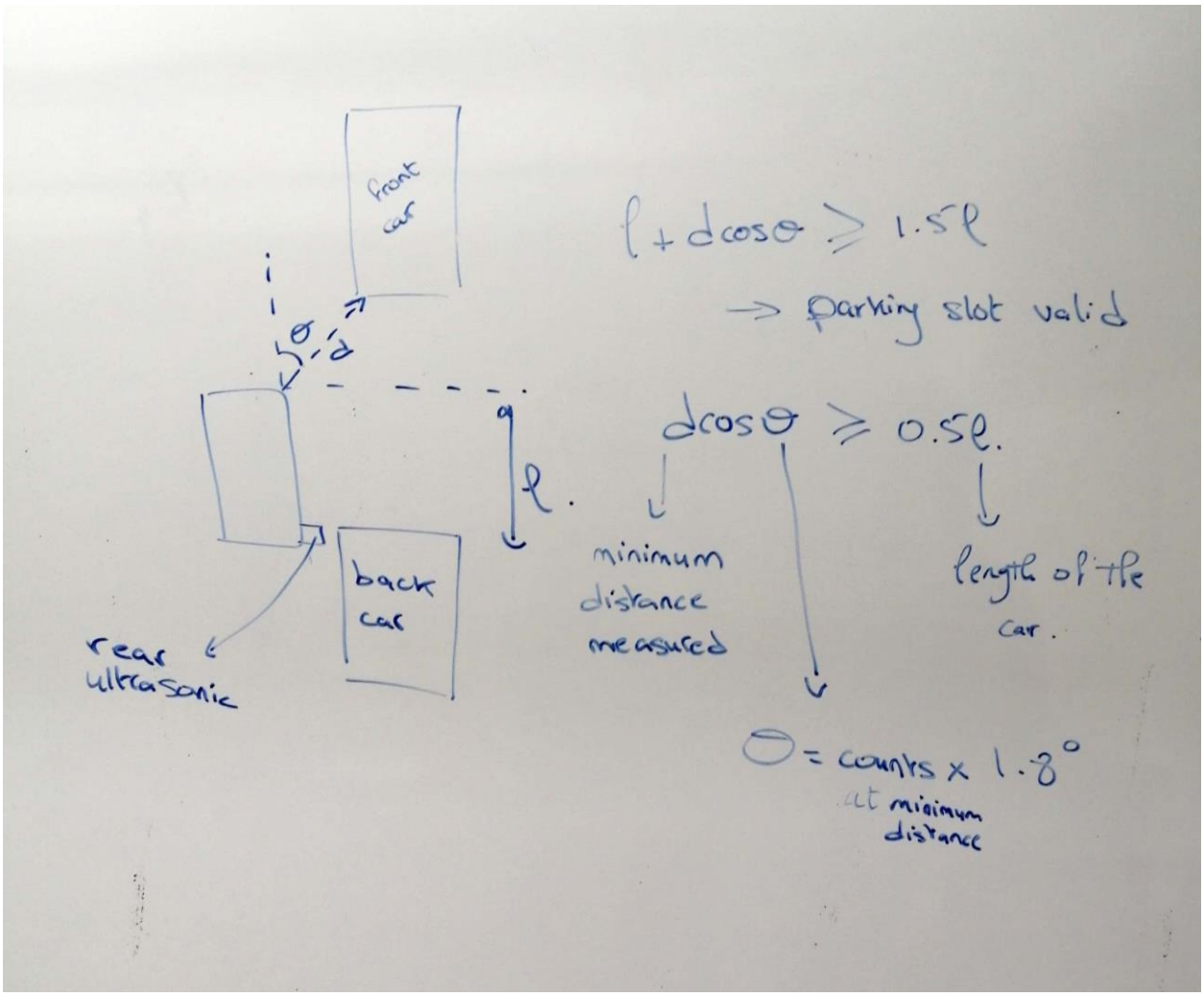

Figure 46 Checking parking slot length logic

The slot length is measured according to the logic above, and if found to be suitable for parking, the car will proceed into the next loop, which is the parking process. If not, an error message on the LCD will be displayed stating: “NO PARK”.

```
// self parking method
void park(bool check){
  if (check){

  double distB = ultra_measure(TrigBack, EchoBack);
  motor_control(true, 0);
  while(distB > 9){
    distB = ultra_measure(TrigBack, EchoBack);
    if(distB < 9) { //stop when back of the car coincides with the back of the front obstacle
      motor_control(true, -20);
      motor_control(false, 0); //get to a suitable position to start parking
      delay(100);
      motor_control(false, -20);
    }
  }
```

Figure 47 Self-parking

The car will move forward till its rear end is beside the rear end of the front car, then it will move backwards for a small distance (0.1s at a PWM of 125).

```
  //steer the back ultra sonic to measure the rear back-right
  StepperCCW_B(25);
  // let the car move backwards, to the right to park

  rear_back = ultra_measure(TrigBack, EchoBack);
   while(rear_back>4){
       steer(true, 100, false);
      rear_back = ultra_measure(TrigBack, EchoBack);
   }
      delay(500);
      analogWrite(enA,0);
      digitalWrite(in3, LOW);
      digitalWrite(in4, LOW);

  // steer wheel back when close to the wall
  // steer the rear ultrasonic back

  StepperCW_B(25);
  rear_back = ultra_measure(TrigBack,EchoBack);
  while(rear_back>4){
     steer(false, 100, false);
     rear_back = ultra_measure(TrigBack, EchoBack);
  }
    delay(500);
    analogWrite(enA,0);
    digitalWrite(in3, LOW);
    digitalWrite(in4, LOW);
  }
  else {
    display_error();
  }
}
```

Figure 48 Self-parking

The car is then steered to the right and moves till it's in an appropriate position to steer to the left and move backwards. It keeps moving backwards until it's parked well. Then everything is taken back to its initial position.

Car Steering

To control the steering of the car, the UK1122 motor driver is used to switch the polarities of the voltage across the motor and thus steer right and left according to the given direction. This is done using the following code:

```
void steer(bool rot, double time, bool dir) {
  int pwmOutput = 255; // Map the potentiometer value from 0 to 255
  analogWrite(enB, pwmOutput); // Send PWM signal to L298N Enable pin

  //true for right / false for left
  if(rot) {
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
    motor_control(dir, time);
    delay(time);
  }else {
    digitalWrite(in4, HIGH);
    digitalWrite(in3, LOW);
    motor_control(dir, time);
    delay(time);
  }
    digitalWrite(in4, LOW);
    digitalWrite(in3, LOW);
}
```

Figure 49 Steering code

The method takes the direction of rotation as Boolean input and sets the IN3 & IN4 pins of the motor driver to HIGH & LOW according to the desired direction.

The method also takes the time of the steer, for which the car will move forward and backward when the motor is steered.

The maximum steer of the wheels is limited by the structure of the car. By the end of the method, the pins IN3 and IN4 are set both too LOW to get the direction back to its default position (directly forward).

Stepper motor control

To change the direction of the ultrasonic and measure the distance to different sides, stepper motors are used, with ultrasonic sensors on them. The stepper motors are controlled as follows:

```
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

#define STEPPER_PIN_1 8
#define STEPPER_PIN_2 9
#define STEPPER_PIN_3 10
#define STEPPER_PIN_4 11
int step_number = 0;
```

Figure 50 Stepper motor initialization and pin definition

```
void StepperCW(int steps) {
  for(int i = 0; i < steps; i++) {
  switch(step_number){
        case 0:
        digitalWrite(STEPPER_PIN_1, HIGH);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
        case 1:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, HIGH);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
        case 2:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, HIGH);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
        case 3:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, HIGH);
        break;
      }
      step_number++;
      if(step_number > 3){
        step_number = 0;
      }
      delay(2);
  }
}
```

Figure 51 Stepper motor clockwise control

```
void StepperCCW(int steps) {
  for(int i = 0; i < steps; i++) {
    switch(step_number){
          case 0:
          digitalWrite(STEPPER_PIN_1, LOW);
          digitalWrite(STEPPER_PIN_2, LOW);
          digitalWrite(STEPPER_PIN_3, LOW);
          digitalWrite(STEPPER_PIN_4, HIGH);
          break;
          case 1:
          digitalWrite(STEPPER_PIN_1, LOW);
          digitalWrite(STEPPER_PIN_2, LOW);
          digitalWrite(STEPPER_PIN_3, HIGH);
          digitalWrite(STEPPER_PIN_4, LOW);
          break;
          case 2:
          digitalWrite(STEPPER_PIN_1, LOW);
          digitalWrite(STEPPER_PIN_2, HIGH);
          digitalWrite(STEPPER_PIN_3, LOW);
          digitalWrite(STEPPER_PIN_4, LOW);
          break;
          case 3:
          digitalWrite(STEPPER_PIN_1, HIGH);
          digitalWrite(STEPPER_PIN_2, LOW);
          digitalWrite(STEPPER_PIN_3, LOW);
          digitalWrite(STEPPER_PIN_4, LOW);
      }
        step_number++;
        if(step_number > 3){
          step_number = 0;
        }
        delay(2);
    }
}
```

Figure 52 Stepper motor anticlockwise control

User Interface

For the GUI, we used Visual Studio to design our interface. We made use of different tools and designs to make it aesthetic and pleasing to the eye. Regarding the code behind it, each button will be sending a string consisting of one character that will later be caught by the main code and its respective action will be taken. The submit button, however, will be sending the distance value appended to the specific character.

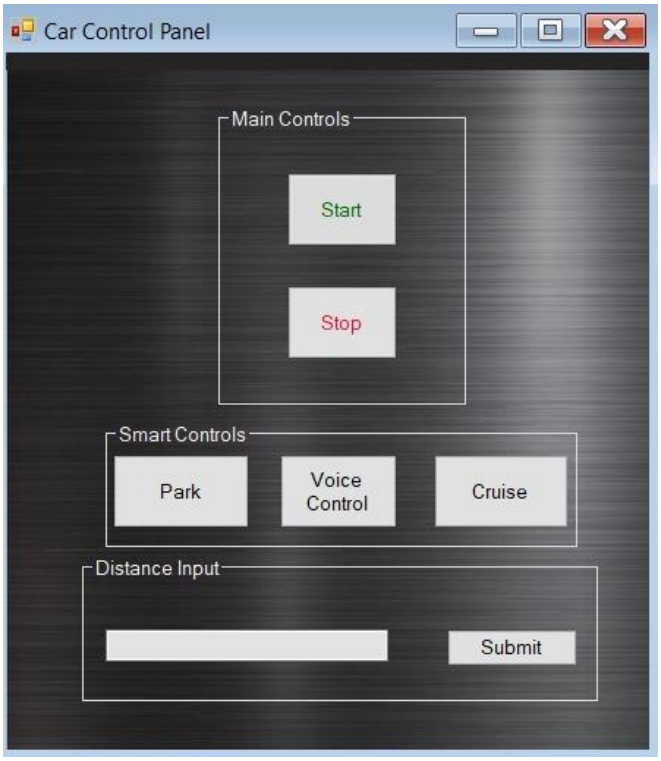

Figure 53 Graphical User Interface

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CarGUI
{
    3 references
    public partial class Form1 : Form
    {
        1 reference
        public Form1()
        {
            InitializeComponent();
            serialPort1.Open();
        }
```

Figure 54 GUI code

```
1 reference
private void button2_Click(object sender, EventArgs e)
{
    serialPort1.Write("s");
}

1 reference
private void button3_Click(object sender, EventArgs e)
{
    serialPort1.Write("c");
}

1 reference
private void button5_Click(object sender, EventArgs e)
{
    serialPort1.Write("p");
}
1 reference
private void button1_Click(object sender, EventArgs e)
{
    serialPort1.Write("S");
}

1 reference
private void Submit_Click(object sender, EventArgs e)
{
    string m1 = "D" + textBox1.Text;
    serialPort1.Write(m1);
}
1 reference
private void button4_Click(object sender, EventArgs e)
{
    serialPort1.Write("v");
}
```

Figure 55 GUI code

Voice Control App

Using MIT app builder, we created a basic app to connect our smart phones to the Arduino Bluetooth module called HC-05. The user interface of the app along with the back end are shown below. After selecting the HC-05 module from the list of available Bluetooth devices, the app will be connected, and we can now command the car via voice recognition.

As a first input, the user is responsible for entering a required distance. So, the first vocal input will be a string with the form: “Distance Control ###”.

The second vocal command will be the functionality of the car.

- “drive”: the car will enter cruise control mode where its speed and acceleration will be controlled.
- “park”: the car will enter the self-parking mode
- “forward”: the car will simply drive forward with no control
- “backward”: the car will simply drive backward with no control
- “stop”: the car will stop moving

The above sequence of the logical operations is concluded in the main loop shown below.

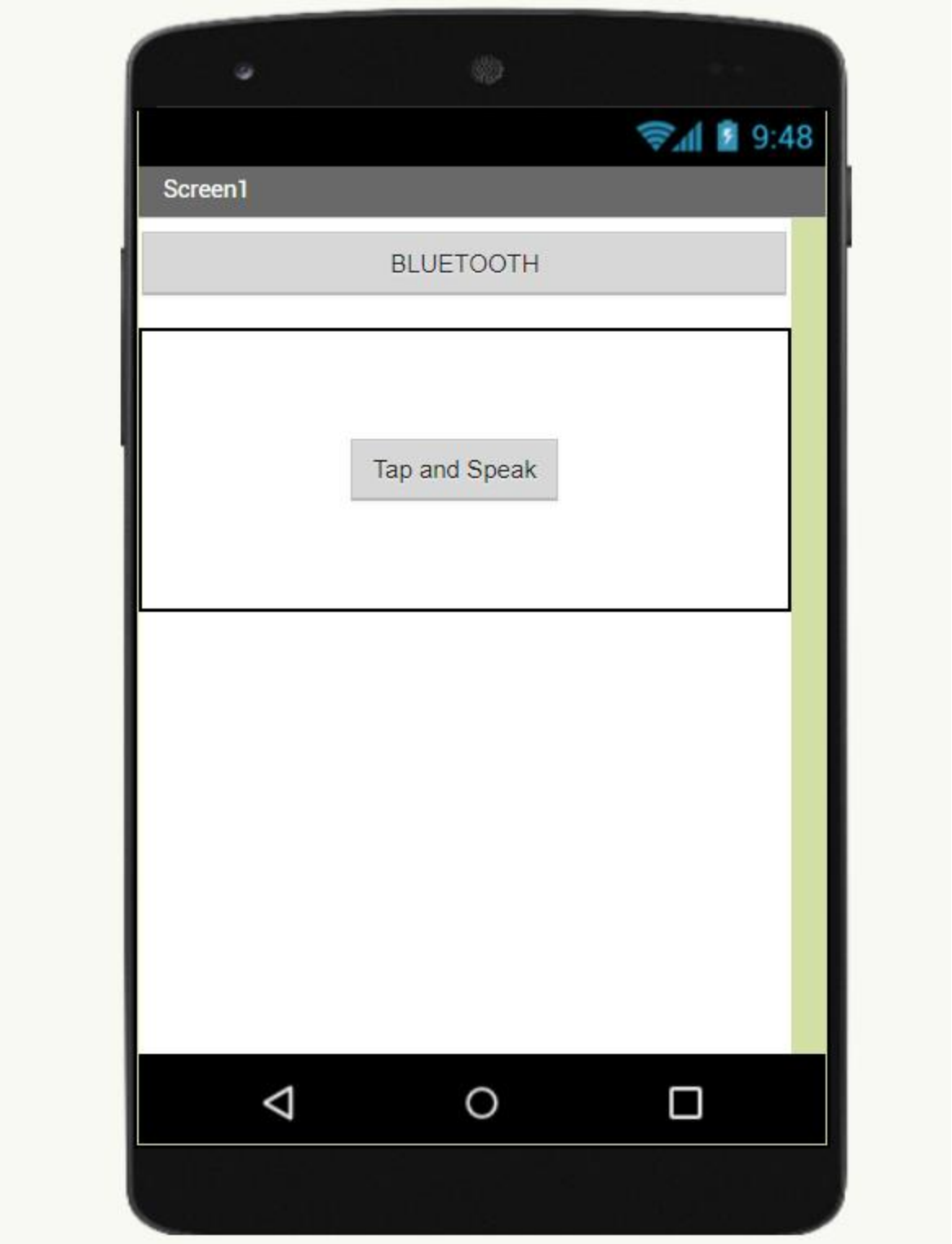

Figure 56 User interface

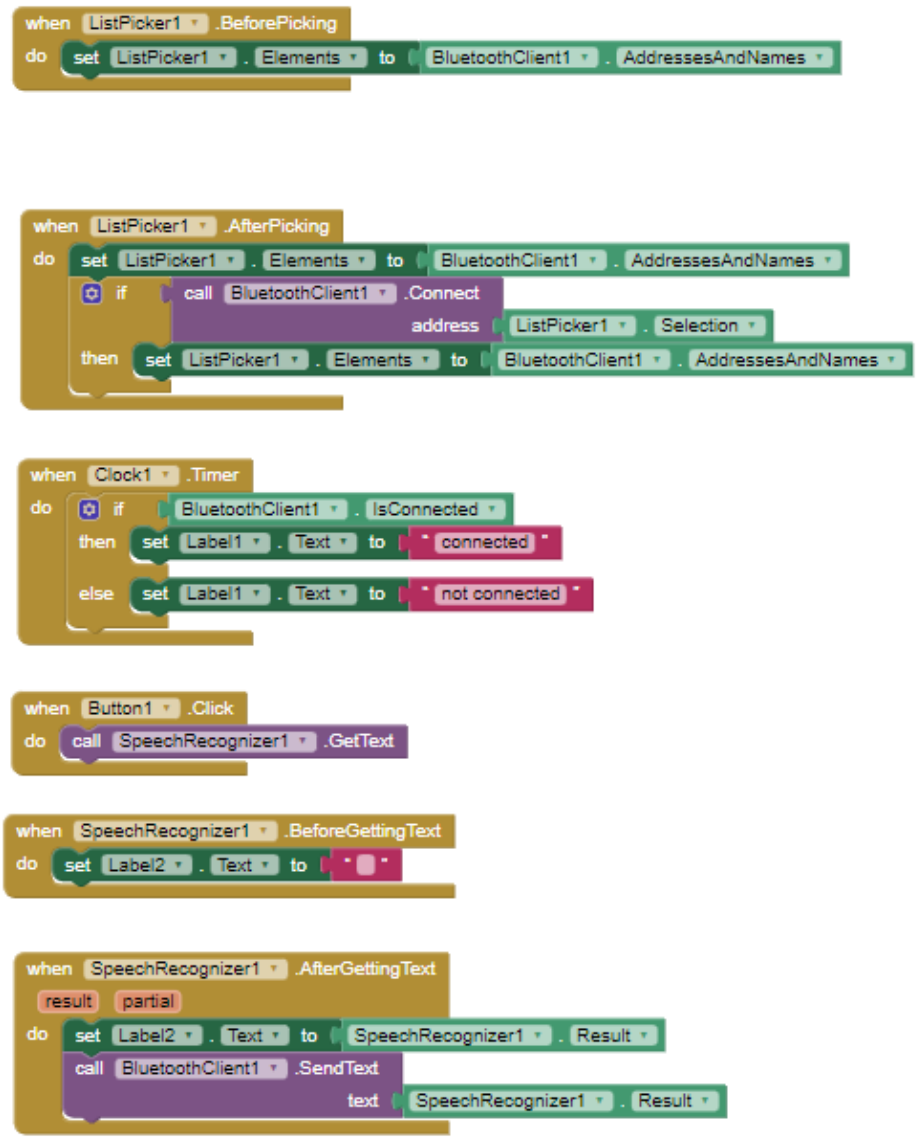

Figure 57 Application Backend

Main loop

The main loop decides which code should execute (Distance controller/ Self-parking/ other functions.) according to the user input from the GUI, or the voice input.

```
void loop()
{

  Serial.println(value);
  if(bluetooth.available()) {
     value = bluetooth.readString();
     if(value.substring(0,16) == "Distance Control"){

      Required_distance = value.substring(20,value.length());
     }
     value = bluetooth.readString();
      if (value == "drive") {
        double distance = ultra_measure(TRIGPIN,ECHOPIN);
        double acceleration = measure_acceleration()-acceleration_bias;
        display(distance, acceleration);

        Control_Distance();
      }

    if (value == "park"){
      bool check = check_park();
      park(check);
    }

    if (value == "forward"){
      motor_control(true, 5);
    }

     if (value == "backward"){
      motor_control(false, 5);
    }
     if(value == "stop"){
      motor_control(true, -20);
     }
  }
}
```

Figure 58 Main Loop

Obstacles

1. The UK1122 model wouldn't work on the DC motor responsible for steering right and left, while the other one responsible for the forward and backward drive was working perfectly. To find out what the problem was, we tried different combinations of inputs for IN3 and IN4 and measured the voltages across the output terminals. This led to the deduction that one of the transistors (S3 or S2) of the second H-bride is damaged, leading the elimination of our chance to use the IN3 =1 & IN4 = 0 case, rending us not being able to steer left.

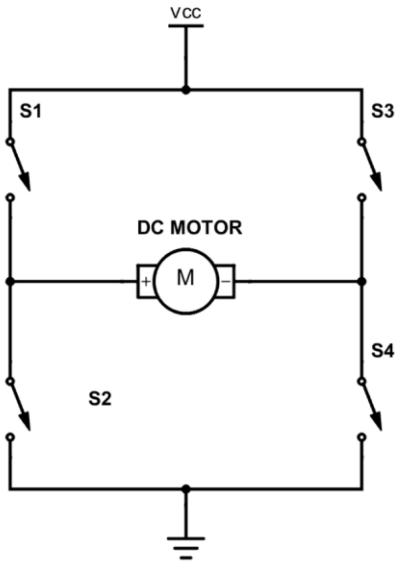

Figure 59 H-bridge module

2. The difference in battery quality between different brands, led to some confusion when it comes to the functioning of the motors. Since some of the motors need a higher current than others, we needed to figure out what type of batteries shall be used for each.

3. Two ultrasonic sensors where available and we couldn't get anymore due to time constraints in time we need 5 ultrasonic sensors, one on each corner of the car, and one on the front. For that, placed 2 ultrasonics on 2 stepper motors, one in front and one on the back of the car, which will turn the sensor to the direction where It needs to measure by moving a certain number of steps clockwise or anticlockwise.