German University in Cairo

Mechatronics Engineering (MCTR601)

“Automated Spacing Self-Parking Car”

Name	ID	Lab Number
Omar Ashraf Fetouh	46-20497	T-66
Omar Tarek Abofreikha	46-2875	T-54
Omar Ehab Maher	46-20932	T-66

Table of Contents

1. Project Description

Our project's idea is working on reducing car accidents due to bad parking skills of the drivers.
Thus***, our objective*** is to offer a solution for one of the most challenging driving tasks, which is parallel parking by automating the action. The automatic parking system aims to enhance the comfort and safety of driving in constrained environments where much attention and experience is required to steer the car. Moreover, securing the car from any sudden movement made by other cars parking around.

Our project mainly is divided into 2 applications each in a different phase, vividly illustrated in our prototype:

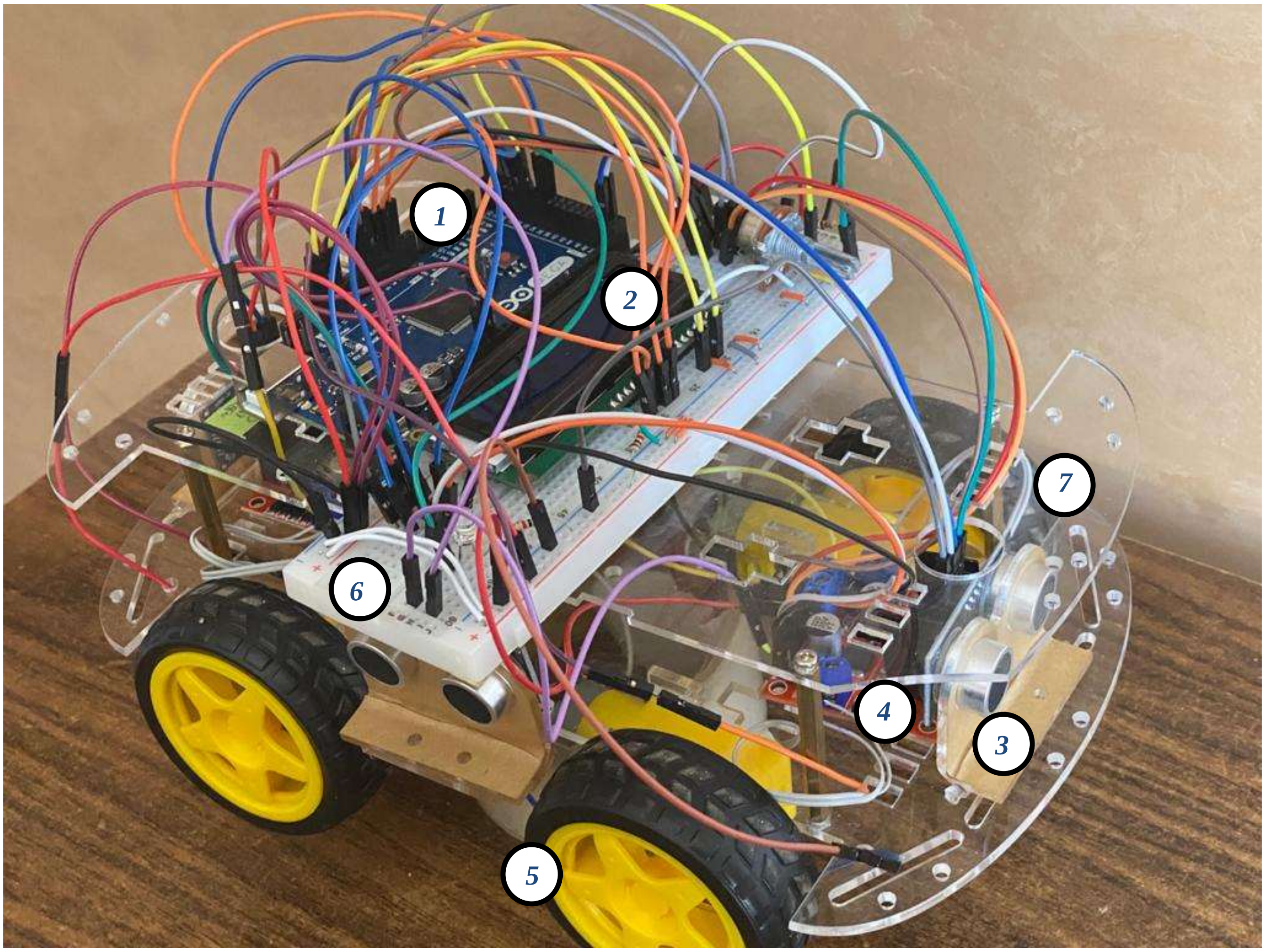
Phase 1 (Parking algorithm):

We have built a prototype for implementing a parallel parking algorithm on a mobile robot car, using an Arduino Mega, ultrasonic sensors and an LCD. A hardware push button starts process of self-parking the car. On pressing the button, the robot moves forward, while scanning for vacant spots on the side using ultrasonic sensor array. On finding a suitable spot of appropriate dimensions, the robot stops at an appropriate distance. The robot then sends signals to the actuators (4 DC motors) to make a 45 degree turn and back up into the spot. Then make another 45 degree in the opposite direction to become straight.

Phase 2 (Car recentering algorithm):

In our prototype using the car built, an Arduino Mega, ultrasonic sensors and an LCD, after the car has successfully parked, we applied a closed loop feedback control with a Proportional gain to recenter the car in its parking slot between its neighboring parking cars autonomously whenever one of them has moved from its position to keep the safest distance between both cars. This was possible with 2 ultrasonic sensors in the front and in the rear to read the distances from the neighboring cars, get the readings average and set this value as the target where the car tries to reach by adjusting the actuators (4 DC motors) speed to reach the required destination.

Functional Diagram of the Robot



Part number	Name
1	Arduino Mega
2	LCD Display
3	Ultrasonic Sensor
4	H-Bridge
5	DC Motor with Wheel
6	Breadboard
7	Acrylic Chassis

Main Components

- **Microcontroller**

  Used to drive the dc motors and sync it with the readings of the ultrasonic sensors to recognize the parking slots, help parking and taking safety measures when coming close from a vehicle.

- **4x Dc Motor Robot Kit**

  Used as wheels of the self-parking car; it is programmed with the microcontroller for every wheel to move on its own and have the bi-directional movement option.

- **3x HC-SR04 Ultrasonic Sensor**

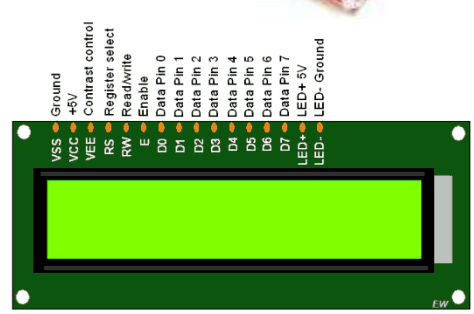
  Used to detect the surroundings of the car in order to keep it safe from other vehicles in the parking area and help the car find suitable parking slots.

- **L298N H-bridge**

  It's a high current motor drive shield. Arduino's maximum DC current from VCC and GND pins is merely 200 mA. This shield provides up to 2 A current to drive the car's motors.

- **16X2 LCD Display Green**

  Used to display the state the car is in. making it easier for us to recognize the action it is doing at the moment.



2. Methodology

2.1 Mechanical design

System's Mechanical Components:

1. Acrylic Chassis

A transparent double layer chassis made from acrylic to create dynamic handling of the components mounted on it. It's chosen to be with appropriate size and plenty of holes to simplify the process of mounting different components on this body.

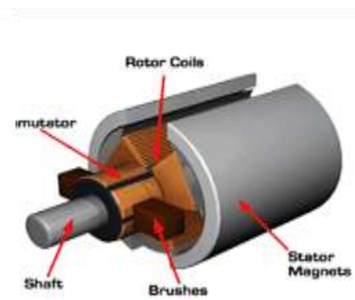
2. Bolts and Nuts

Used in building 4 primary pillars in the acrylic chassis to hold its 2 layers together for more stability during motion of the car.

System's Electromechanical Components:

1. 4x Dc Motor Robot Kit with their motor drivers

Are the main component of the car since it's used to move it in different directions. Thus, it's possible to apply the parking and recentering algorithms by set of instructions that execute the motion desired by the prototype.



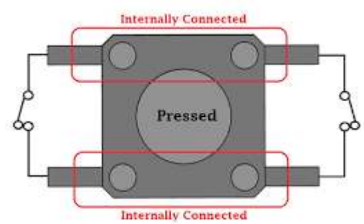
Working Mechanism:
DC motor is constructed of 2 main parts:

- Rotating part called Armature
- Stationary part called Stator

The basic working principle is that whenever a current carrying conductor places in the magnetic field, it experiences a mechanical force. The magnets that generated the magnetic field within the motor are the Stators, while the coil that has current passing through it experiencing mechanical forces is the Armature which is connected to the shaft of the motor generating the given torque of the wheels.

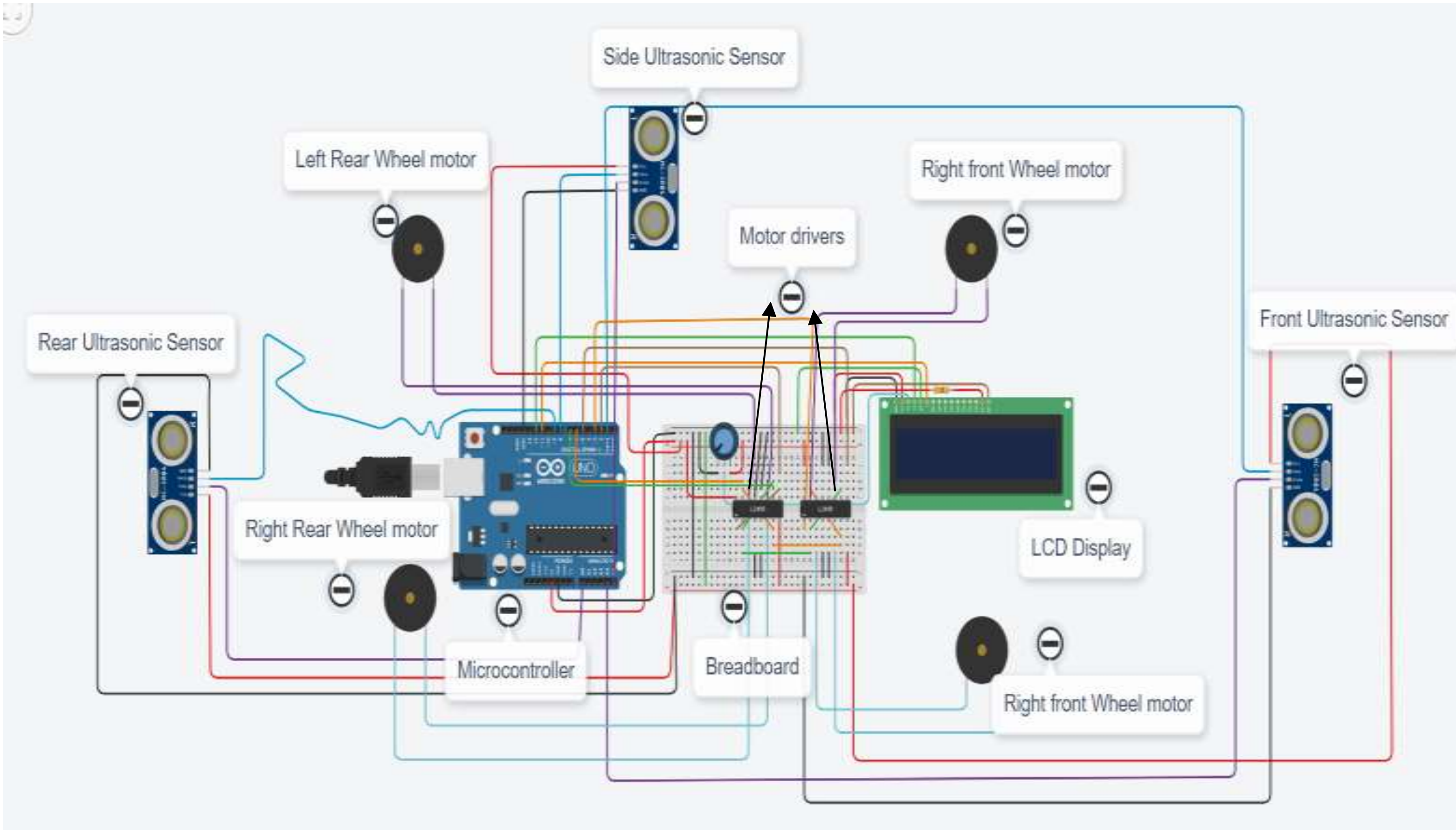
2. Push Button (Momentary Contact)

Used as an electromechanical switch to give a starting signal for the system to start operating and doing the desired function.



Working Mechanism:
A push button switch is a small, sealed mechanism that completes an electric circuit when you press on it. When it's on, a small metal spring inside makes contact with two wires, allowing electricity to flow. When it's off, the spring retracts, contact is interrupted, and current won't flow. The button is normally off since we are applying pull down mechanism. The body of the switch is made of non-conducting plastic.

2.2 Electrical design



Microcontroller:

In our prototype we used Arduino Mega since it has a sufficient number of pins that would be appropriate to complete our connections. However, the schematic shown above was formulated on TINKERCAD software that didn't have the Arduino Mega component, so UNO was used for simplicity.

We programmed the microcontroller using the Arduino IDE to perform the required algorithms by the system including:

- Parking Algorithm
- Recentering Algorithm

Wheel Motors and their drivers (Actuators):

4x DC motors were used to control the car each 2 connected to a motor driver (L298N) to control the motion direction of each wheel on its own through the IN1, IN2, IN3 and IN4 signals and control the speed of each wheel using ENA and ENB PWM signals.

The signals of the H-Bridge (L298N) where obtained from the microcontroller (Arduino Mega). And power was supplied to it through a 12V adapter to move the wheels and a 5V signal from the microcontroller to power up the H-bridge main circuit.

Ultrasonic Sensors:

Side Ultrasonic Sensor:

Used to search for an empty parking slot to initiate the parking algorithm whenever a parking slot is found.

Front and Read Ultrasonic Sensor:

Used as the feedback sensors for our proportional controller as their readings gives us the desired position and the actual position as well.

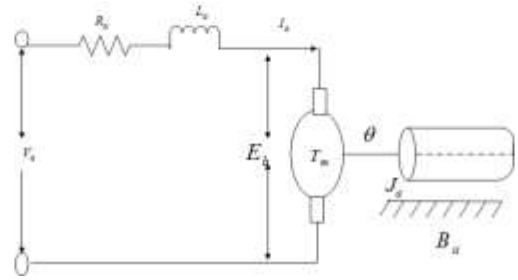
LCD Display:

Used to display specific message at each stage of the code to make it easier for us to detect any possible errors and to make it clear of which operation is the car already performing.

Thus, it is considered as an assisting device for the developers to track the code and the users to understand what’s happening.

2.3 Control

2.3.1. Modeling



Equation of the current passing in the DC motor assuming inductance is neglectable is

I=Vin÷(Rm)

Coupling equation between electric circuit and mechanical is

Tm = Kt * I

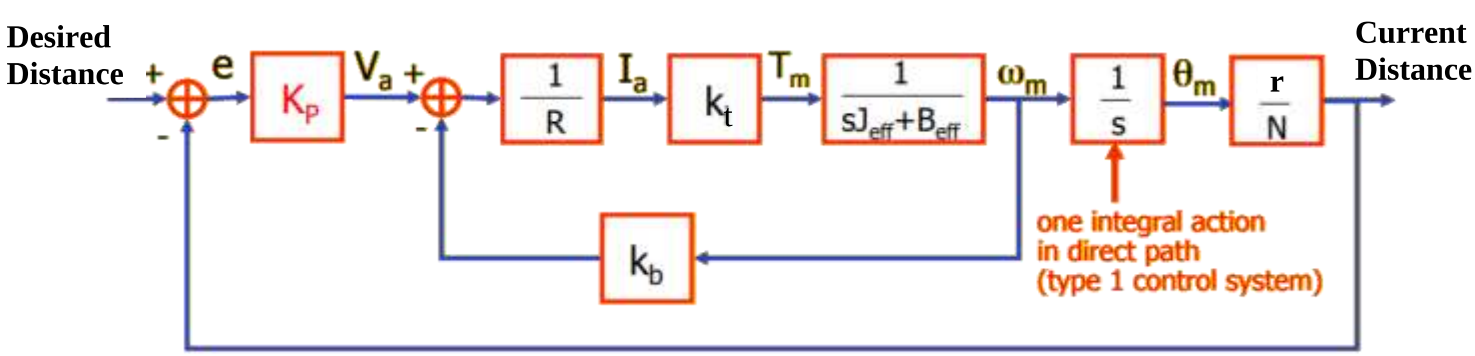
Equation of angular velocity produced by given torque in S domain is

ω = Tm ÷ (Js + B)

Equation of distance covered by a given angular velocity is

X = Integration (ω) * N * r

Symbols and their values in our systems if constant Type equation here.**:**
I: Current supplied to the motor
Vin: Voltage supplied to the motor circuit
Rm: Resistance of motor circuit
Tm: Torque produced by the motor
Kt: Motor torque constant
ω: Angular velocity of wheel
J: System moment of inertia of the rotor
B: Motor viscous friction
N: Reduction ratio of gear set
Kb: back electromotive force (EMF) assuming its neglectable
r: Radius of the wheel
Kp: Proportional Gain
Block Diagram:



By using the equations illustrated in the previous page we can derive the transfer function of Voltage as input and current distance as output by substituting with the equations in each other in S domain:

The open loop transfer function is:

$$\frac{CurrentDistance}{Vin} = \frac{Kt}{(J \times R \times N)S^2 + (B \times R \times N)S} = G(H)$$

The closed loop transfer function is:

$$\frac{CurrentDistance}{Vin} = \frac{1}{1 + \frac{Kt}{(J \times R \times N)S^2 + (B \times R \times N)S}}$$

The Desired Distance is acquired from Vin as

$$Vin = DesiredDistance * error * Kp$$

2.3.2. Analysis

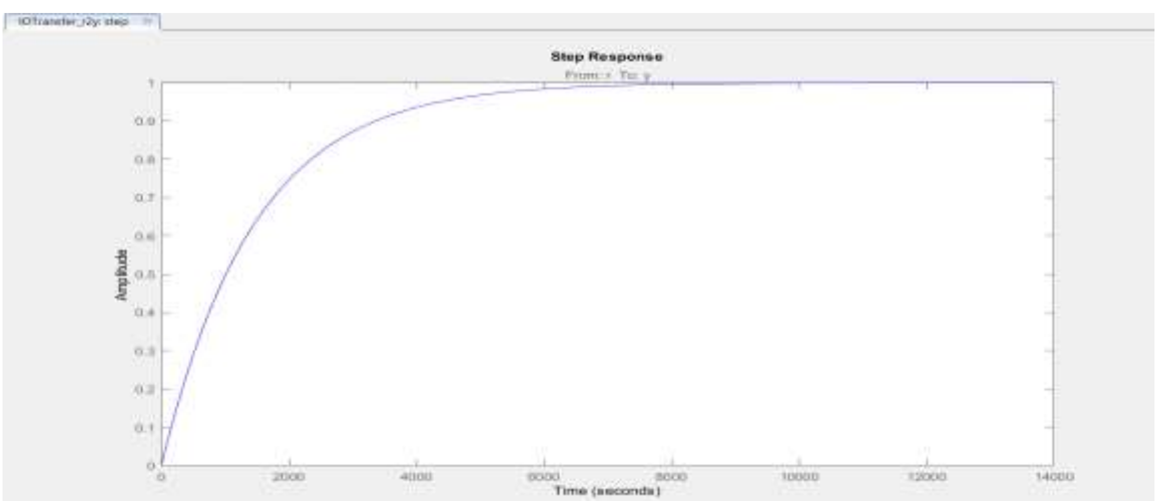
Initially, MATLAB’s sisotool shows the open-loop response of the system given its open loop transfer function which doesn’t use a feedback signal; thus, only triggers the system to move towards the desired location or distance and doesn’t take into account any external disturbances or errors that might have led to the system not reaching its desired value.

MATLAB Code:

```
Kt = 13.8;
R = 1.6231;
N = 30;
Jeff = 0.051;
Beff = 12.4;
WheelRadius = 0.03;
G = tf([Kt*WheelRadius],[(Jeff*R*N) (Beff*R*N) 0]);
sisotool(G)
```

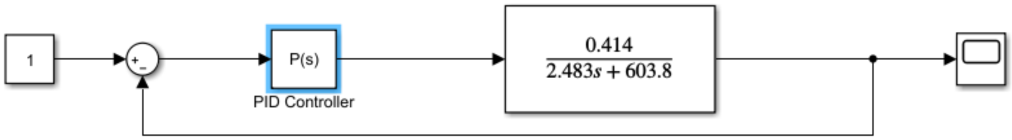
```
Command Window
>> Analysis

G =

        0.414
  -------------------
  2.483 s^2 + 603.8 s

Continuous-time transfer function.

fx >>
```

Sisotool’s Open Loop Step Response:



2.3.3. Controller Design

Drawing the system’s block diagram using Simulink:

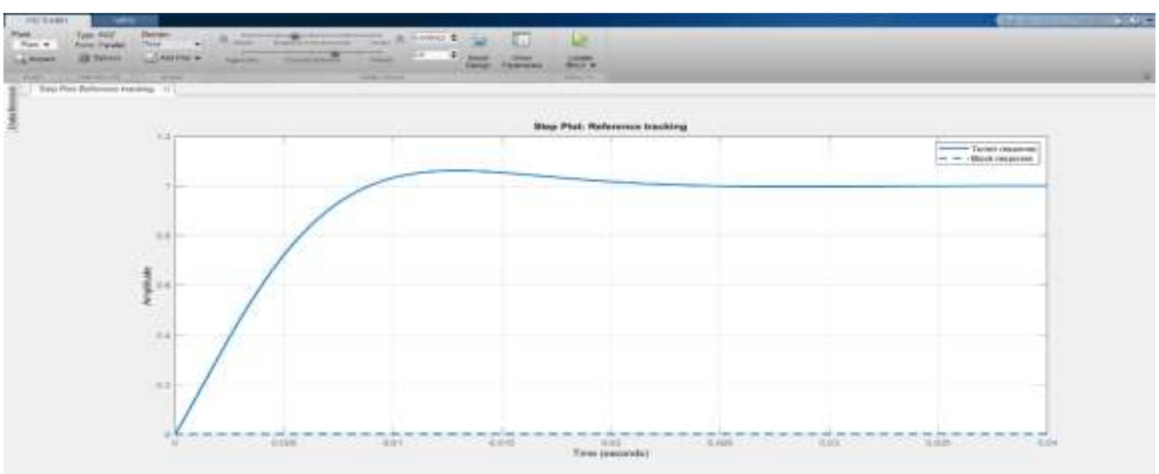


After using PID Tuning the control law is clearly shown:

```
>> c2d(C,0.05)

ans =

                Ts
  Kp + Ki * ------
                z-1

  with Kp = 873, Ki = 4.73e+05, Ts = 0.05

Sample time: 0.05 seconds
Discrete-time PI controller in parallel form.
```

To discretize the controller (s), we may use the MATLAB **c2d** command. It takes the transfer function $C(s)$ and the discretization period T, then generates the discrete frequency domain transfer function $C(z)$, with some mathematical manipulations and by using the inverse z-transform identities, the control input signal (using control law) must be then fed using PWM to motors enables but since in the programming code we are already using a **Millis()** function that enables calling PID method after constant time interval; also, performing multiple tasks along with this task we didn’t need to feed this control input signal to the microcontroller or our system was simple enough that it didn’t require doing so.

The following graph shows the system response after performing tuning on the closed loop transfer function using MATLAB:



2.4 Programming

```
#include <LiquidCrystal.h>       //LCD Library
#include <PID_v1.h>              //PID Library

LiquidCrystal LCD (12,10,5,4,3,2);
const int trigPin = 13;
const int echoPin = 11;
const int trigFrontPin = 52;
const int echoFrontPin = 53;
const int trigBackPin = 36;
const int echoBackPin = 37;
const int LEN = 6;          //Enable Pin for the vehicle left side
const int REN = 7;          //Enable Pin for the vehicle right side
const int LIN1 = 22;
const int LIN2 = 24;
const int RIN1 = 23;
const int RIN2 = 25;
const int LEDPin = 30;
const int buttonPin = 32;
const float enoughTime = 550;            //(Constant Car Speed*EnoughTime) = Appropriate distance for parking
const unsigned long ultraInterval = 10;  //Period of reading distances from the right sensor
const unsigned long ultrasInterval = 2;  //Period of reading distances from the front&Rear sensors
const unsigned long PIDInterval = 10;    //Period of calling the PID Method that controls the system stability
unsigned long prevUltraTime = 0;
unsigned long prevFrontTime = 0;
unsigned long prevRearTime = 0;
unsigned long prevPIDTime = 0;
int motorsSpeed = 55;                    //Car Searching Speed
float speedOfSound = 343;
bool foundParkingSlot = false;
bool finishedParking = false;
bool flag = true;
bool enter = false;
bool Parking = false;
bool switching = false;
bool exitParking = false;
float duration, durationFront, durationBack;
float distance, distanceFront, distanceBack;
float elapsedTime;
double kp = 255.0/100.0, ki = 0, kd = 0;     //PID Gains
double setPoint, Input, Output;              //PID Parameters
PID myPID(&Input, &Output, &setPoint, kp, ki, kd, DIRECT);
```

```
void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(trigFrontPin, OUTPUT);
  pinMode(trigBackPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(echoFrontPin, INPUT);
  pinMode(echoBackPin, INPUT);
  pinMode(LIN1, OUTPUT);
  pinMode(LIN2, OUTPUT);
  pinMode(RIN1, OUTPUT);
  pinMode(RIN2, OUTPUT);
  pinMode(LEN, OUTPUT);
  pinMode(REN, OUTPUT);
  pinMode(LEDPin,OUTPUT);
  pinMode(buttonPin,INPUT);
  LCD.begin(16,2);                  //Initializing LCD Screen
  myPID.SetMode(AUTOMATIC);         //Enabling PID
  myPID.SetSampleTime(10);          //Determines how often the PID algorithm evaluates.
  myPID.SetOutputLimits(-255,255); //PID's Output Range which reflects a PWM Signal so shouldn't be <0 or >255
}

void loop() {
  unsigned long currentTime = millis();    //Capture Current time from millis function.
  digitalWrite(trigPin, LOW);
  digitalWrite(trigFrontPin, LOW);
  digitalWrite(trigBackPin, LOW);
  int buttonStatus = digitalRead(buttonPin);
  if(buttonStatus == HIGH)
  {
    enter = true;
  }
  if(enter == true){
  if((currentTime - prevUltraTime >= ultraInterval) && (Parking == false) && (finishedParking == false)){
   ultrasonicAndLCD();           //Measuring Distances from neighbouring Cars
   prevUltraTime = currentTime;
  }
  if(foundParkingSlot == false)
  {
    Searching();            //Move with constant speed and searching for a parallel parking slot
  }
  if(foundParkingSlot == true && (Parking==false))
  {
    Parking = true;
    Park();                 //Performing parking algorithm executed when a parking slot is found
  }
  if((((currentTime - prevFrontTime >= ultrasInterval) && (finishedParking == true) &&(switching == false)) |
   ((currentTime - prevRearTime >= ultrasInterval)  && (finishedParking == true)
    &&(switching == true))) && (exitParking == false))
  {
```

```
    if(switching == false){
    Front();              //Measuring distance from front car
    prevFrontTime = currentTime;
    switching = true;     //Enabling the switch between front & Rear Measurments
    }
    else{
     Rear();              //Measuring distance from rear car
     prevRearTime = currentTime;
     switching = false;
    }
   }
   if((currentTime - prevPIDTime >= PIDInterval) && (finishedParking == true) && (exitParking == false))
   {
    PIDControl();        //Calls PID Method for controlling car's position when parking is done
    prevPIDTime = currentTime;
   }
  }
}
void PIDControl(){
  if(distanceFront<5 && distanceBack<5){
    ExitParking();
  }
  else{
  setPoint = (distanceFront + distanceBack)/2.0 ; //Target
  Input = distanceBack;  //ultrasonic readings from behind
  myPID.Compute();
  if (setPoint > distanceBack){           // positive error
      FORWARD();
  }
  else if (setPoint < distanceBack){      // negative error
       Output = Output * (-1) ;
       BACKWARD();
  }
  long x = map(Output,0,255,50,255);
  if(x == 50){
    x = 0;
  }
void Searching(){
  analogWrite(LEN,motorsSpeed);
  analogWrite(REN,motorsSpeed);
  FORWARD();
  if(millis()-elapsedTime >= enoughTime && (flag == false))
  {
    Serial.print("Difference: ");
    Serial.println(millis()-elapsedTime);
    foundParkingSlot = true;
    analogWrite(LEN,0);
    analogWrite(REN,0);
    digitalWrite(LEDPin,HIGH);
  }
}
```

```
void ultrasonicAndLCD (){
  digitalWrite(trigPin, HIGH);
  duration = pulseIn(echoPin, HIGH);
  duration = duration/1000000.0;
  distance = (speedOfSound*duration) / 2;
  LCD.setCursor(0,0);
  LCD.print("                ");
  LCD.setCursor(0,0);
  LCD.print("Distance:");
  distance = distance*100.0;
  LCD.setCursor(0,1);
  LCD.print("                ");
  LCD.setCursor(0,1);
  LCD.print(distance);
  LCD.print(" cm");
    if(distance < 25 && distance>0)
  {
    elapsedTime = 0;
    flag = true;
  }
  if(distance >= 30 && (flag == true))
  {
    elapsedTime = millis();
    flag = false;
  }
}
void Front(){
    digitalWrite(trigFrontPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigFrontPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigFrontPin, LOW);
    durationFront = pulseIn(echoFrontPin, HIGH);
    durationFront = durationFront/1000000.0;
    distanceFront = (speedOfSound*durationFront) / 2.0;
    distanceFront = distanceFront*100.0;
    LCD.setCursor(0,0);
    LCD.print("                ");
    LCD.setCursor(0,0);
    LCD.print("Front: ");
    LCD.print(distanceFront);
    LCD.print(" cm");
    delayMicroseconds(100);
}
void Rear(){
    digitalWrite(trigBackPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigBackPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigBackPin, LOW);
    durationBack = pulseIn(echoBackPin, HIGH);
    durationBack = durationBack/1000000.0;
    distanceBack = (speedOfSound*durationBack) / 2.0;
    distanceBack = distanceBack*100.0;

```

```
    LCD.setCursor(0,1);
    LCD.print("                ");
    LCD.setCursor(0,1);
    LCD.print("Rear: ");
    LCD.print(distanceBack);
    LCD.print(" cm");
    delayMicroseconds(100);
}

void Park(){
  LCD.setCursor(0,0);
  LCD.print("                ");
  LCD.setCursor(0,1);
  LCD.print("                ");
  LCD.setCursor(0,0);
  LCD.print("PARKING ....");
  delay(1000);
  analogWrite (REN , 175) ;
  analogWrite (LEN , 175) ;

  digitalWrite (LIN1 , LOW) ;
  digitalWrite (LIN2 , HIGH) ;

  digitalWrite (RIN1 , HIGH) ;
  digitalWrite (RIN2 , LOW) ;

  delay (300);  //300
  STOP();

  analogWrite (REN , 75) ;
  analogWrite (LEN , 75) ;

  digitalWrite (LIN1 , LOW) ;
  digitalWrite (LIN2 , HIGH) ;

  digitalWrite (RIN1 , LOW) ;
  digitalWrite (RIN2 , HIGH) ;

  delay (550) ; //550

  STOP();

  analogWrite (REN , 175) ;
  analogWrite (LEN , 175) ;
```

```
  delay (300) ;

  STOP();
  LCD.setCursor(0,0);
  LCD.print("                ");
  LCD.setCursor(0,1);
  LCD.print("                ");
  LCD.setCursor(0,0);
  LCD.print("PARKED ....");
  delay(1000);
  finishedParking = true;
}
void ExitParking(){     //Method Called when front&Rear Cars are too close !!
  exitParking = true;
  LCD.setCursor(0,0);
  LCD.print("                ");
  LCD.setCursor(0,1);
  LCD.print("                ");
  LCD.setCursor(0,0);
  LCD.print("Tab Slamo 3leko");

  analogWrite (REN , 175);
  analogWrite (LEN , 175);

  digitalWrite (LIN1 , LOW);
  digitalWrite (LIN2 , HIGH);
  digitalWrite (RIN1 , HIGH);
  digitalWrite (RIN2 , LOW);
  delay (250);

  digitalWrite (LIN1 , LOW) ;
  digitalWrite (LIN2 , LOW) ;
  digitalWrite (RIN1 , LOW) ;
  digitalWrite (RIN2 , LOW) ;
  delay(100);

  FORWARD();
  delay(400);

  digitalWrite (LIN1 , LOW) ;
  digitalWrite (LIN2 , LOW) ;
  digitalWrite (RIN1 , LOW) ;
  digitalWrite (RIN2 , LOW) ;
  delay(100);
```

```
void FORWARD(){     //Setting DC Motors' Direction to Forward
  digitalWrite (LIN1 , HIGH);
  digitalWrite (LIN2 , LOW);
  digitalWrite (RIN1 , HIGH);
  digitalWrite (RIN2 , LOW);
}

void BACKWARD(){     //Setting DC Motors' Direction to Backward
  digitalWrite (LIN1 , LOW);
  digitalWrite (LIN2 , HIGH);
  digitalWrite (RIN1 , LOW);
  digitalWrite (RIN2 , HIGH);
}

void STOP(){        //Stopping all DC Motors
  digitalWrite (LIN1 , LOW) ;
  digitalWrite (LIN2 , LOW) ;
  digitalWrite (RIN1 , LOW) ;
  digitalWrite (RIN2 , LOW) ;
  delay(300);
}
```

Software Flowchart

Link of the flow chart:
https://lucid.app/lucidchart/invitations/accept/inv_b42618a2-af91-4b90-b32e-44df039a8956?viewport_loc=-553%2C1121%2C3840%2C1674%2C0_0

3. Design Evaluation

Hardware Evaluation:

The Designed car was made using only P-Controller not PI as resulted from the software controlling design and this controller was determined using trial & error approach.
In conclusion, the car approaches the average distance between the front & rear cars in a speed relative how far it's from this location that's totally safe, but in the actual hardware design the car did so small oscillations before reaching the point of stability; that's reasonable because of the moment gained from this increased speed signal generated using PWM on the motors ENABLE.

Software Response Evaluation:

We started with the values of the gains obtained from the simulation in the controller design section as an initial guess for the gains in PID controller in our car. Then, experimentally through trial and error approach we obtained new value for gains that lead to a more stable response.

Reason for different gain values from the control design section vs. from practical experimentation:

This is because of many assumptions done in the modeling section to simplify the calculation process and make the process of calculating the transfer function easier. Also, not all parameters are calculated accurately; thus, since the modeling section doesn't reflect the real system accurately, so it's normal that gains obtained from the simulation doesn't mean that it should be taken for granted; hence, the trial & error approach was needed in attempt of finding the perfect gain values that stabilize the system and trying to reach a zero error in the shortest rise time possible with a minimal overshoot percentage.