Design & Development of an Autonomous Automotive Parking System

Thesis submitted for the partial fulfillment of Bachelor Degree in Mechatronics Engineering

By

Abdelrahman Abd-Elmoez Ahmed	(20190019)
Abdelrahman Emad Ahmed	(20190233)
Ayman Hamdy Salem	(20190169)
Daniel George Alfonse	(20190013)
Kareem Abdelkader Mohamed	(20190128)
Omar Ehab El-Sayed	(20190021)
Peter Sameh Henri	(20200278)
Shehab Gomaa Mohamed	(20190171)
Youssef Yacoub Saad	(20190085)

Under the supervision of

Dr. Ahmed Abdelaziz Mohamed

Pyramids higher institute for engineering and technology

2024

Preface

The concept of self-parking vehicles has for long captured the imagination of innovators and consumers alike. This project presents a significant stride in making this technology a tangible reality. It encapsulates the culmination of extensive research and development in the field of autonomous driving, with a particular focus on the self-parking system. Our endeavor is to dissect and reconstruct the complex orchestration of sensors, algorithms, and actuators that enable a vehicle to execute precise parking maneuvers without human interference.

The significance of this project extends beyond academic achievement; it is an indication of the transformative potential of autonomous systems in everyday life. The Self-parking Car System exemplifies the convergence of safety, efficiency, and innovation, aiming to alleviate the challenges of urban parking. We are immensely grateful for the mentorship and resources provided by our academic institution, the valuable advice from industry experts, and the unwavering support from our peers. This project is not merely a testament to our educational journey but also a contribution to the evolving landscape of autonomous vehicle technology.

The Department of Mechatronics is an interdisciplinary field that combines mechanical, electronic engineering, and computer science. It would be a valuable resource for anyone wants to develop a self-parking project. Offering crucial resources, the department provides expertise, tools, and practical skills necessary for designing and building these systems. It also fosters essential soft skills like project management and problem-solving. Innovators seeking to develop self-parking systems are encouraged to tap into these resources by contacting the department.

Contents

List of Figures

List of Tables

Abstract

This project focuses on the development a single task of the automotive autonomous driving tasks, which is ***Automatic Parking*** throughout the integration of sensors and actuators controlled by a microcontroller together with trajectory planning and coding of the proper parking path. The phases of this project will be presented in the remainder of this report, which goes from conceptual design to critical design and finally to the implementation and testing phase of our prototype. Car is an example of a non-holonomic system in which the number of control commands available is less than the number of coordinates representing its location and direction; According to non-holonomic constraints, a shortest path algorithm for the self-parking problem under specific initial conditions has been proposed and proven. Appropriate project components have been selected to increase the efficiency and accuracy and would be explained. The platform (car model) has utilized the same nondimensional values of the real car, but at a smaller size. The stages of developing and writing the required control codes to implement the self-parking algorithms will be discussed.

Self-parking is a difficult task that requires the car to perceive its surroundings, make decisions, and control its actuators precisely. In this project, a self-parking system is proposed that uses an array of sensors, such as ultrasonic sensor, to detect the car's surroundings. The system also includes a software unit that processes sensor data and generates the proper steering control signals to drive the car on the parking path. A servo motor was used for the steering system, and two DC motors for the propulsion system. The microcontroller uses several algorithms that control the movement of the robot car to achieve accurate and reliable parking maneuvers according to the available parking area.

Chapter 1

Introduction

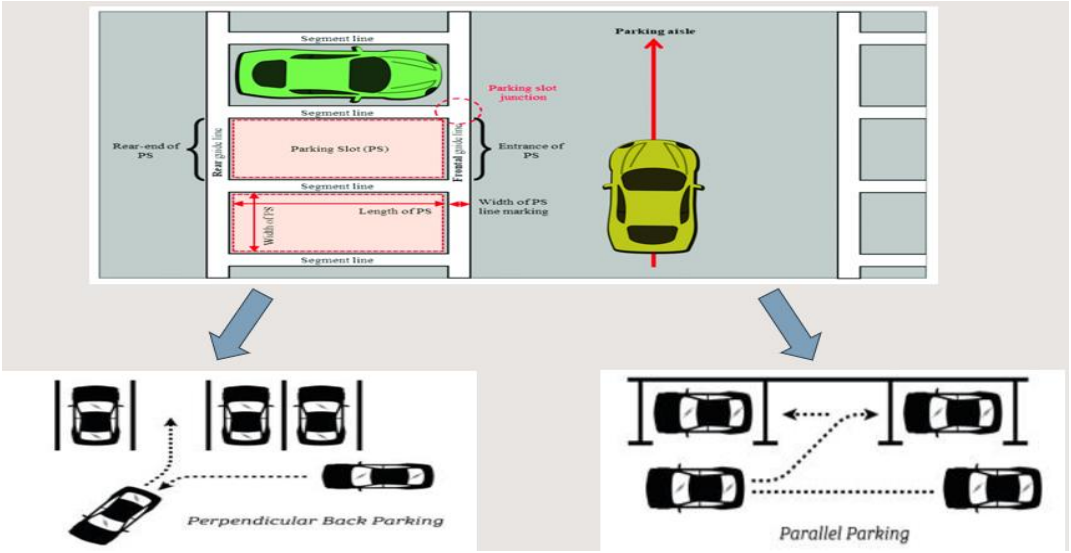
1.1 History and Motivation

Advanced driver-assistance systems (ADAS) are electronic systems in a vehicle that use advanced technologies to assist the driver. They can include many active safety features, such as self-parking, driver alertness, and adaptive cruise control. For decades, parallel parking and navigating tight spaces have been the bane of many a driver's existence. This report delves into the world of self-parking cars. The problem we are working to solve is parking, and we are using embedded systems technologies to help drivers find parking spots and maneuver into them. These systems aim to improve parking efficiency, reduce driver fatigue, and reduce the risk of accidents.

1.2 Goals and Objectives

Automatic parking is an autonomous car-maneuvering system that moves a vehicle from a traffic lane into a parking spot to perform parallel, perpendicular, or angle parking. The goal of the automated parking system is to park in narrow spaces, reduce collision accidents while parking, and increase safety and well-being for drivers.

1.3 Short Description of the Project



To implement perpendicular or parallel self-parking; 4 ultrasonic sensors were used, two on the right side to search for empty parking spaces, one for the front of the car and the other at the end to detect any obstacles to avoid. Two DC motors were used for the rear wheel drive and moving the car, and a servo motor was used for the steering system. It will all be under the control of the microcontroller.

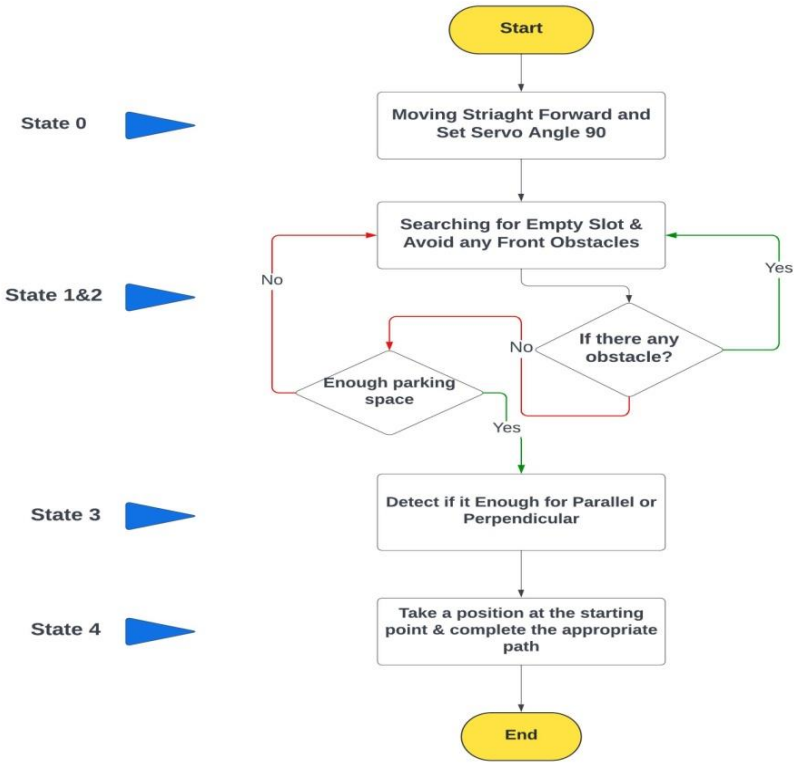


Fig 1.1 Operation Flow Chart

1.4 Development

Research began on the importance of self-parking systems for cars, and with the development of technology and the use of electronics in all fields, including the field of cars, care was taken to seek to implement these systems in cars. We researched how this project works and what are the correct sequences and algorithms, and reviewed projects implemented in the field of autonomous parking to learn and gain experience.

Then the second stage began by searching for the appropriate algorithm to implement the project objectives accurately and efficiently by reviewing references and scientific research that dealt with the same topic, then comparing the results with the expected results of the project and the extent of the possibility of implementing them. Through simulation programs such as MATLAB. After considering the problems, one type of collision-free shortest path has been proposed in the case of approaching the vehicle to the target by following a fixed path while avoiding obstacles. The iterative path planning algorithm is introduced for parking in a narrow space, which is not large enough to run the parking process simultaneously.

Then we began creating a car design based on the dimensions and design of real cars to achieve realism for our program, and a car design from BMW was chosen.

In parallel with the design, appropriate components for the project were selected, such as the operations center, sensors, communication buses, and actuators. The selection was made based on criteria for each component that fit our project and from other projects that were like ours.

1.5 Financial Cost

Components	Quantity	Price
Car Design	-	1100
Bolts, Nuts, and Bushings	-	100
Blue Pill Board	1	450
Ultrasonic Sensor	4	170
Ultrasonic Holder	4	60
DC Motor	2	760
Servo Motor	1	200
L298N Motor Driver	1	80
Lithium Battery	6	360
Battery Holder	2	80
Step Down	2	240
BMS	1	160
Buzzer	1	20
Boat Rocker Switch	2	10
Mini Bread Board	1	25
Wires	-	60
----------	**----------**	**----------**
Total		3875

Chapter 2

Literature Review and Theoretical

2.1 Overview

This section provides a short review of some related systems and introduces a theoretical background of the project. Based on our research, we came across very limited products that are related to the current project.

2.2 Literature Review of Previous Projects

The smart parking system implemented mainly in the Europe, United States and Japan is developed with the incorporation of advanced technologies and researches from various academic disciplines. Now- days, there is rapid growth in the parking system. Manpower is needed for each car parking slot to select a parking slot manually and give direction to drive properly into slot. So, there is a need to develop an automatic parking system which will reduce manual work as well as being useful for careful parking of cars and other vehicles. Parking system routinely experience parking related challenges, especially in the urban and metropolitan areas. While doing a survey we have found that this automatic car parking system has been proposed by various researchers using different technology. In some papers, some researchers have proposed this system using Around View Monitor (AVM). In their paper they have discussed fusion of AVM and ultrasonic sensor, used to detect the vacant parking slot in the automatic car parking system. The AVM provides a virtually 360-degree scene of the car in bird „s eye view. The AVM helps the driver to maneuver into parking spots. Through the bird „s eye view, a driver can check for obstacles around the vehicle. First, the parking slot marking detected in the AVM image sequence. A tree structure-based method detects parking slot marking using individual AVM image sequence and image registration technique. Second, empty slots are detected using ultrasonic sensors. The probability of parking slot occupancy is calculated utilizing ultrasonic sensor data acquired while the vehicle is passing by parking slots, and finally the selected empty slot is tracked, and the vehicle is

properly parked in selected parking slots. Some other researchers have discussed this system using another technology i.e. GSM Technology. The functionality of the technology is that the user sends a message to the GSM modem which is placed at the parking end. The GSM modem will send a conformation message to the user whether the slot is vacant or not. If it is vacant then the user must message the exact time and duration, he/she wants to park the vehicle in the parking slot. Then the GSM modem will send a password and the parking lot number to access the reserved parking lot. Once the conformation message has been sent, the counter for the reservation time will automatically start for sending message. Another paper attempts to discuss this system using FPGA Technology. In their paper they have discuss how to implement an automatic car parking system using FPGA technology, where the access in the parking which is made by barrier, if there are vacancies with the lifting of the barrier a ticket is issued with a client code and there starts a timer for measuring the time left in the parking. The analog signals transferred through a digital analog converter as input signals in the FPGA. To work with FPGA Xilinx software must be used. Another paper discusses a system using some digital key along with some robotic technique. When a car enters the entry of the automated car parking system, an IR detection subsystem detects the presence. Then the driver is promoted to enter a valid key and to choose the option of either parking or retrieving the car. Each key is checked for accuracy and assigned a designated parking slot. Upon entering the correct key, the car is picked up along with the pallet from the stack system and placed in the designated spot. When drivers return to pick up the car, he enters the valid key for which the system will check in its database and the car is returned to the driveway. The stack system will pull down the pallets to make room for incoming pallets. The system includes a robotic lift with motors for picking up the car and placing it in the designated spots. Another paper discusses a system where microcontroller 89S51 has been used, in their paper they have discussed a system which is automated with the user being given a unique ID corresponding to the trolley being allocated to him/her. The idea is to park and move cars with no disturbance to the already parked cars in their system. some other researchers have discussed this system using RFID. According to their system, the vehicle owner must first register the vehicle with the parking owner and get the RFID tag. When the car must be parked, the RFID tag is placed near the RFID reader, which is installed near the entry gate of the parking lot. As soon as the RFID tag is read by the reader, the system automatically deducts the specified amount from the RFID tag and the entry gate boomer opens to allow the car inside the parking area. At the same

time, the parking counter increments by one. Similarly, the door is opened at the exit gate and the parking counter is decremented. There are many kinds of methods that have been proposed by many researchers. Those methods whether use single intelligent or combinations of two intelligent. Xiaochuan Wang and Simon X. Yang have developed neuro-fuzzy control system for obstacle avoidance of a nonholonomic mobile robot. They combined four infrared sensors for distance to obstacle detection around the mobile robot. The distance information is processed by the proposed neuro-fuzzy controller to adjust the velocities of two separate driven wheels of the robot. They constructed eighteen fuzzy rules for obstacle avoidance. Based on their research it showed that the development of two algorithms is more effective than using a single algorithm. Besides that, Gustavo Pessin, Fernando Osório, Alberto Y. Hata and Denis F. Wolf give an idea to develop multi-robot system by using combination of two algorithms (Genetic Algorithm and Artificial Neural Network). In this project, Genetic Algorithm is being used to plan and evolve positioning strategy for mobile robot performance. They developed a mobile robot with distance sensor that is controlled by multilayer perception (MLP) in ANN. ANN was trained and to control the robot's actuators. It allows mobile robot to move in dynamic environments with obstacle avoidance. Based on their results, it shows that the ANN satisfactorily controls the mobile robot.

Regarding those researchers, it can summarize that the uses of Fuzzy Logic, Artificial Neural Network, Non-holonomic Systems and other algorithms by using different methods gives different performances for the desired output. Therefore, for this project, the hybrid technique includes Non–holonomic Constraints Systems and RTOS is proposed to solve path planning obstacles avoidance problem in a dynamic environment.

Chapter 3

Path Planning and Mapping

3.1 Overview

With the advancement of technology making everything so convenient these days and ages, autonomous systems are a very interesting area of innovating technology. A variety of robotic systems have been developed, and they have shown their effectiveness in performing different kinds of tasks including smart home environments, airports, shopping malls, manufacturing laboratories, etc. A huge number of research problems exist, such as navigation, which is one of the fundamental problems of mobile robotics systems, avoiding obstacles, timing, complexity and shortest path. Those problems should be determined to reach the goal by an efficient and right method.

To solve the robot navigation problem, we need to find answers to the three following questions: Where am I? Where am I going? How do I get there? These three questions are answered by the three fundamental navigation functions: localization, mapping, and motion planning.

- Localization: determines the robot's location in the environment by several methods (sensors, cameras, etc.), expressed as topological coordinates or absolute coordinates.
- Mapping: helps the robot know its direction and location; it can be placed manually, or the robot discovers its environment.
- Path planning: requires an appropriate addressing scheme that the robot can follow.

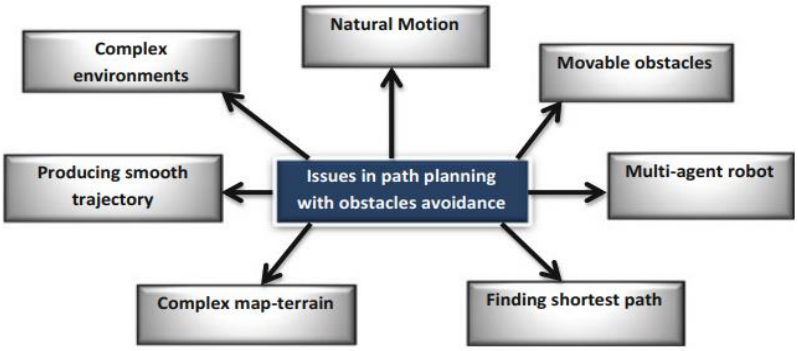


Fig3.1. Different issues of path planning

Path-planning is a key issue in the automatic parking assist system due to the non-holonomic constraints. A shortest path algorithm for the parallel parking problem in a certain condition is proposed and proved. A feasible path-planning approach is presented to meet the requirement that the parking space is narrow and needs parking iteratively by improving the shortest path. Considering several possibilities of collision with obstacles in the parking process, the parking region where the cars can park with no collision based on the proposed algorithm is designed. The proposed algorithm is verified combined with vehicle dynamics constraints under the limitation of the steering angle rotating speed.

In this project, after synthetically considering problems such as path optimization, narrow space parking, and path with no collision, we propose one kind of shortest collision-free path in a certain condition to approach the car to the goal, following a stable trajectory while avoiding the obstacles. And an iterative path-planning algorithm is presented to park in the narrow space, which is not big enough to operate parking at one time.

3.2 Path Planning Categories

Path Planning has some categories that define the nature of its environment, the knowledge of map that robot feeder by, and how to complete its mission.

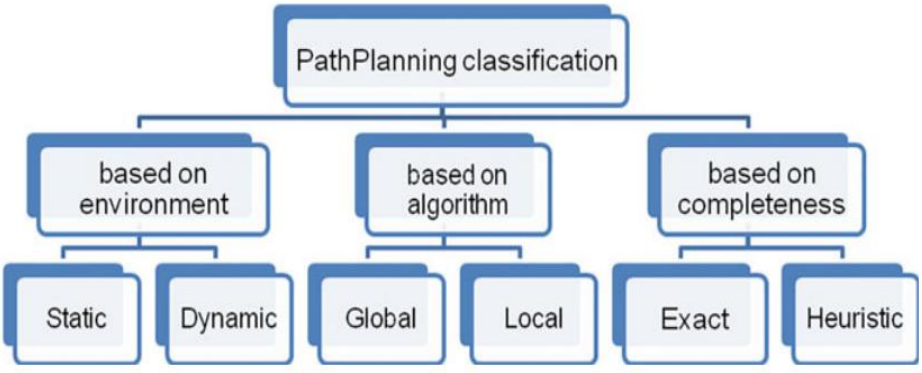


Fig3.2. Path Planning Categories

Environment Nature: The path planning problem can be done in both static and dynamic environments. A static environment is unvarying, the source and destination positions are fixed, and obstacles do not vary locations over time.

However, in a dynamic environment, the location of obstacles and goal position may vary during the search process. Typically, path planning in dynamic environments encompasses is more complex than that in static environments due to uncertainty of the environment. As such, the algorithms must adapt to any unexpected change such as the advent of new moving obstacles in the preplanned path or when the target is continuously moving. When both obstacles and targets are moving, the path planning problem becomes even more critical as it must effectively react in real time to both goal and obstacle movements. The path planning approaches applied in static environments are not appropriate for the dynamic problem.

Map knowledge: Mobile robots' path planning basically relies on an existing map as a reference to identify initial and goal location and the link between them. The amount of knowledge to the map plays an important role for the design of the path planning algorithm. According to the robot's knowledge about the environment, path planning can be divided into two classes: In the first class, the robot has an a priori knowledge about the environment modeled as a map. This category of path planning is known as global path planning or deliberative path planning. The second class of path planning assumes that the robot does not have priori information knowledge about its environment (i.e., uncertain environment). Consequently, it must sense the locations of the obstacles and construct an estimated map of the environment in real time during the search process to avoid obstacles and acquire a suitable path toward the goal state. This type of path planning is known as local path planning or reactive navigation.

Local Path Planning	Global Path Planning
Sensor based	Map-based
Reactive navigation	Deliberative navigation
Fast response	Relatively slower response
Suppose that the workspace area is incomplete or partially incomplete	The workspace area is known
Generate the path and moving toward target while avoiding obstacles or objects	Generate a feasible path before moving toward the goal position
Done online	Done offline

Table3.1. Global and local path planning

Completeness: Depending on its completeness, the path planning algorithm is classified as either exact or heuristic. An exact algorithm finds an optimal solution if one exists or proves that no feasible solution exists. Heuristic algorithms search for a good-quality solution in a shorter time.

When looking at our project, it consists of an unknown or unstructured environment (Dynamic environment), the robot must know the surrounding environment through sensors and controllers (Local path planning) and choose the best way to navigate.

3.3 Path Planning Spatial Representations

Qualitative (Route) Path Planning: In this category, there is no priori map that represents the world, but it is rather specified by routes from the initial to target location. The world is represented as a connection between landmarks, and a sequence of connected landmarks will represent the route. This approach is like how humans describe a route in their natural language. Landmarks can be any kind of recognizable and perceivable objects that can uniquely identify a location.

Metric (Layout) Path Planning: The world is specified using a layout representation that is the map. The map provides an abstract representation of the world (e.g., street layout, intersections, roads). It must be noted that it is possible to generate routes based on layout representation of the environment, but not in other way. In robotics, metric path planning is more attractive than qualitative path planning as it is possible to represent the environment in a clear structure that the robot can use to plan and find its path. Furthermore, it allows the robot to reason about space. In fact, metric maps decompose the environment into a set of distinct points, called waypoints, which are fixed locations identified by their (x,y) coordinates. Planning the path will then consist in finding the sequence of connected waypoints that lead to the goal position while minimizing the path cost. The path cost can be defined as the path length, or path with minimum energy consumption, or the path delay, etc., depending on one of the problem requirements.

3.4 Vehicle Kinematic Model

Our case study will follow non-holonomic constraints for vehicles and use a low-speed method.

3.4.1 Holonomic vs Non-Holonomic Systems

A holonomic system refers to a system that can independently control its position and orientation in all degrees of freedom. In a holonomic system, each degree of freedom is controllable. This means that a holonomic robot can move laterally, rotate, and change orientation without needing to navigate in arcs or make complex maneuvers.

Non-holonomic Systems that have constraints on their motion and cannot move independently in all degrees of freedom. The number of control inputs (such as velocities or steering angles) is fewer than the number of degrees of freedom. As a result, the system cannot instantaneously change its position or orientation in any direction. Instead, it must follow certain constraints and specific motion patterns to achieve the desired movements.

3.4.2 Vehicle Kinematic

It is assumed that the vehicle moves with the non-sliding method in the parking process because of the low speed. In the reference coordinate system, r is the midpoint of the rear wheel, f is the midpoint of the front wheel, x=x(t) and y=y(t) are the coordinates of r, θ=θ(t) is the course angle of the car with respect to the global coordinate system, φ=φ(t) is the steering angle, v=v(t) is the velocity of f, l is the wheel base, and R is the turning radius of r.

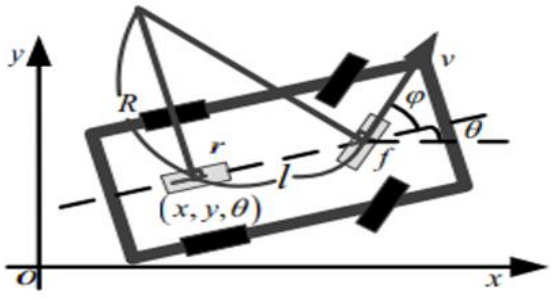


Fig3.3. Kinematic model of front wheel drive car

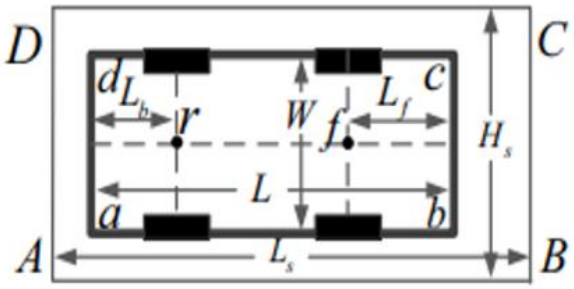


Fig3.4. Description of car and parking space

The parking environment can be constructed with information from sensors. In which abcd represents the four corners of the car, while ABCD represents the four corners of parking space, let the length of the rear and front overhang be Lb and Lf, L and H are the length and width of the car, and the length and width of parking space, which dominate the difficulty of parking, are defined as Ls and Hs.

3.5 Shortest Path Design

Work was done to invent and design the shortest collision-free path with the fewest number of maneuvers.

3.5.1 Shortest Path Design for Parallel Parking

The working procedure of a path planner is partitioned into two parts:

- Firstly, the system decides whether the space is big enough to park by analyzing the information from the ultrasonic sensors.
- Secondly, the planner generates a feasible collision-free parking path with the consideration of choosing the appropriate start and end positions if the space meets the parking requirement.

Circle - Straight Line - Circle Path (CSC)

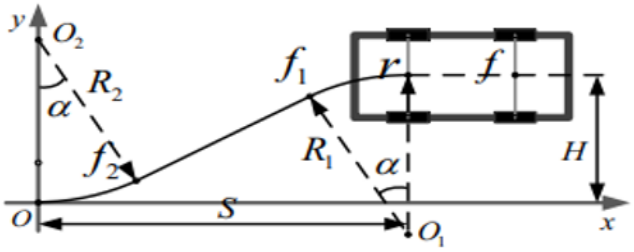


Fig3.5. CSC path

Where O1 and O2 are the circle centers of the first and final stage, R1 and R2 are the radius of the circles O1 and O2, f1 and f2 are tangency points between lines and circles O1 and O2. αis the angle of the arc path, O the origin of the coordinate, and S and H are the horizontal and vertical coordinates of the start position. From Fig. the

coordinates of O1 and O2 are $(x_1, y_1) = (S, H-R_1)$ *and* $(x_2, y_2) = (0, R_2)$. The parking space is defined as a narrow space when the car fails to park by one operation. For narrow parking spaces, drivers may move forward and backward several times to park; the shortest parking path is improved to adjust the narrow space circumstances to a certain extent.

The path can be described as the equations:

$$\begin{cases} r \to f_1 : (x-S)^2 + \left[y - (H - R_{\min}) \right]^2 = R_{\min}^2 \\ f_1 \to f_2 : kx - y + m = 0 \\ f_2 \to O : x^2 + (y - R_{\min})^2 = R_{\min}^2 \end{cases}.$$

Where the values of k, m and the coordinates of f1, f2 are as follows:

$$\begin{cases} k = \dfrac{S(H-2R_{\min}) + \sqrt{4R_{\min}^2 (S^2 + H^2) - 16R_{\min}^3 H}}{S^2 - 4R_{\min}^2} \\ m = R_{\min}\left(1 - \sqrt{1+k^2}\right) \\ (x_{f_1}, y_{f_1}) = \left(S - \dfrac{k}{\sqrt{1+k^2}} R_{\min}, H - \left(1 - \dfrac{1}{\sqrt{1+k^2}}\right) R_{\min} \right) \\ (x_{f_2}, y_{f_2}) = \left(\dfrac{k}{\sqrt{1+k^2}} R_{\min}, \left(1 - \dfrac{1}{\sqrt{1+k^2}}\right) R_{\min} \right) \end{cases}$$

Let the total length of the path be S0 and the length of the line path be S1, and equations can be obtained

$$\begin{cases} (R_1 + R_2)\sin\alpha + S_1 \cos\alpha = S \\ (R_1 + R_2)(1 - \cos\alpha) + S_1 \sin\alpha = H \\ S_0 = S_1 + (R_1 + R_2)\alpha \end{cases}$$

Where,

$$\begin{cases} S_1 = \dfrac{S - (R_1 + R_2)\sin\alpha}{\cos\alpha} = \dfrac{H - (R_1 + R_2)(1 - \cos\alpha)}{\sin\alpha} \\ S_0 = \left(\dfrac{H\cos\alpha - S\sin\alpha}{\cos\alpha - 1} \right)\left(\alpha - \dfrac{1 - \cos\alpha}{\sin\alpha} \right) + \dfrac{H}{\sin\alpha} \end{cases}.$$

In this range there are $f(a) > 0$, $S'(a) > 0$, from which S_0 increase with the sum of two radius increases.

The R_{min} *is* $R_{min} = L / tan(\varphi_{max})$ so the shortest path is generated when $R_1 = R_2 = R_{min}$.

3.5.1 Shortest Path Design for Perpendicular Parking

The working procedure of a path planner is partitioned into two parts:

- Firstly, the system decides whether the space is big enough to park by analyzing the information from the ultrasonic sensors.
- Secondly, the planner generates a feasible collision-free parking path with the consideration of choosing the appropriate start and end positions if the space meets the parking requirement.

Circle - Straight Line (CS)

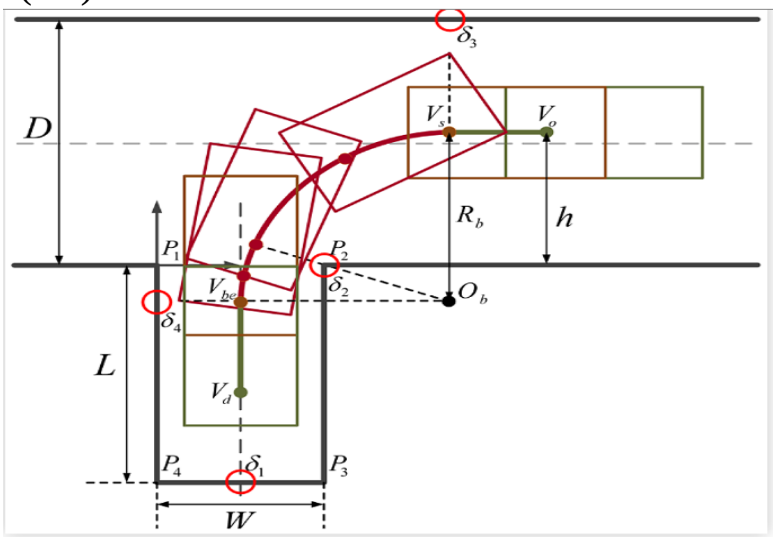


Fig3.6. CSC path

It is customary to use $P_{1,2,3,4}$ to represent the four corner points of the slot to be parked, the origin of the parking slot coordinate system is considered to be at point P_1 [$(x_{p1}, y_{p2}) = (0,0)$]. V_o (x_o,y_o) is the initial location beginning to park; V_s (x_s,y_s) is the location of starting to turn steering; V_{be} (x_{be},y_{be}) is the location of starting to turn back the steering wheel; V_d (x_d,y_d) is the final stop location. There are four potential collision points in parking process: The rear bumper of vehicle collides with the bottom of parking

slot, the right side of rear axle collides with the corner point P_2, the front left corner collides with the left side of aisle, the rear left corner collides with the left edge of parking slot or the right side of aisle.

Denoting R_b as the constrained turning radius of the rear axle center, the coordinate of turning center O_b in this backward maneuver can be calculated as:

$$x_{O_b} = x_s + R_{b\min 2} \cdot \sin\theta_s = \sqrt{L_r^{\,2} + \left(R_{b\min 2} + \frac{W_a}{2} \right)^2} + \delta_4$$

$$\begin{aligned} D - y_{O_b} &= D - (y_s - R_{b\min 1} \cdot \cos\theta_s) \\ &= \sqrt{(L_a - L_r)^2 + \left(R_{b\min 1} + \frac{W_a}{2} \right)^2} + \delta_3 \end{aligned}$$

The final stop position and orientation depend on the coordinates of angular points and the depth of slot, V_d is set as:

$$\begin{cases} x_d = W / 2 \\ y_d = \max\left(L_r - L_a, \delta_1 + L_r - L \right) \\ \theta_d = \pi / 2 \end{cases}$$

3.6 Parking Space Environment

The start and end positions are given. But the path planner in the parking assist system should also be concerned about the collision avoidance problem based on the planned path.

3.6.1 Parking Space Environment for Parallel Parking

R_{min} occurs when φ is *max*,

$$\boldsymbol{R_{min} = l / tan(\varphi_{max})}$$

The side clearance between ab and AB is

$$d_2 = \sqrt{(R_{\min} + W/2)^2 + L_b^2} - R_{\min} - W/2$$

The front clearance between b and C is

$$R_b = \sqrt{(l + L_f)^2 + (R_{\min} + W/2)^2}$$

So, the minimum length of parking space is

$$\min L_s = \sqrt{R_b^2 - (R_{\min} + W/2 + d_2 - H_s)^2} + L_b$$

The minimum width the of parking space is

$$H_s = W + d_2$$

The minimum parallel distance between the car and an obstacle is

$$d_3 = (R_{\min} - W/2) - \sqrt{(R_{\min} - W/2)^2 - (S - L_s + L_b)^2} \quad (16)$$

The width of the track is

$$d_4 = R_b + (R_{min} + W/2)$$

3.6.2 Parking Space Environment for Perpendicular Parking

$R_{min} = R_b$ occurs when φ is *max*,

$$R_{min} = l / tan\ (\varphi_{max})$$

To avoid collision with the boundary line $P_{3,4}$ of parking slot, at the location V_{be} stopping to turn,

$$H - R_b > y_d$$

After restricting the turning radius, the pull-in condition will also be checked by the collision of right side of rear axle with P_2 set as,

$$\sqrt{\left(R_b - \frac{W}{2}\right)^2 + (h - R_b)^2} < R_b - \frac{W_a}{2} - \delta_2$$

3.7 Path Planning Design

The actual dimensions of the vehicle should be considered, the parameters of the test vehicle are:

- The car's length (L_Car) = 41.5 cm (about 1.36 ft)
- The car's width (W_Car) = 18 cm (about 7.09 in)
- The wheelbase (L_axi) = 27 cm (about 10.63 in)
- The front overhang (Lf) = 6.25 cm (about 2.46 in)
- The rear overhang (Lb) = 8.25 cm (about 3.25 in)
- The maximum turning angle for steering wheel is within a range of (45:35) for each side, The maximum turning angle (phi_max) = 45
- The ratio between wheelbase and cross base, the coefficients average values are gained is (0.55, 0.56, 0.66), The ratio between wheelbase and cross base (kb) = W_Car / L_axi = 0.66
- The turning ratio (R_min) = L_axi / Phi_max = 27 cm (about 10.63 in)

3.7.1 Path Planning for Parallel Parking

To evaluate the proposed geometric path planning and steering controls for parallel reverse parking in one trial, simulation results using MATLAB were conducted.

To specify the parking space environment, the following consideration parameters should be taken:

- Minimum side clearance (d2) = 0.87 cm (about 0.34 in)
- Minimum front clearance (Rb) = 35.81 cm (about 1.17 ft)
- Minimum length of parking space (Ls) = 39 cm (about 1.28 ft)
- Minimum Width of parking space (Hs) = 16 cm (about 6.3 in)

Initial position and orientation of the vehicle:

- The horizontal coordinate of start position (S) = Ls
- The minimum parallel distance between the car and the obstacle (d3)
- The vertical coordinate of start position (H) = W_Car + d3

We have three phases of movement,

1. From start position r to f1
2. From f1 to f2
3. From f2 to end position O

With this information, we can create a path plan for our robot, create the environment surrounding it, and know whether these plans are useful and serve their purpose or not.

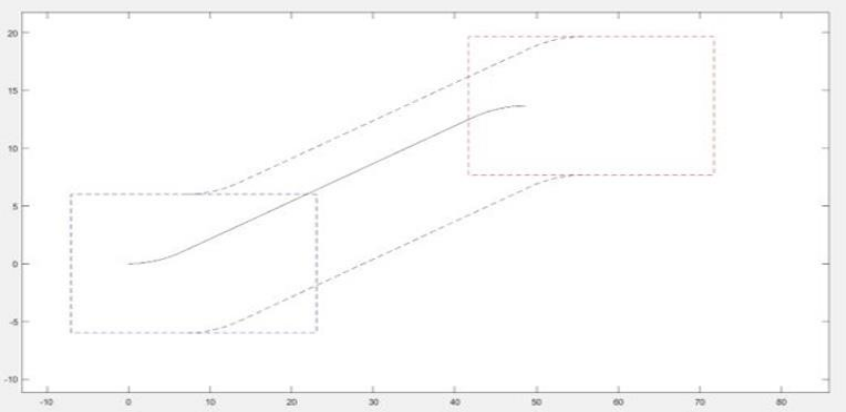


Fig3.7. Path Generation

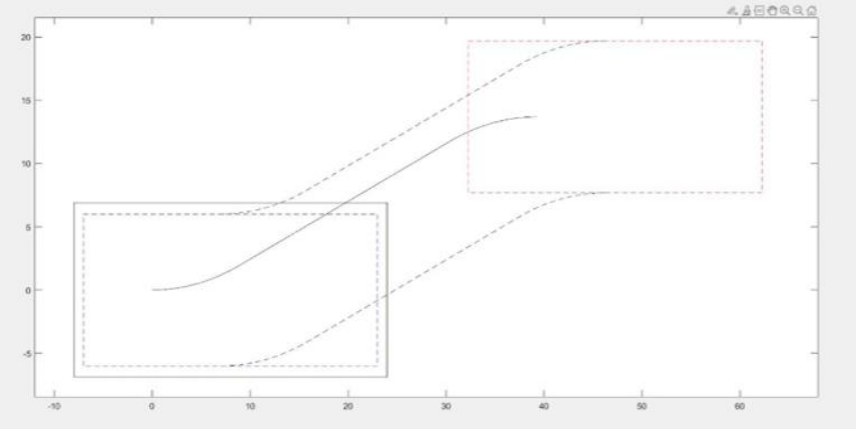


Fig 3.8. The Car Follows the Path

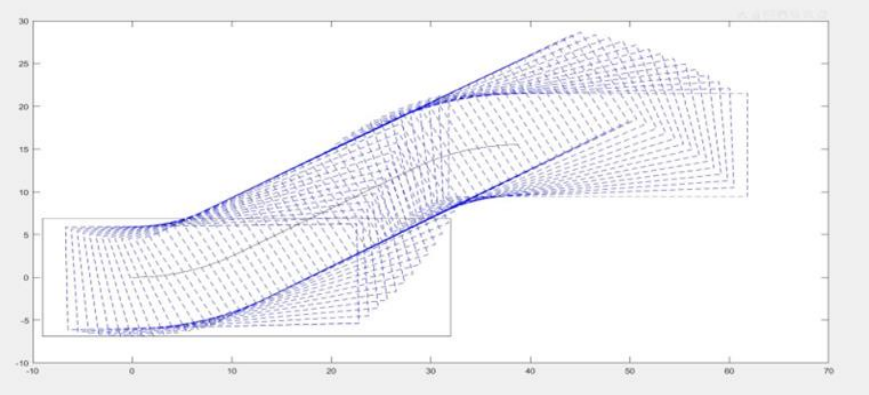


Fig3.9. Model Verification

3.7.2 Path Planning for perpendicular Parking

To evaluate the proposed geometric path planning and steering controls for perpendicular reverse parking in one trial, simulation results using MATLAB were conducted.

To specify the parking space environment, the following consideration parameters should be taken:

- Minimum side clearance = 1.6 cm (about 0.63 in)
- Minimum track width (D) = 20 cm (about 7.87 in)
- Minimum length of parking space (Ls) = 20 cm (about 7.87 in)
- Minimum Width of parking space (Hs) = 40 cm (about 1.31 ft)

Initial position and orientation of the vehicle:

- The horizontal coordinate of start position (S) = Ls
- The minimum parallel distance between the car and the obstacle (d3)
- The vertical coordinate of start position (H) = Hs

We have two phases of movement,

1. From start position V_s to V_{be}
2. From V_{be} to V_d

With this information, we can create a path plan for our robot, create the environment surrounding it, and know whether these plans are useful and serve their purpose or not.

3.8 Summary

In this chapter, an overview of the path planning problem is given; The different categories of this problem, the methods used to solve it, its complexity, etc. Many

intelligent algorithms have been proposed to solve this problem efficiently. These algorithms span across many technologies. But in the end, we settled on one algorithm that meets our requirements with the best performance.

One type of shortest path planning algorithm is proposed for automatic parallel, and perpendicular parking of non-holonomic vehicles in a certain situation. A practical model with iterative parking process in a narrow space is developed by optimizing the shortest path planning algorithm. Several collision conditions are taken into consideration to plan the start and end position area which makes the vehicle move without a collision. The model is verified in the vehicle dynamics model by high-resolution vehicle dynamics simulation software considering the steering wheel rotation speed. The results demonstrate the effectiveness of the proposed self-parking algorithm in a practical environment.

Chapter 4

Design

4.1 Overview

The essence of any car, self-driving or otherwise, is its ability to navigate its surroundings. This chapter delves into the two basic systems that make this mobility possible: the steering system and the drive system. It will be covered how to choose the correct vehicle dimensions, shape, and arrangement for the steering system and drive system, starting from design to implementation.

4.2 Planning and Selection Stage

The dimensions of the car were chosen in relation to the dimensions of a real car by taking the ratio of length to width; The dimensions of a real car were chosen so that the project would be as close to reality as possible and be able to be applied to modern car systems. A car was chosen Tesla Cybertruck,

- Length (A) = 5,682.9
- Width (B) = 2,413.3
- Ratio = A/B = 2.3

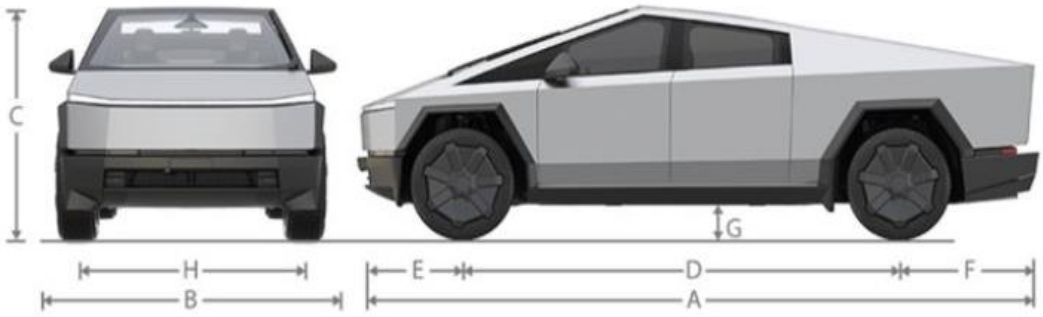


Fig 4.1. Tesla Cybertruck Car's Dimensions

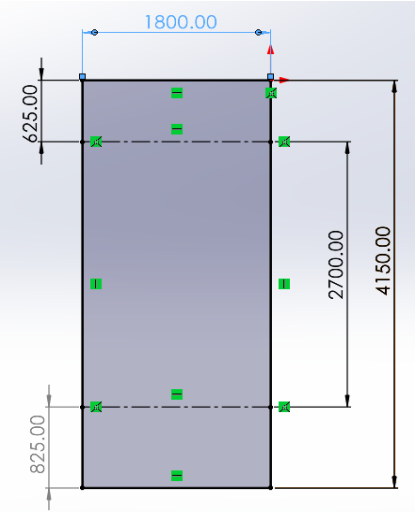


Fig 4.2. Our Car's Dimensions

The car has two basic systems: the steering system and the propulsion system. To design a steering system to meet the requirements of the project, it must be designed with a flexible, homogeneous system that is closer to the real systems. To design a propulsion system that suits the requirements of the project, a design must be chosen that is able to bear the loads and achieve stability for the entire vehicle.

Steering System: The vehicle's steering wheels are turned at different angles by a steering trapeze. Any turning radius corresponds to a concrete correlation between angles *Θār* and *Θie*. The maximum turning angle for Steering wheels are within a range of 35°...45° to each side, but the steering trapeze cannot provide a correct correlation within the whole range. Therefore, a trapeze geometry is set in a way that is correct Correlation for turning angles is provided at the turning angles of steering wheels that are commonly found referred to in user manuals. At a maximum turning angle of steering wheels, the divergence of angles from the theoretically correct angle is the greatest.

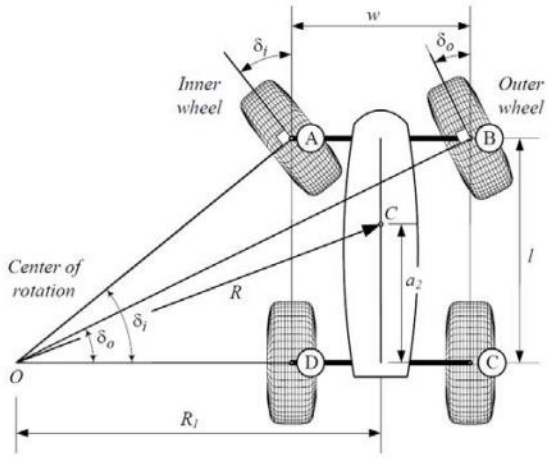


Fig 4.3. Ackerman Kinematics & Linkage for Front Wheel Steering

Ackermann's principle, also referred to as toe out on turns or steering radius, causes the outside wheel to reduce its turning angle on a turn (typically about 1° to 2° @ 20°-wheel turn). This change is necessary due to the extra distance the outside wheel must travel when the vehicle is turning. Without a toe change, the front tires squeal during low speed turning and vehicle instability in higher speed turns.

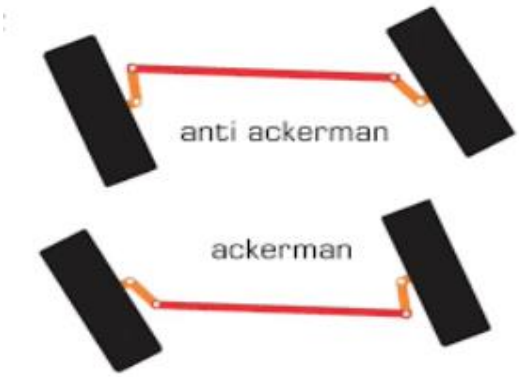


Fig 4.4. Ackerman Kinematics & Anti Ackerman

Drive System: It was designed with a rear-wheel drive system using two motors for each rear wheel. A motor was designed for them to avoid movement as a result of vertical and horizontal vibration. The wheel was connected to the two motors with a coupling system.

The body of the car was designed with platforms made of acrylic, and they were unloaded in specific places to be used in the electrical connection between the control components. 2 motors were used and 4 wheels were used, covered with rubber, with a diameter that suits the dimensions of the car, and the steering system (Ackerman System) consists of the two front wheels connected by a connecting rod Connector to servo motor.

4.3 Drawing Parts

The SOLIDWORK mechanical design program was used to draw the parts of the project. Some of these parts were purchased ready-made from the market, and some were printed on a 3D printer machine to reach the appropriate final shape. Among these parts are:

At the beginning of the design, we started with the wheel and the design was built on it. There is a difference between the front and rear wheels because the rear was used from the market, while the front was designed with the specifications of the rear wheel, but with modifications to suit the steering system.

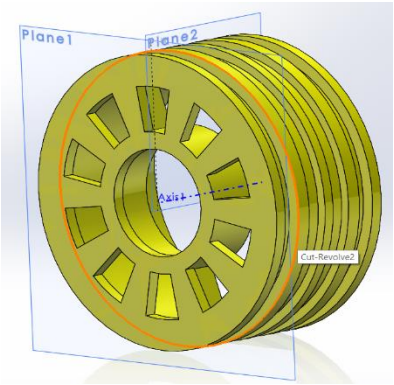


Fig 4.5. Front Wheel

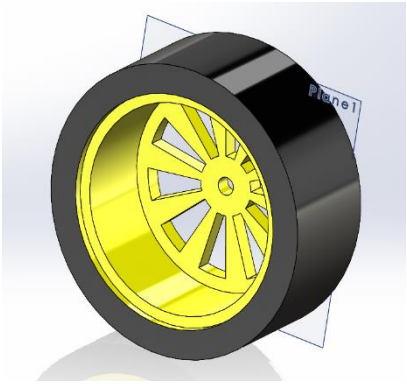


Fig 4.6. Rear Wheel

The car's body was designed using acrylic platforms to form the shape of the car and connect the car's mechanisms and components.

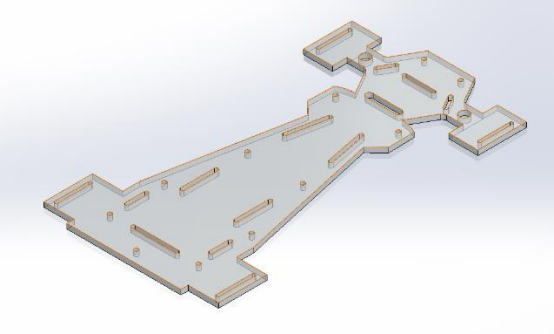


Fig 4.7. Top Pallet

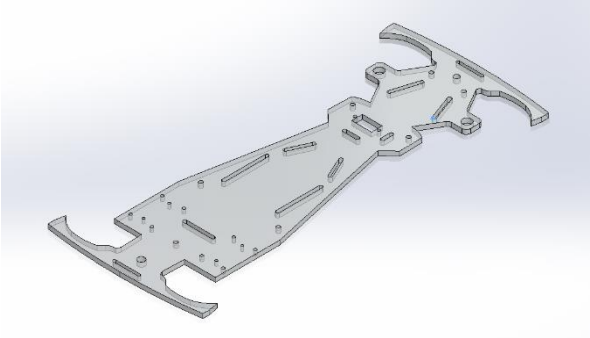


Fig 4.8. Bottom Pallet

The drive system was designed with two motors, each mounted on the rear wheels. The motor was also attached to a piece in the platforms, and its function is also to carry the load of the car along with the motor shaft.

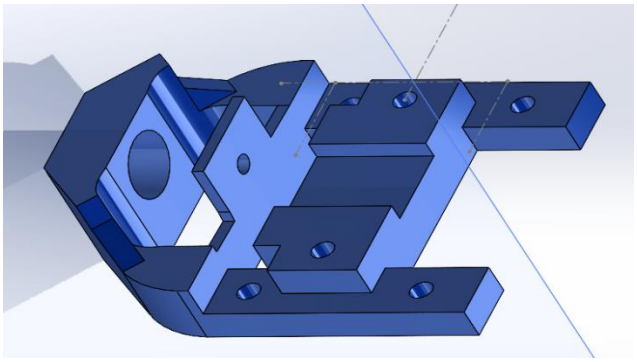


Fig 4.9. Installation and load carrying system

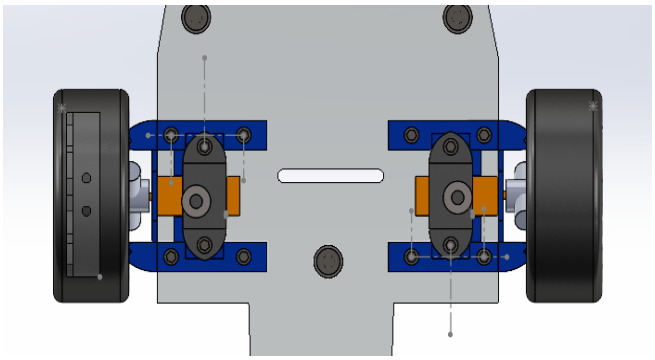


Fig 4.10. Drive System

The steering system is designed with the two front wheels having a wear socket. There is a screw inside the wheel that is attached to a mounting column for turning right and left. This column is attached to acrylic pads, and the two steering columns are connected with a link linked to the servo motor to move the two wheels together.

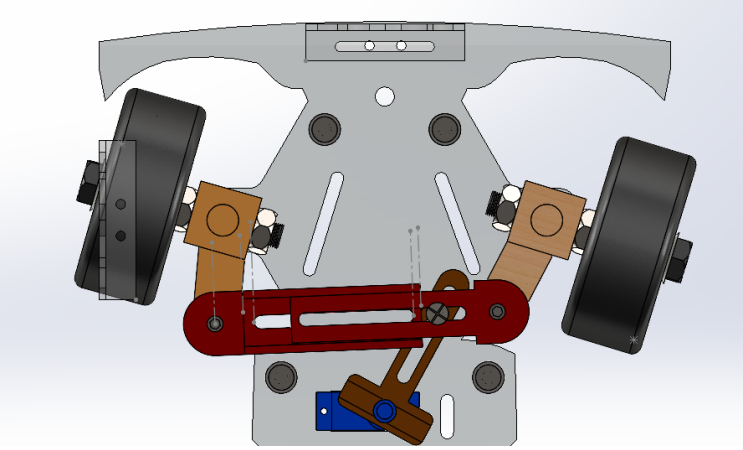


Fig 4.11. Steering system

4.4 Assembly Mechanism

In the assembly stage, all parts were assembled together, ensuring that there were no errors and the consistency of all parts in one mechanism to meet the requirements of the project, including a steering system and a propulsion system, with the dimensions of a real car and a good shape.

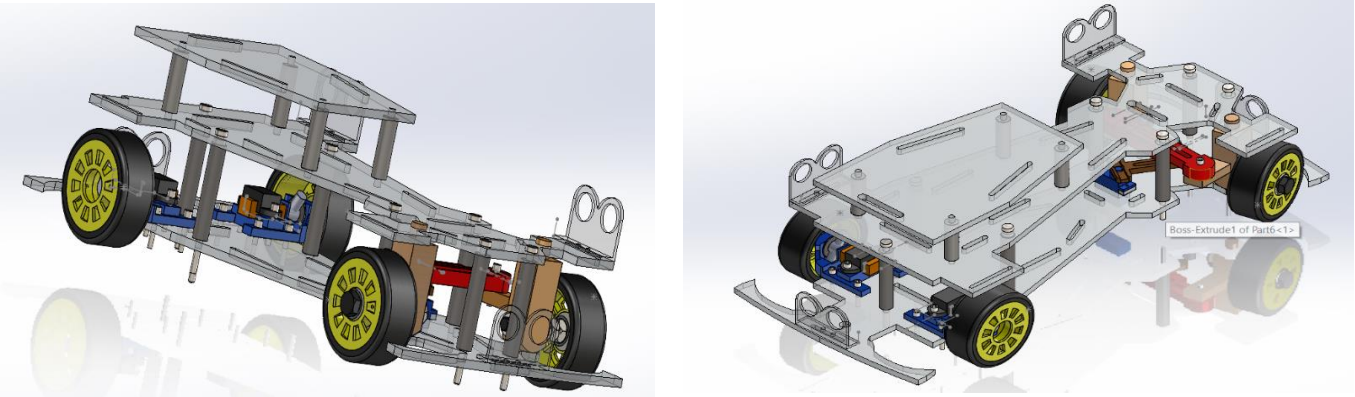


Fig 4.12. Car Assembly

4.5 Printing Parts

3D printing, With the design phase completed, we meticulously assembled all the parts into a cohesive 3D model. The intricate components were then 3D printed, adding a layer of modern manufacturing technology to our project. The decision to 3D print allowed for the creation of complex and customized parts, facilitating a more efficient assembly process

Fig 4.13. Wheels

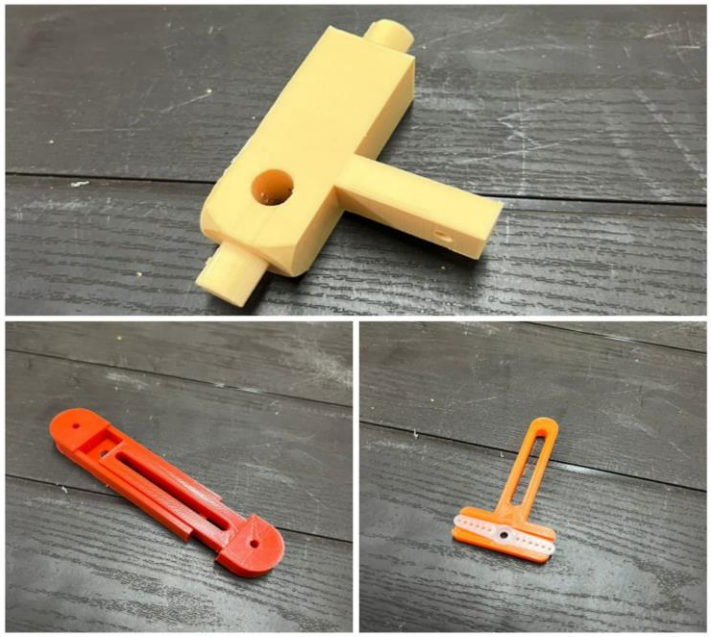


Fig 4.14. Steering Parts

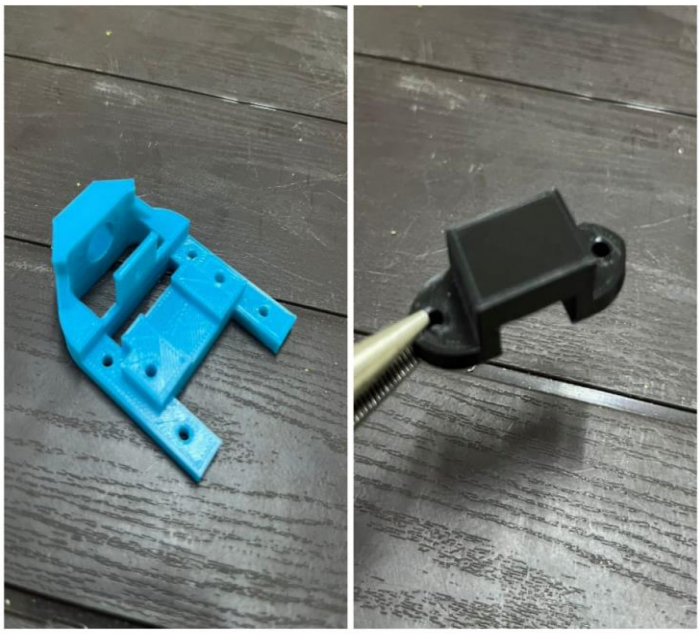


Fig 4.15. Drive Parts

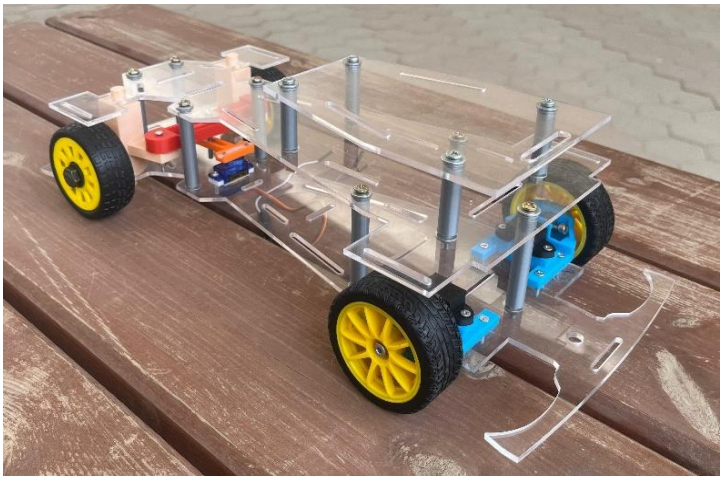
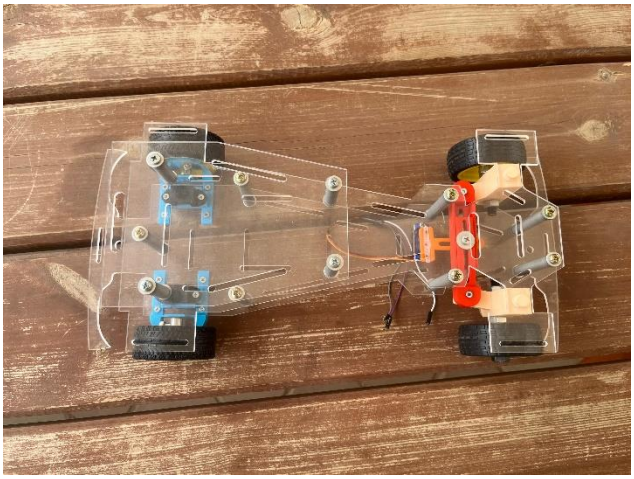
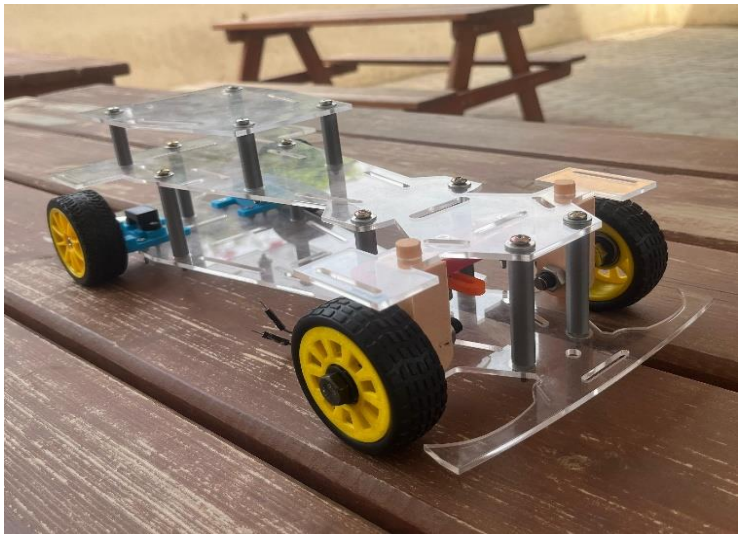




Fig 4.16. Car Body

4.6 Initial designs

Before arriving at the final design, two designs were made that had errors. These errors were noticed after they were completed, and this helped us avoid these errors in the final design and reach the best possible result that helps achieve the project objectives.

The first design was at a high height, which caused instability for the car because the car's body was much higher than its wheels, and the suspension of the motor and not being fixed well caused it to cause a large state of vibration, and the teeth of the gears were slipping past each other, as shown in the following Figures.

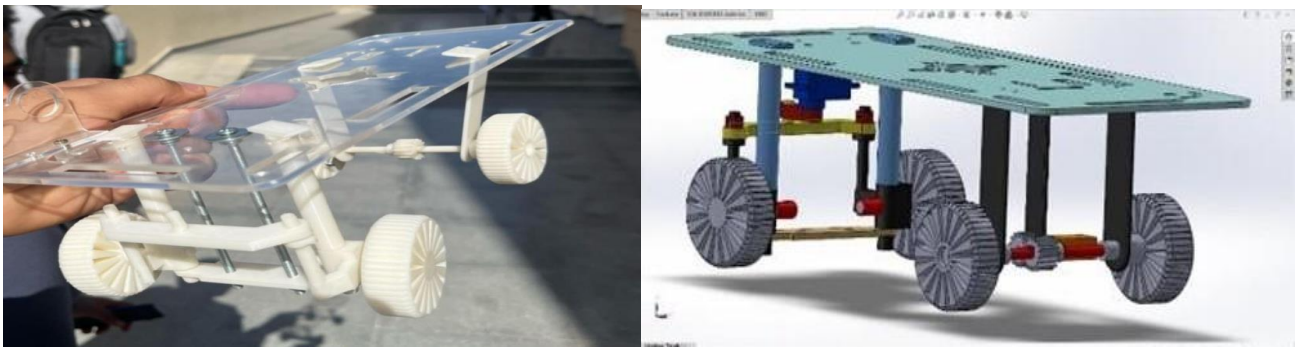


Fig 4.17 First Design

When the second design was made, the car was more stable after its height was lower, but it also did not give the required stability, and the wheels did not give a good performance because they were materials from a 3D printing machine, and there was always slipping on the ground and deviation in the car's path, which made the project's goals not be achieved. The third design was made with a layer in the middle of the wheels for better stability. Wheels supported by bearings from the market were used to avoid slipping or deviating in the track.

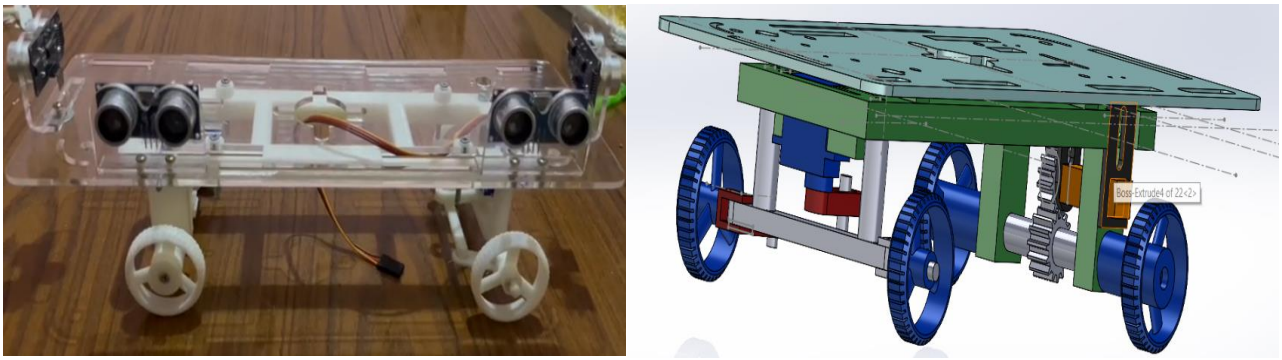


Fig 4.18 Second Design

4.6 Summary

In order to design a car that can move and maneuver that is close to reality in its dimensions and steering method, it was necessary to design a steering system with a flexible steering angle (30:45 degrees) in the manner of Ackerman and a rear-wheel drive system with large torque and medium to small speed. The parts were designed and tested using the SOLID Work program. The manufacturing was done at the lowest cost, the most appropriate manufacturing method (3D drawing), and the best shape, all to meet the requirements of the project.

Chapter 5

Components

5.1 Overview

In this chapter, we delve into the intricate components that constitute the self-parking car project, each playing a crucial role in its functionality and efficiency. From microcontrollers to sensors, motor drivers to power supplies, we explore the technical specifications, functionalities, and reasons behind their selection. Here's a brief overview of the components covered in this chapter:

A Blue Pill controller was chosen with 4 ultrasonic sensors to map the environment surrounding the robot, a servo motor to control the steering angle, 2 DC motors for propulsion and movement, a Dual H-Bridge Motor Driver, and 6 lithium batteries, which were divided into 3 batteries to feed the controller, sensors, and servo, and 3 batteries to feed the motors. 2 devises step down voltage, one for the controller and the other for the motor driver, and a 2 start and lock buttons.

5.2 Blue Pill - STM32F103C8 Microcontroller

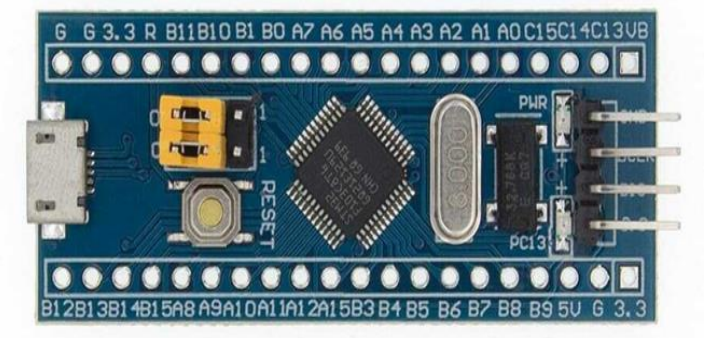


Fig 5.1. Microcontroller

The STM32F103C8 is a popular microcontroller within STMicroelectronics' STM32 family. It's based on the ARM Cortex-M3 core, which is a 32-bit RISC processor

architecture known for its balance of performance and power efficiency. Here are some key features and characteristics of the STM32F103C8.

Specifications:

- ARM Cortex-M3 Core: The Cortex-M3 core operates at a maximum frequency of 72 MHz and offers excellent performance for a wide range of embedded applications.
- Memory: The STM32F103C8 features various memory options, including up to 64 KB of Flash memory for program storage and up to 20 KB of SRAM for data storage.
- Peripherals: It includes a rich set of peripherals, such as multiple timers, UARTs, SPI, I2C interfaces, ADCs, DACs, and more. These peripherals provide flexibility and functionality for interfacing with external devices and sensors.
- GPIO: The microcontroller offers several General-Purpose Input/Output (GPIO) pins, which can be configured and controlled to interact with external circuits and devices.
- Low Power Features: The STM32F103C8 incorporates various low-power modes, allowing developers to optimize power consumption for battery-operated or energy-efficient applications.

5.3 ULTRASONIC SENSOR

Fig 5.2. ULTRASONIC SENSOR

PIN Number	PIN Name	Description
1	Vcc	The Vcc pin powers the sensor, typically with +5V.
2	Trigger	Trigger pin is an Input pin. This pin has to be kept high for 10us to initialize measurement by sending US wave.
3	Echo	Echo pin is an Output pin. This pin goes high for a period of time which will be equal to the time taken for the US wave to return back to the sensor.
4	Ground	This pin is connected to the Ground of the system.

Table 5 .1. PIN Data

HC-SR04 Sensor Features:

- Operating voltage: +5V
- Theoretical Measuring Distance: 2cm to 400cm
- Accuracy: 3mm (about 0.12 in)
- Measuring angle covered: <15°
- Operating Current: <15mA
- Operating Frequency: 40Hz

HC-SR04 Ultrasonic Sensor Working

As shown above the HC-SR04 Ultrasonic (US) sensor is a 4-pin module, whose pin names are VCC, Trigger, Echo and Ground respectively. This sensor is a very popular sensor used in many applications where measuring distance or sensing objects are required. The module has two eyes like projects in the front which forms the Ultrasonic transmitter and Receiver. The sensor works with the simple formula that *Distance = Speed × Time*.

The Ultrasonic transmitter transmits an ultrasonic wave. This wave travels in air and when it gets objected to by any material it gets reflected toward the sensor. This reflected wave is observed by the Ultrasonic receiver module as shown in the picture below.

Now, to calculate the distance using the above formulae, we should know the Speed and time. Since we are using the Ultrasonic wave, we know the universal speed of US wave at room conditions which is 330m/s. The circuitry inbuilt on the module will calculate the time taken for the US wave to come back and turns on the echo pin high for that same amount of time, this way we can also know the time taken. Now simply calculate the distance using a microcontroller or microprocessor.

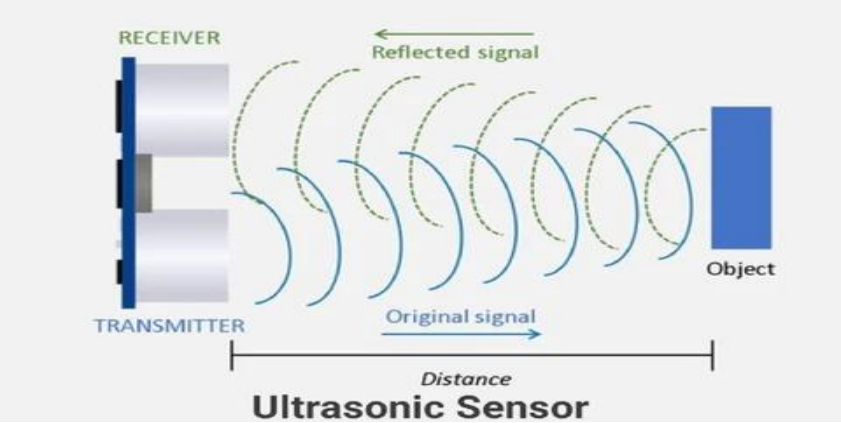


Fig 5.3. ULTRASONIC SENSOR SIGNALS

How to use the HC-SR04 Ultrasonic Sensor

HC-SR04 distance sensor is commonly used with both microcontroller and microprocessor platforms like AVR Arduino, ARM, PIC, Raspberry Pie etc...

Power the Sensor using a regulated +5V through the VCC and Ground pins of the sensor. The current consumed by the sensor is less than 15mA and hence can be directly powered by the on board 5V pins. The Trigger and the Echo pins are both I/O pins and hence they can be connected to I/O pins of the microcontroller. To start the measurement, the trigger pin must be made high for 10uS and then turned off. This action will trigger an ultrasonic wave at frequency of 40Hz from the transmitter and the receiver will wait for the wave to return. Once the wave is returned after it is getting reflected by any object the

Echo pin goes high for a particular amount of time which will be equal to the time taken for the wave to return to the sensor.

The amount of time during which the Echo pin stays high is measured by the MCU/MPU as it gives the information about the time taken for the wave to return to the Sensor.

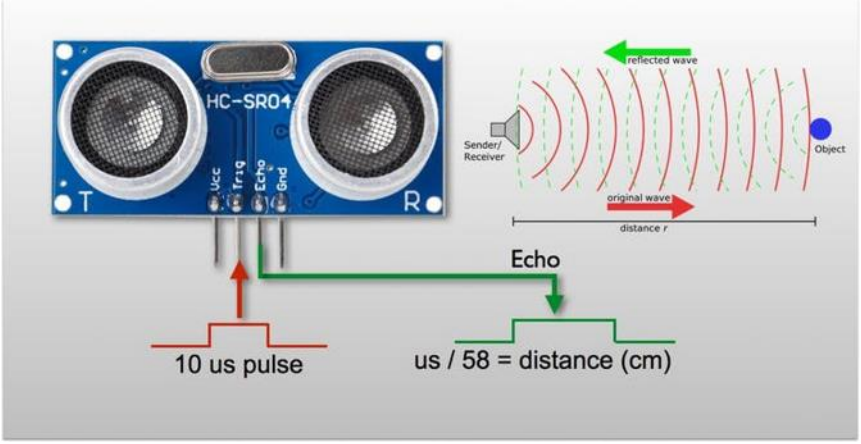


Fig 5.4. ULTRASONIC SENSOR OPERATION

Using this information, the distance is measured as explained in the above heading.

5.3.1 Why Ultrasonic sensor has been selected over Infrared (IR) sensor

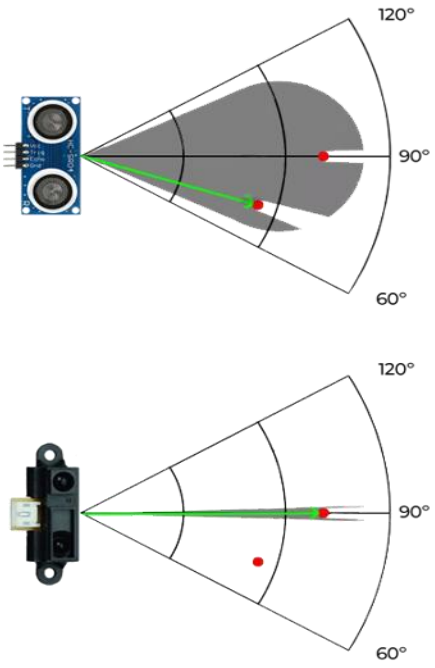


Fig 5.5. ULTRASONIC SENSOR BEZEL AND IR

Ultrasonic sensors and infrared (IR) sensors are both commonly used in various applications for proximity sensing, object detection, and distance measurement. Here's a comparison between the two:

	Ultrasonic sensor	Infrared (IR) sensor
Detection Range:	It has a longer detection range compared to IR sensors. It can typically detect objects within a range of a few centimeters to several meters, depending on the sensor's specifications.	Usually has a shorter detection range compared to ultrasonic sensors, typically ranging from a few centimeters to a couple of meters.
Environmental Factors:	Are less affected by ambient light conditions because they operate using sound waves rather than light.	Can be affected by ambient light sources, particularly sunlight and artificial lighting.
Accuracy:	Provide accurate distance measurements within their specified range.	May encounter inaccuracies due to the reflective properties of the object's surface and the ambient light conditions.

Table 5.2. ULTRASONIC SENSOR BEZEL AND IR

5.4 L298N Dual H-Bridge Motor Driver

This dual bidirectional motor driver is based on the very popular L298 Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently one motor up to 2A each in both directions. It is well suited for connection to a microcontroller requiring just a couple of control lines per motor. This board is equipped with power LED indicators, on-board +5V regulator and protection diodes.

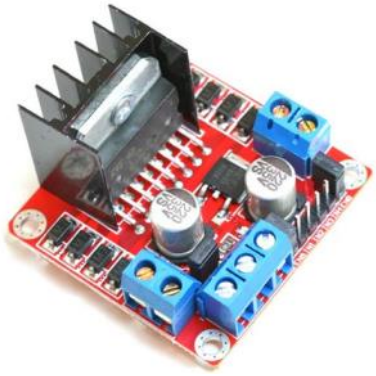


Fig 5.6. Motor Driver

Brief Data:

- Input Voltage: 3.2V~40Vdc.
- Power Supply: DC 5 V - 35 V
- Peak current: 2 Amp
- Operating current range: 0 ~ 36mA
- Maximum power consumption: 20W (when the temperature T = 75 ℃).

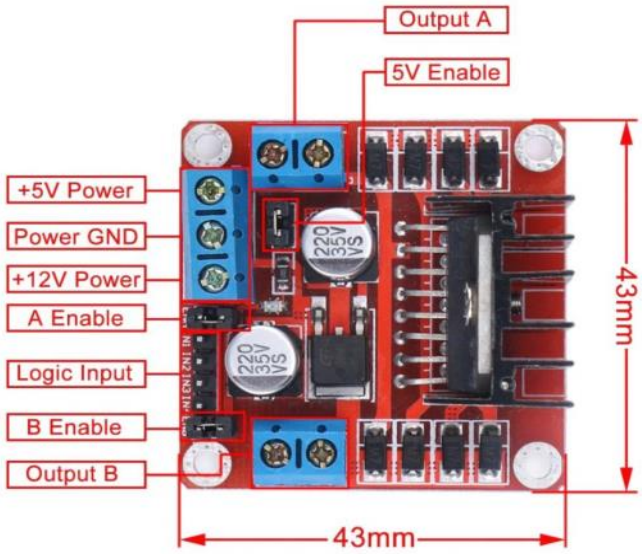


Fig 5.7. Motor Driver Specification

5.5 SERVO MOTOR

Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction).

Fig 5.8. SERVO MOTOR

Specifications:

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 8.5 kgf·cm (4.8 V), 10 kgf·cm (6 V)
- Operating speed: 0.2 s/60º (4.8 V), 0.16 s/60º (6 V)
- Operating voltage: 4.8 V to 7.2 V
- Metal Gears for longer life

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

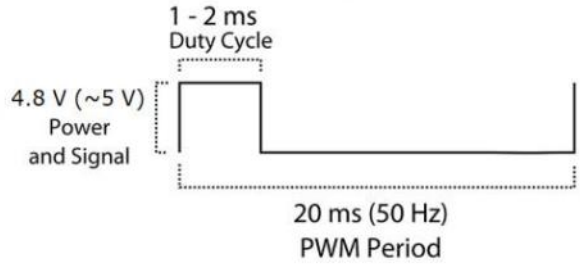


Fig 5.9. SERVO SIGNAL

The wire colors are Red = Battery (+), Brown = Battery (-), Orange = Signal.

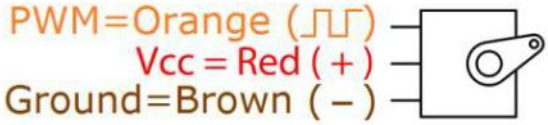


Fig 5.10. pin color

5.6 DC Geared Motor

Description:

- Full metal gears are wear resistant.
- Rated Voltage: 6~12V DC power supply.
- Torque: Refer to Table 1.
- Rated Current: 0.04A.
- Stall Current: 0.67A.
- Total Length: 34mm (about 1.34 in).
- Gearbox Size: 15 x 12 x 10mm (LxWxH).
- Shaft Size: Ø3 x 10mm (D*L). D-Type Shaft.
- Net Weight: 10g.

Fig 5.11. DC Geared Motor

Specification:

- Voltage: 6~12V DC
- Speed: 30 RPM
- Max. Gear Ratio: 1:1000
- 45mA (Current of No Load) 8 KG.CM (Stalling Torque)

5.6.1 Why DC motor has been selected over stepper motor?

Fig 5.12. DC Motor and Stepper

Here's a comparison between the two:

	DC Motor	Stepper Motor
Control:	Offer variable speed control, allowing smooth acceleration and deceleration.	Offer variable speed control, allowing smooth acceleration and deceleration.
Motion:	Have continuous motion.	Have incremental motion.
Weight and size:	Smaller size and weight which makes them more suitable.	Big size and weight.
Noise and heat:	Often quieter and generate less heat.	More noise and heat generation.

Table 5.3. Comparison Between DC Motor and Stepper Motor

5.7 Power Supply

Using rechargeable lithium batteries with a voltage rating of 3.7V is a common choice for powering various electronic devices and systems.

Total Voltage Output: When you connect three 3.7V batteries in series, the total voltage output of the battery pack would be the sum of the individual voltages, which equals 3.7V * 3 = 11.1V.

Fig 5.13. BATTERIES

5.8 Battery Management System (BMS)

Is technology dedicated to the oversight of a battery pack, which is an assembly of battery cells electrically organized in a row-x-column matrix configuration to enable delivery of a targeted range of voltage and current for a duration of time against expected load scenarios.

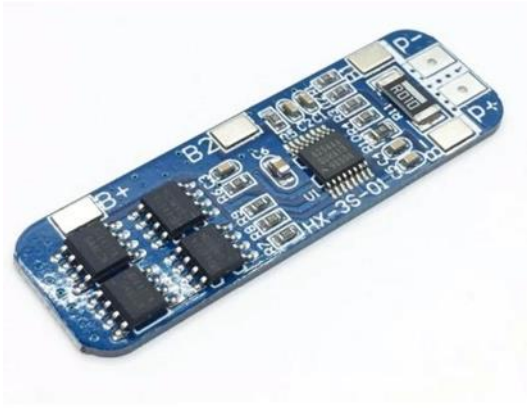


Fig 5.14. (BMS)

Specification:

- Size: approx.50x16x1mm
- Maximum discharge current: 10A
- Overcharge detection voltage: 4.25-4.35V±0.05V
- Over-discharge detection voltage: 2.3-3.0V±0.05V
- Maximum working current: 5A-8A
- Transient current: 9A-10A
- Operating temperature: -40~50°
- Storage temperature: -40~80°
- Service life: ＞30,000 hours

Feature:

- Short circuit protection.

- Overcharge protection.
- Over-discharge protection.
- Overcurrent protection.
- Can be applied to various 3.7V lithium cores.
- Compact size and lightweight.
- Ensure the security of battery pack.
- Suitable for 18650 or 3.7V lithium battery rated voltage

5.9 LM2596HVS-ADJ DC-DC Step-Down Module (3A)

Fig 5.15. Step-Down Module

- 3.3V, 5V, 12V, and adjustable output versions
- Adjustable version output voltage range, 1.2V to 37V
- Guaranteed 3A output load current
- Input voltage range up to 40V
- 150 kHz fixed frequency internal oscillator
- TTL shutdown capability
- Low power standby mode, IQ typically 80 μA
- High efficiency
- Uses readily available standard inductors
- Thermal shutdown and current limit protection

In our project, we required a reliable and efficient power converter for step-down voltage regulation. After careful consideration and evaluation, we selected the LM2596

SIMPLE SWITCHER Power Converter. This brief aims to explain the reasons behind choosing the LM2596 and highlight its key features and advantages.

5.10 Boat Rocker Switch ON-OFF

Fig 5.16. Boat Rocker Switch ON-OFF

5.11 Battery Holder (3 x 18650)

Fig 5.17. Battery Holder (3 x 18650)

5.12 Summary

This project involves the development of a self-parking car system incorporating both mechanical and electronic components.

The electronic components integrated into the system comprise:

1. Blue Pill Board: STM32F103C8 MCU based on ARM architecture with 64 KB of Flash memory for program storage and up to 20 KB of SRAM for data storage, it offers several General-Purpose Input/Output (GPIO) pins, it includes a rich set of peripherals, such as multiple timers, UARTs, SPI, I2C interfaces, ADCs, DACs, and more.
2. Ultrasonic Sensors: Positioned strategically, including one forward-facing, one rear-facing, and two lateral sensors, these devices enable accurate detection of obstacles and distances during parking maneuvers.
3. Servo Motor: Employed for precise steering control, allowing the vehicle to accurately adjust its direction during parking operations.
4. DC Geared Motor: Utilized for forward and backward movement, enabling controlled movement of the vehicle in response to parking commands.
5. Stepdown Converters: Two converters are employed to regulate voltage, ensuring a variable output from 1V to 60V to power various electronic components within the system.
6. Battery Management System (BMS): Essential for managing and monitoring the performance of the six 18650 lithium batteries, ensuring optimal power distribution and safety.
7. Lithium Batteries (18650): Six rechargeable lithium-ion batteries provide the necessary power to the system, ensuring prolonged operation and reliability.

By combining these mechanical and electronic components, the self-parking car system aims to automate parking processes, enhancing convenience and safety for users.

Chapter 6

Control System

6.1 Overview

In this chapter, we delve into the circuits and electrical components that make up a self-parking car project, each of which plays a critical role in its functionality and efficiency. Starting from providing and regulating power for microcontrollers and sensors and working on purifying power for some components to increase their efficiency, operating and controlling motors, regulating (charging and discharging) power sources, and providing appropriate power for each component in the control circuit. We explore the technical specifications and functions.

6.2 Electric Characteristic for the system

6.2.1 Blue pill Board

- VDD = 2.0 to 3.6 V: external power supply for I/Os and the internal regulator.
- Provided externally through VDD pins.
- VSSA, VDDA = 2.0 to 3.6 V: external analog power supplies for ADC, reset blocks, RCs, and PLL (minimum voltage to be applied to VDDA is 2.4 V when the ADC is used).
- VDDA and VSSA must be connected to VDD and VSS, respectively.
- VBAT = 1.8 to 3.6 V: power supply for RTC, external clock 32 kHz oscillator and backup registers (through power switch) when VDD is not present.

The STM32F103xx performance line supports three low-power modes to achieve the best compromise between low-power consumption, short startup time and available wakeup sources:

- Sleep mode
- Stop mode

- Standby mode

6.2.2 Ultrasonic Sensor

- Operating voltage: +5V
- Operating Current: <15mA
- Operating Frequency: 40Hz

65.2.3 L298N Dual H-Bridge Motor Driver

- Input Voltage: 3.2V~40Vdc.
- Power Supply: DC 5 V - 35 V
- Peak current: 2 Amp
- Operating current range: 0 ~ 36mA
- Maximum power consumption: 20W

6.2.4 Servo Motor

- Operating voltage: 4.8 V (~5V)
- The wire colors are Red = Battery (+), Brown = Battery (-), Orange = Signal.

6.2.5 DC Motor

- Rated Voltage: 6~12V DC power supply.
- Rated Current: 0.04A.
- Stall Current: 0.67A.
- 45mA (Current of No Load) 8 KG.CM (Stalling Torque)

6.2.6 Lithium Batteries

- Using rechargeable lithium batteries with a voltage rating of 3.7V is a common choice for powering various electronic devices and systems.

- Total Voltage Output: When you connect three 3.7V batteries in series, the total voltage output of the battery pack would be the sum of the individual voltages, which equals 3.7V * 3 = 11.1V.

6.2.7 Battery Management System (BMS 3S 10A 12V)

- Maximum discharge current: 10A
- Overcharge detection voltage: 4.25-4.35V±0.05V
- Over-discharge detection voltage: 2.3-3.0V±0.05V
- Maximum working current: 5A-8A
- Transient current: 9A-10A

6.2.8 LM2596HVS-ADJ DC-DC Step-Down Module (3A)

- 3.3V, 5V, 12V, and adjustable output versions
- Adjustable version output voltage range, 1.2V to 37V
- Guaranteed 3A output load current
- Input voltage range up to 40V
- 150 kHz fixed frequency internal oscillator
- TTL shutdown capability
- Low power standby mode, IQ typically 80 μA

6.3 Circuit Design

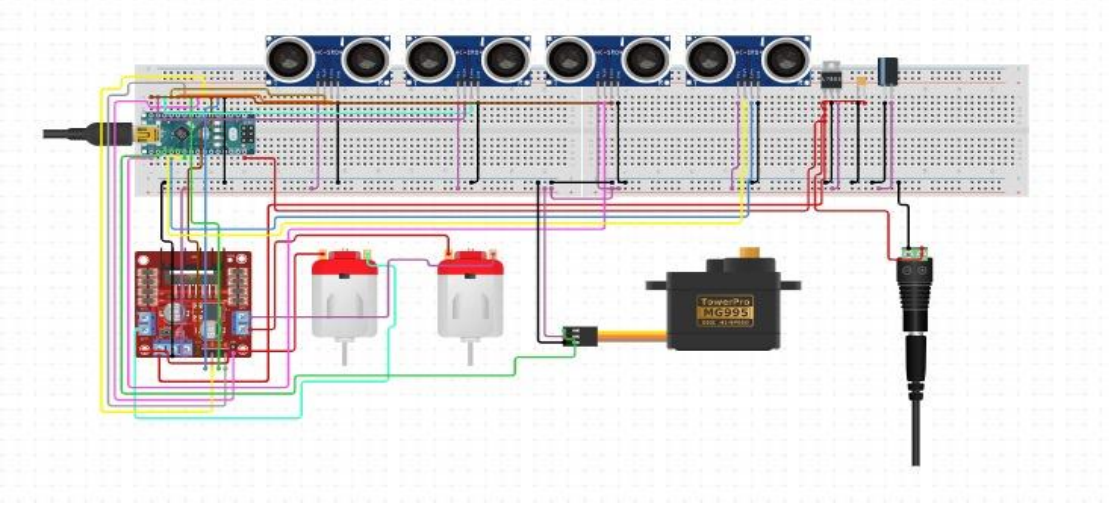
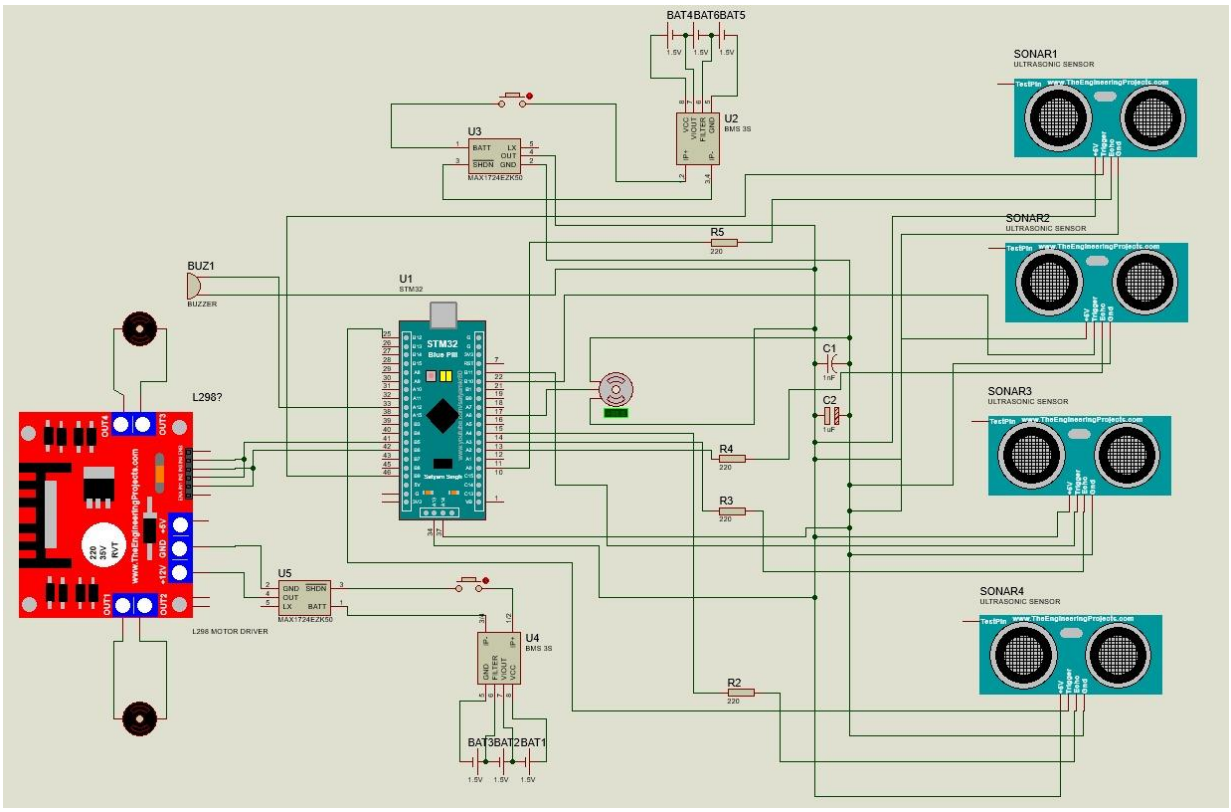


Fig 6.1. Circuit Design



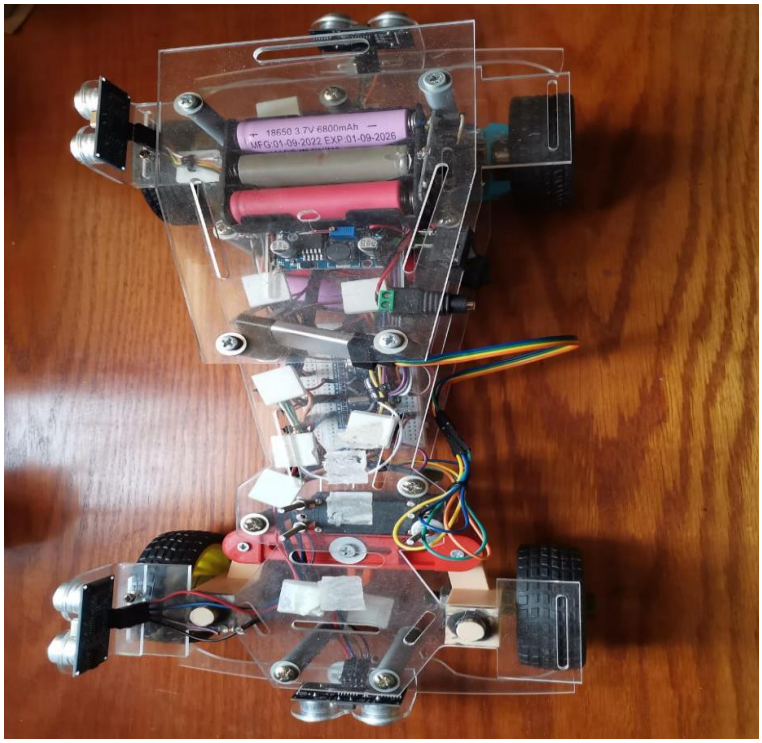


Fig 6.2. Car Circuit Design

Chapter 7

Software

7.1 Overview

The embedded system software for the auto parking car project aims to enhance the parking experience by efficiently managing and controlling multiple systems within the parking facility. Its main tasks include identifying available parking spots and automatically guiding cars to them, thus improving efficiency and saving time and effort for users.

The software program is a set of layers, to connect with the microcontroller and the application.

- Microcontroller Abstraction Layer (MCAL)
- Hardware Abstraction Layer (HAL)
- Real-Time Operating System Layer
- Application Layer

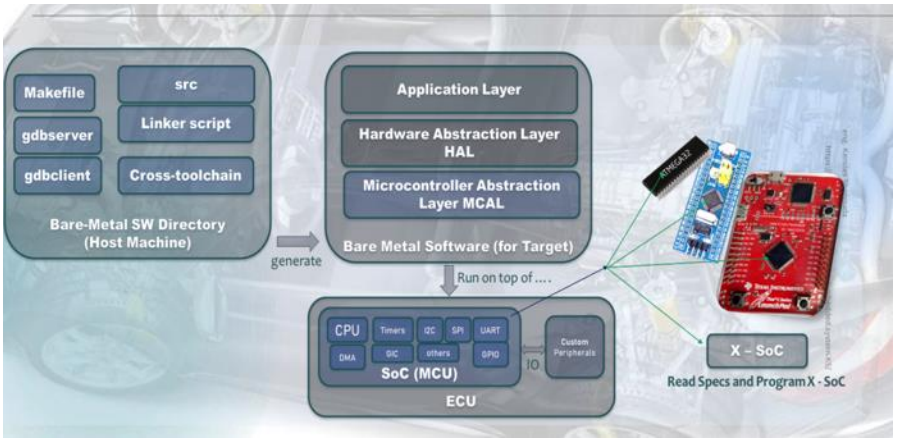


Fig 7.1. SW Layers

7.2 Microcontroller Abstraction Layer (MCAL)

This is the lower layer of the microcontroller. This layer has all the drivers which can connect with the ECU (Electrical Control Unit) to the Upper layer is HAL. It consists of MCU memories, buses, and peripherals.

7.2.1 RCC Driver

The RCC (Reset and Clock Control) peripheral in STM32 microcontrollers plays a vital role in configuring and controlling the system clocks, which are essential for the operation of the entire microcontroller and its peripherals.

Functionality of RCC:

- Clock Generation: The RCC peripheral generates and distributes clock signals to various components of the microcontroller, including the CPU, peripherals, and external interfaces. These clock signals synchronize the operation of different components, ensuring proper timing and functionality.
- Power Management: RCC allows for dynamic adjustment of clock frequencies and power modes, enabling efficient power management. By controlling clock sources and frequencies, RCC helps optimize power consumption based on the system's operational requirements, thereby extending battery life in battery-powered applications like the self-parking car.
- Peripheral Configuration: RCC configures clock settings for individual peripherals, enabling them to operate at desired frequencies and synchronization with other system components. Proper configuration of peripheral clocks ensures reliable communication and functionality of peripherals, such as timers, UART, SPI, and I2C interfaces.
- System Stability: RCC ensures the stability and reliability of the entire system by providing accurate and consistent clock signals. Proper clock configuration prevents timing errors, data corruption, and system crashes, thereby enhancing system stability and performance.

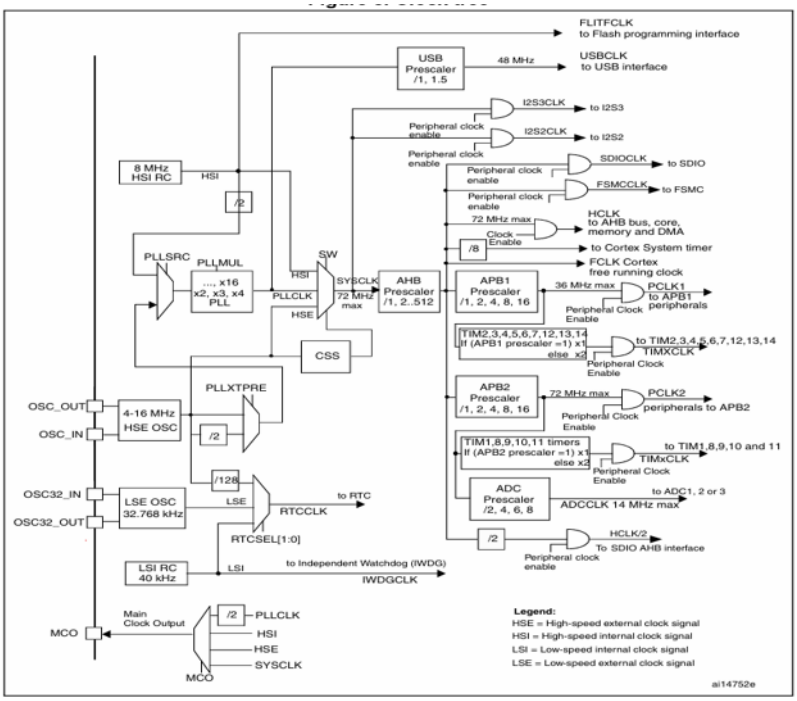


Fig 7.2. STM32F1xx Clock Tree

Usage in the Project:

- Clock Configuration: RCC initializes and configures the system clock settings during the microcontroller's startup. This includes selecting the clock source (e.g., internal oscillator, external crystal), setting the PLL (Phase-Locked Loop) parameters for clock multiplication, and configuring prescalers/dividers to derive desired clock frequencies.
- Peripheral Clock Configuration: RCC configures individual peripheral clocks to enable their operation at specific frequencies. For example, RCC configures the clock for UART communication, timer peripherals for timing functions, and GPIO ports for digital I/O operations.
- Power Management: RCC manages power modes and clock frequencies based on the system's operational requirements and power-saving strategies. For instance, RCC adjusts clock frequencies and power modes during idle or low-power states to minimize energy consumption while maintaining essential functionality.
- Clock Monitoring: RCC monitors the integrity of clock sources and detects clock failures using features like the Clock Security System (CSS). In the self-parking car project, RCC ensures the reliability of clock signals critical for system operation, preventing potential timing errors or system malfunctions.

- System Stability: RCC contributes to the overall stability and reliability of the self-parking car system by providing accurate and consistent clock signals. Proper clock configuration and monitoring help maintain system integrity, prevent data corruption, and ensure smooth operation of the microcontroller and its peripherals.

```
/*******************************************************************************
*                                  APIs                                        *
*******************************************************************************/
// function to enable MCU Peripherals
void MCAL_RCC_Peripherals_enable(RCC_Buses_t bus , uint16_t peripheral , uint8_t status);
// function to Reset MCU Peripherals
void MCAL_RCC_Peripherals_Reset(RCC_Buses_t bus , uint16_t peripheral , uint8_t status);
//this function is to set system bus frequency
void MCAL_RCC_GetSYS_CLCKFreq(RCC_speed_t RCC_Speed);
//this function is to set AHB bus frequency
uint32_t MCAL_RCC_GetHCLKFreq(void);
//this function is to set ABP1 bus frequency
uint32_t MCAL_RCC_GetPCLK1Freq(void);
//this function is to set ABP2 bus frequency
uint32_t MCAL_RCC_GetPCLK2Freq(void);
//this function is to set ADC frequency
uint32_t MCAL_RCC_GetPCLK2_ADCFreq(void);

```

Fig 7.3. RCC Driver APIs

7.2.2 General Purpose I/O Driver

General Purpose I/O (GPIO) pins are fundamental components of microcontrollers like the STM32 series, providing a versatile interface for interacting with external devices and peripherals.

Functionality of GPIO:

- Versatility: GPIO pins can be configured as either input or output, allowing them to interface with a wide range of external devices, such as sensors, actuators, LEDs, switches, and displays. This versatility enables the microcontroller to interact with its environment and control various system components.
- Flexibility: GPIO pins offer flexibility in hardware configuration and system design. Developers can dynamically assign GPIO pins to different functionalities based on the project's requirements, adapting the system's behavior as needed without requiring hardware modifications.
- Integration: GPIO pins seamlessly integrate with the microcontroller's peripheral ecosystem, enabling communication and coordination between different hardware components. They serve as the interface through which the microcontroller interacts with the external world, facilitating data exchange, signal processing, and system control.

- Low-Level Control: GPIO pins provide low-level control over digital signals, allowing precise manipulation of voltage levels and signal states. This level of control is essential for tasks such as sensor reading, actuator control, signal conditioning, and communication protocol implementation.

Usage in the Project:

- Sensor Interface: GPIO pins can be used to interface with sensors for detecting environmental parameters relevant to the self-parking car project, such as distance sensors for obstacle detection, light sensors for ambient light measurement, or temperature sensors for environmental monitoring.
- Actuator Control: GPIO pins can control actuators responsible for vehicle movement and system operation. For example, GPIO pins can drive motor driver modules to control motor speed and direction, activate solenoids for door locking mechanisms, or trigger relays for switching external devices.
- User Interface: GPIO pins can interface with user input devices, such as push buttons, switches, or touch sensors, to provide user interaction capabilities in the self-parking car project. These inputs can be used to initiate parking maneuvers, adjust system settings, or trigger emergency stops.
- Interrupt Handling: GPIO pins can generate interrupts to signal external events requiring immediate attention. This feature is useful for handling time-critical tasks, such as emergency stops triggered by user input or obstacle detection events requiring prompt reaction.

```
/*******************************************************************************
*                                   APIs                                       *
*******************************************************************************/

//Initialization of GPIOx PINy according to a specific configuration
void MCAL_GPIO_INIT(GPIO_TypeDef* GPIOx,uint16_t pin,uint32_t pinmode);//port , pin , mode
//De-initialization of GPIOx PINy
void MCAL_GPIO_DEINIT(GPIO_TypeDef* GPIOx);
//Write on pin
void MCAL_GPIO_WRITEPIN(GPIO_TypeDef* GPIOx,uint16_t pin,uint8_t status);
//Write on port
void MCAL_GPIO_WRITEPORT(GPIO_TypeDef* GPIOx,uint16_t status);
//Values of a port
void set_Value_PORT(GPIO_TypeDef* GPIOx,uint16_t status,uint16_t position);
void res_Value_PORT(GPIO_TypeDef* GPIOx,uint16_t status,uint16_t position);
//Read from pin
uint8_t MCAL_GPIO_READPIN(GPIO_TypeDef* GPIOx, uint16_t pin);
//Read from port
uint16_t MCAL_GPIO_READPORT(GPIO_TypeDef* GPIOx);
//Toggle pin
void MCAL_GPIO_TOGGLEPIN(GPIO_TypeDef* GPIOx, uint16_t pin);

//Toggle port
void MCAL_GPIO_TOGGLEPORT(GPIO_TypeDef* GPIOx);
void TogglePin(GPIO_TypeDef* GPIOx, uint16_t pin);

//Lock pin
uint8_t MCAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t pin_number);
```

Fig 7.4. GPIO Driver APIs

7.2.3 External Interrupts Driver

External interrupts, also known as EXTI (External Interrupt) in STM32 microcontrollers, are a crucial feature for embedded systems that need to respond to external events in real-time. Here's how you can expand on this topic within the context of your project: External interrupts play a crucial role in embedded systems like the self-parking car project, enabling real-time responsiveness to external events and improving system efficiency.

Functionality of External Interrupts:

- Real-Time Event Handling: External interrupts allow the system to respond promptly to external events, such as sensor inputs or user interactions. This real-time responsiveness is critical for tasks like obstacle detection, vehicle navigation, and user interaction in the self-parking car project.
- Interrupt-Driven Architecture: By utilizing interrupts, the system can conserve CPU resources and power consumption by entering low-power states when idle. Interrupt-driven architecture enables efficient multitasking and event-driven programming, enhancing system performance and responsiveness.
- Simplified Control Flow: External interrupts facilitate a streamlined control flow, allowing the system to focus on critical tasks while waiting for external events to occur. This simplifies the software design and improves code readability, maintainability, and scalability.

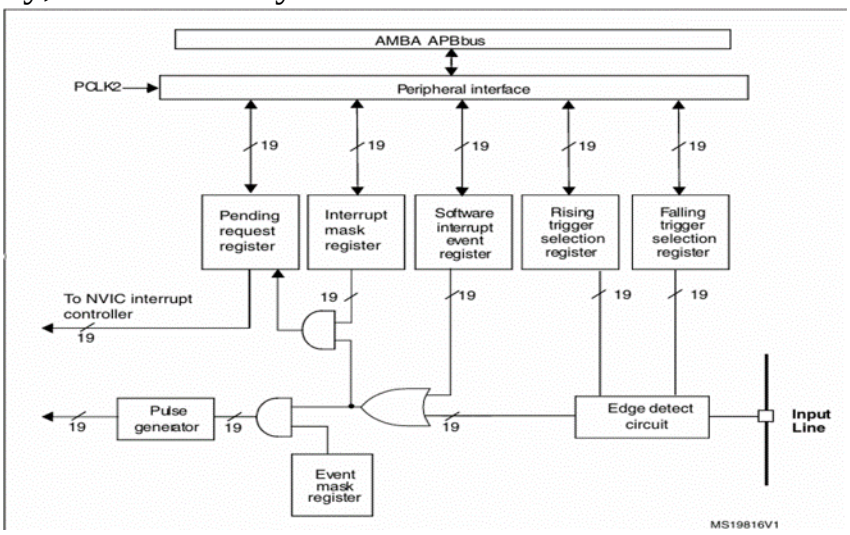


Fig 7.5. STM32F1xx EXTI Block Diagram

Usage in the Project:

- Sensor Input Handling: External interrupts can be employed to handle inputs from sensors, such as ultrasonic sensors for obstacle detection or proximity sensors for

parking space detection. When a sensor detects an event (e.g., obstacle detected), it triggers an external interrupt, allowing the system to respond immediately.

- User Interaction: External interrupts can also be used to handle user interactions, such as button presses or touch screen inputs. When a user interacts with the interface (e.g., pressing a button to initiate parking), it generates an external interrupt, triggering the corresponding action in the system.
- Interrupt Service Routines (ISRs): In the self-parking car project, ISRs are implemented to handle external interrupts. These ISRs are short, efficient routines designed to respond quickly to the interrupt event, perform the necessary actions (e.g., updating system state, initiating maneuvers), and return control to the main program flow.
- Interrupt Prioritization: Depending on the project's requirements, external interrupts may need to be prioritized to ensure that critical events are handled promptly. Prioritization mechanisms, such as interrupt nesting or priority levels, can be implemented to manage multiple external interrupts effectively.

```
62
63 //************************
64  //API   == == == == ==
65 //************************
66
67 void EXTI_Init (GPIO_TypeDef* PORTx ,uint16_t pin,uint16_t triggercase,void (*function_addres) (void));
68 void MCAL_EXTI_GPIO_DEINIT(void) ;
69
```

Fig 7.6. EXTI Driver APIs

7.2.4 Timer Driver

Timers are crucial components in embedded systems like the self-parking car project, providing accurate timing and scheduling capabilities essential for various tasks such as sensor reading, actuator control, and system synchronization.

Functionality of Timers:

- Precision Timing: Timers generate accurate time intervals and timestamps, enabling precise control of system events and operations. This precision is essential for tasks such as sensor sampling, motor control, and synchronization of multiple system components in the self-parking car project.
- Event Generation: Timers can be configured to generate periodic or one-shot events, triggering actions or interrupts at predetermined intervals. These events

facilitate real-time response to external stimuli, such as sensor inputs or user commands, enhancing system responsiveness and efficiency.

- PWM Generation: Timers with pulse-width modulation (PWM) capability can generate variable-width pulses, enabling precise control of motor speed, LED brightness, and servo position in the self-parking car. PWM signals are widely used for analog output and motor control applications.
- Interrupt Generation: Timers can generate interrupts at specified intervals or when certain conditions are met, allowing the microcontroller to perform background tasks or handle time-critical events without continuous CPU polling. Interrupt-driven timer handling improves system efficiency and responsiveness, conserving CPU resources for other tasks.

Usage in the Project:

- Sensor Sampling: Timers trigger periodic sensor readings at fixed intervals, allowing the self-parking car to continuously monitor its surroundings for obstacles, distance measurements, and environmental conditions. This enables real-time decision-making and navigation during parking maneuvers.
- Actuator Control: Timers generate PWM signals to control motor speed, direction, and torque during vehicle movement and parking operations. By adjusting PWM parameters, such as duty cycle and frequency, the system can achieve precise control over motorized components, such as wheels, steering mechanisms, and braking systems.
- System Synchronization: Timers synchronize the operation of multiple system components, such as sensors, actuators, and communication interfaces, to ensure coordinated behavior and timing. Synchronized operation enhances system stability, reliability, and efficiency, especially in complex autonomous systems like the self-parking car.
- Event Timing and Logging: Timers provide accurate timestamps for event timing and logging, allowing the self-parking car to record and analyze system behavior, sensor data, and user interactions. Time stamped data facilitates debugging, performance analysis, and optimization of parking algorithms and control strategies.

```
46
47 /*****************************************************************************
48 *                                APIs                                       *
49 *****************************************************************************/
50
51 //timer3 ch_1 --> A6  <> timer3 ch_2 --> A7 <> timer3 ch_3 --> B0 <> timer3 ch_4 --> B1
52 void MCAL_PWM_Init(PWM_Modes_t mode ,TIMER_channels_t channel , double duty_cycle , uint32_t freq,uint32_t TIM_PSC);
53
54 //Enter in while loop and after timer interrupt it break the loop
55 void delay(uint16_t time,uint8_t U,uint32_t clk);
56
57 uint32_t TIME_CALCULATION(uint32_t clk,uint8_t TIMER_ST);
58
59 uint32_t MCAL_COUNTER(uint32_t clk,uint8_t TIMER_ST);
60
```

Fig 7.7. Timer Driver APIs

7.2.5 Watch Dog Timer Driver

The Watchdog Timer (WDT) serves as a crucial safety mechanism in embedded systems, including the self-parking car project, enhancing its reliability and fault tolerance.

Functionality of WDT:

- System Reliability: The WDT continually monitors the system's operation and detects potential software faults or hang-ups. It ensures that the system remains responsive and functional, even in the presence of unexpected errors.
- Fault Tolerance: By providing a fail-safe mechanism, the WDT helps prevent system crashes or freezes caused by software bugs, glitches, or external interference. It facilitates graceful recovery and maintains the system's operational integrity.
- Safety Assurance: In safety-critical applications like the self-parking car, where human lives and property are at stake, the WDT plays a pivotal role in ensuring the vehicle's safe operation. It helps mitigate risks associated with software failures and ensures the system behaves predictably.

Usage in the Project:

- Initialization and Configuration: The WDT should be initialized and configured appropriately during the system startup. This involves setting parameters such as the timeout period, operating mode (e.g., windowed or independent watchdog), and interrupt/callback handlers.
- Periodic Refresh: The WDT must be periodically refreshed or "fed" before its timeout period elapses. This prevents the WDT from triggering a reset,

indicating that the system is functioning correctly. The periodic refresh can be achieved using dedicated WDT driver APIs or software routines.

- Fault Handling and Recovery: In the event of a software fault or system hang-up, where the WDT is not refreshed within the specified timeout period, the WDT triggers a reset, initiating system recovery. Proper fault handling mechanisms should be implemented to diagnose the cause of the fault and take appropriate recovery actions, such as resetting the system, logging diagnostic information, or entering a safe state.

```
30 void IWDG_set(void);
31 //independent watch dog timer disable func.
32 void IWDG_RESET(void);
33
34 /*
35  *pr:
36  *  0: divider /4   : tick_time : 0.1ms  max_time : 409.6ms
37     1: divider /8   : tick_time : 0.2ms  max_time : 819.2ms
38     2: divider /16  : tick_time : 0.4ms  max_time : 1638.4ms
39     3: divider /32  : tick_time : 0.8ms  max_time : 3276.8ms
40     4: divider /64  : tick_time : 1.6ms  max_time : 6553.6ms
41     5: divider /128 : tick_time : 3.2ms  max_time : 13107.2ms
42     6: divider /256 : tick_time : 6.4ms  max_time :26214.4ms
43  */
44 //independent watch dog timer time set func.
45 void IWDG_time_set(uint32_t pr, uint32_t time);
46
```

Fig 7.8. WDG Driver APIs

7.2.6 Universal Asynchronous Receiver and Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) communication protocol is fundamental in embedded systems like the self-parking car project, facilitating serial communication between the microcontroller and external devices.

Importance of UART:

- Versatile Communication: UART enables bi-directional serial communication between the microcontroller and external devices, such as sensors, actuators, GPS modules, and communication modules (e.g., Bluetooth, Wi-Fi). This versatility allows for seamless integration of various peripherals and enables data exchange in real-time.
- Simplicity and Ubiquity: UART is widely supported and easy to implement, making it a common choice for communication between microcontrollers and peripherals. Its simplicity reduces hardware complexity and software overhead, facilitating rapid development and prototyping in projects like the self-parking car.

- Point-to-Point Communication: UART supports point-to-point communication, allowing the microcontroller to communicate directly with individual devices without requiring complex network configurations or addressing schemes. This simplicity makes UART suitable for applications where direct device-to-device communication is sufficient.
- Robustness: UART communication is relatively robust and immune to noise and interference, making it suitable for use in noisy environments or applications where signal integrity is essential. Error detection and correction mechanisms, such as parity checking and framing, can be implemented to enhance communication reliability.
- Real-Time Data Transfer: UART enables real-time data transfer at configurable baud rates, allowing the microcontroller to exchange data with external devices quickly and efficiently. This real-time capability is crucial for applications requiring rapid response times, such as sensor data acquisition or control signal transmission.

Usage in the Project:

- Sensor Data Acquisition: UART is used to receive data from sensors, such as GPS modules, IMUs (Inertial Measurement Units), and environmental sensors, providing real-time information about the vehicle's position, orientation, and surroundings. This data is essential for autonomous navigation and obstacle detection in the self-parking car.
- Actuator Control: UART can transmit control signals to actuators or peripheral devices, such as motor controllers, servo controllers, or LED drivers, enabling precise control of vehicle movement, steering, and signaling. This allows the microcontroller to execute parking maneuvers and adjust vehicle behavior based on sensor inputs.
- Wireless Communication: UART interfaces with communication modules, such as Bluetooth or Wi-Fi modules, to enable wireless communication between the self-parking car and external devices, such as smartphones or remote-control units. This wireless connectivity enhances user interaction, system monitoring, and remote operation capabilities.
- Debugging and Diagnostics: UART can be used for debugging and diagnostics by transmitting debug messages, status updates, and error codes to a connected terminal or debugging tool. This allows developers to monitor system behavior, diagnose issues, and perform troubleshooting during development and testing phases.

```
//Initialization of USARTx according to a specific configuration
void MCAL_USART_INIT(USART_TypeDef* USARTx, USART_Config* USART_ConFig);
//DeInitialization of USARTx according to a specific configuration
void MCAL_USART_DEINIT(USART_TypeDef* USARTx);
//Send data
void MCAL_USART_SEND_DATA(USART_TypeDef* USARTx, uint16_t* Buffer, enum Polling_Mechanism PollingEN);
//Read data
void MCAL_USART_READ_DATA(USART_TypeDef* USARTx, uint16_t* Buffer, enum Polling_Mechanism PollingEN);
//Wait
void MCAL_UART_WAIT_TC(USART_TypeDef* USARTx);
//Set pins with its considerations to work
void MCAL_USART_SETPIN(USART_TypeDef* USARTx);
```

Fig 7.9. UART Driver APIs

7.2.7 Inter-Integrated Circuit Driver

The Inter-Integrated Circuit (I2C) communication protocol is integral to many embedded systems, including the self-parking car project, due to its simplicity, versatility, and efficiency in connecting multiple devices on the same bus.

Functionality of I2C:

- Multi-device Communication: I2C enables communication between multiple devices (such as sensors, actuators, displays, and microcontrollers) connected on the same bus. This allows for seamless data exchange and coordination between various system components in the self-parking car project.
- Simple Protocol: The I2C protocol is relatively simple, consisting of only two wires (SCL for clock and SDA for data), making it easy to implement and integrate into embedded systems. Its simplicity reduces hardware complexity and facilitates rapid development and prototyping.
- Efficient Data Transfer: I2C supports efficient serial data transfer, allowing for bidirectional communication at variable speeds. This enables the exchange of data between devices with minimal overhead, making it suitable for real-time applications like obstacle detection, distance measurement, and navigation in the self-parking car.
- Addressable Communication: Each device connected to the I2C bus has a unique address, allowing the microcontroller to selectively communicate with specific devices. This addressability enables efficient data routing and management, enhancing system scalability and flexibility.

- Bus Arbitration: I2C incorporates bus arbitration mechanisms to manage simultaneous data transfers and prevent data collisions. This ensures reliable communication between devices, even in scenarios where multiple devices attempt to communicate simultaneously.

Usage in the Project:

- Display Interfaces: I2C interfaces with display modules, such as OLED or LCD screens, to provide visual feedback and user interface elements in the self-parking car. These displays can convey information about parking status, sensor readings, navigation instructions, and system alerts to the user.
- Configuration and Calibration: I2C facilitates configuration and calibration of sensors and actuators, allowing dynamic adjustment of parameters such as sensor sensitivity, motor speed, or steering angle. This enables fine-tuning of system behavior and performance to optimize parking efficiency and accuracy.
- Peripheral Expansion: I2C supports daisy-chaining and bus expansion, allowing for the addition of new sensors or devices to the system without requiring additional hardware interfaces. This scalability enables future enhancements and upgrades to the self-parking car project without major hardware modifications.

```
167 * ===============================================
168 * APIs Supported by "MCAL I2C DRIVER"
169 * ===============================================
170 */
171
172 //Initialization of I2Cx according to a specific configuration
173 void MCAL_I2C_Init(I2C_TypeDef *I2Cx , I2C_Config_t *I2C_Config);
174 //DeInitialization of I2Cx according to a specific configuration
175 void MCAL_I2C_DeInit(I2C_TypeDef *I2Cx);
176 //Set pins with its considerations to work
177 void MCAL_I2C_Set_Pins(I2C_TypeDef *I2Cx);
178 //Master
179 void MCAL_I2C_Master_TX(I2C_TypeDef *I2Cx, uint16_t DevAddr ,uint8_t* dataOut , uint32_t datalen , Stop_Condition Stop , Repeated_Start Start);
180 void MCAL_I2C_Master_RX(I2C_TypeDef *I2Cx, uint16_t DevAddr ,uint8_t* dataOut , uint32_t datalen , Stop_Condition Stop , Repeated_Start Start);
181 //Slave
182 void MCAL_I2C_SlaveSendData(I2C_TypeDef *I2Cx , uint8_t data);
183 uint8_t MCAL_I2C_SlaveReceiveData(I2C_TypeDef *I2Cx);
184
185 /*Generic Functions*/
186 void I2C_GenerateSTART(I2C_TypeDef *I2Cx, Fuctional_State NewState, Repeated_Start Start);
187 void I2C_GenerateSTOP(I2C_TypeDef *I2Cx, Fuctional_State NewState);
188 void I2C_SendAddress(I2C_TypeDef *I2Cx, uint16_t Address, I2C_Direction Direction);
189 FlagSTATUS I2C_GetFlagSTATUS(I2C_TypeDef *I2Cx, Status Flag);
190 void I2C_AcknowledgeConfig(I2C_TypeDef *I2Cx, Fuctional_State NewState);
191 void Slave_States(I2C TypeDef *I2Cx,Slave State State);
```

Fig 7.10. I2C Driver APIs

7.3 Hardware Abstraction Layer (HAL)

This layer used the MCAL driver to be interfacing the microcontroller with the output section (motor, servo…. etc.).

7.3.1 Motor Driver

This driver helps us in the Application layer to interfacing the microcontroller with the motor driver.

Functionality of motor driver:

- This motor driver inputs the signal from the microcontroller to start (forward, reverse) or stop the motor.

```
/*
 * ================================================
 * APIs Supported by "HAL DC Motor DRIVER"
 * ================================================
 */
void HAL_MOTOR_Init(void);
void HAL_MOTOR_Motion(uint8_t x);
```

Fig 7.11. DC Motor Driver APIs

7.3.2 Servo Motor Driver

This driver helps us in the Application layer to interface the microcontroller with the servo.

Functionality of servo motor driver:

- The servo talks specific PWM from microcontroller to move the servo to the accurate angle.

```
/*******************************************************************************
*                                APIs                                          *
*******************************************************************************/

void HAL_Servo_Set_Angle(double angle,uint32_t clock);
```

Fig 7.12. Servo Driver APIs

7.3.3 Ultrasonic Sensor Driver

The ultrasonic sensor driver is crucial in the self-parking car project as it enables the vehicle to detect obstacles and navigate safely in its environment.

Functionality of Ultrasonic Sensor Driver:

- Obstacle Detection: Ultrasonic sensors use sound waves to detect objects in the vehicle's path, providing crucial information for obstacle detection and avoidance.

The driver facilitates the integration of ultrasonic sensors into the microcontroller-based system, enabling real-time monitoring of the vehicle's surroundings.

- Safety: By detecting obstacles such as walls, vehicles, or pedestrians, the ultrasonic sensor driver enhances the safety of the self-parking car, preventing collisions and accidents. This safety feature is essential for autonomous vehicles operating in dynamic environments where unexpected obstacles may appear.
- Navigation: Ultrasonic sensors help the vehicle navigate complex environments, such as parking lots or narrow streets, by providing distance measurements to nearby objects. The driver processes these measurements to determine safe navigation paths and execute parking maneuvers with precision.
- Parking Assistance: Ultrasonic sensors assist the driver during parking maneuvers by providing feedback on the distance between the vehicle and surrounding objects. The driver interprets sensor data to guide the vehicle into parking spaces accurately, reducing the risk of collisions and optimizing parking efficiency.
- User Experience: The integration of ultrasonic sensors enhances the user experience by providing visual or auditory feedback to the driver, indicating the proximity of obstacles and guiding parking maneuvers. This feedback improves driver confidence and comfort, especially in challenging parking scenarios.

Usage in the Project:

- Sensor Initialization: The ultrasonic sensor driver initializes the sensor hardware, configuring parameters such as operating mode, measurement range, and communication interface. This initialization process prepares the sensor for operation within the self-parking car system.
- Data Acquisition: The driver reads distance measurements from the ultrasonic sensor at regular intervals, capturing real-time information about nearby obstacles. These measurements are processed to determine the vehicle's surroundings and detect potential collision hazards.
- Data Processing: Distance measurements obtained from the ultrasonic sensor are processed and filtered to remove noise and inaccuracies, ensuring reliable obstacle detection. Signal processing algorithms may be employed to enhance the accuracy and stability of sensor readings.
- Obstacle Detection: The driver analyzes distance measurements to detect obstacles within the vehicle's path, identifying their size, position, and proximity. This information is used to assess collision risks and inform navigation decisions during parking maneuvers.

- Collision Avoidance: Based on obstacle detection results, the driver triggers collision avoidance mechanisms to steer the vehicle away from potential hazards or stop its movement if necessary. These safety features prevent accidents and ensure the vehicle's safe operation in congested or confined spaces.

```
48
49 //============================================================================
50
51 /*****************************************************************************
52 *                              APIs                                          *
53 *****************************************************************************/
54
55 void HAL_ULTRASONIC_INIT(ULT_SLEC_T num);
56 uint32_t HAL_ULTRASONIC_GET_DISTANCE(ULT_SLEC_T num);
57
```

Fig 7.13. Ultrasonic Driver APIs

7.4 Real-Time Operating System (RTOS)

RTOS (Real-Time Operating System) plays a significant role in embedded systems like the self-parking car project by providing multitasking, scheduling, and resource management capabilities necessary for handling concurrent tasks and meeting real-time requirements.

Functionality of RTOS:

- Multitasking: RTOS enables multitasking by allowing the system to run multiple tasks or threads concurrently. This is crucial for handling diverse activities simultaneously, such as sensor data acquisition, motor control, user interface interaction, and communication with external devices.
- Task Scheduling: RTOS implements task scheduling algorithms to prioritize and allocate CPU time to different tasks based on their priority levels, deadlines, and resource requirements. This ensures that critical tasks are executed in a timely manner, meeting real-time constraints and maintaining system responsiveness.
- Resource Management: RTOS provides mechanisms for managing system resources, such as memory, CPU, and peripherals, efficiently among concurrent tasks. This prevents resource conflicts, deadlock situations, and priority inversion problems, ensuring reliable operation and system stability.
- Real-Time Responsiveness: RTOS guarantees predictable and deterministic response times for critical tasks, allowing the system to meet stringent real-time requirements. This is essential for applications like autonomous navigation and

obstacle detection, where timely sensor readings and actuator control are paramount.

- Modularity and Scalability: RTOS facilitates modular and scalable system design by organizing functionality into independent tasks or modules. This modular architecture simplifies development, debugging, and maintenance, allowing for easy integration of new features and future enhancements.

Usage in the Project:

- Task Management: RTOS divides the self-parking car project into multiple tasks or threads, each responsible for a specific function or subsystem. Tasks may include sensor reading, motor control, navigation algorithm, user interface handling, and communication with external devices.
- Task Prioritization: RTOS assigns priority levels to tasks based on their importance and real-time requirements. Critical tasks, such as obstacle detection and collision avoidance, are assigned higher priorities to ensure timely execution and prevent system failures.
- Synchronization and Communication: RTOS provides synchronization mechanisms, such as semaphores, mutexes, and message queues, to facilitate communication and data sharing between tasks. This allows for coordinated operation and exchange of information among different system components.
- Interrupt Handling: RTOS handles interrupts generated by hardware peripherals, sensors, and user inputs, ensuring timely response and interrupt servicing. Interrupt-driven processing minimizes latency and allows the system to react promptly to external events.
- Resource Management: RTOS manages system resources, including memory, CPU time, and peripherals, to prevent resource contention and optimize resource utilization. Resource allocation algorithms ensure fair access to resources and prevent starvation or deadlock situations.
- Real-Time Constraints: RTOS guarantees deterministic task execution and meets real-time constraints imposed by the self-parking car application. By providing predictable response times and scheduling mechanisms, RTOS ensures that critical tasks are executed within specified deadlines, enhancing system reliability and safety.

7.5 Application Layer

The Application Layer is a critical component in the self-parking car project as it encompasses the high-level logic, algorithms, and user interaction functionalities necessary for autonomous parking operations.

Functionality of the layer:

- Algorithm Implementation: The Application Layer implements algorithms for obstacle detection, localization, mapping, path planning, and trajectory generation to enable autonomous parking. These algorithms utilize sensor data, environmental information, and vehicle dynamics to make informed decisions and navigate the vehicle safely to its parking spot.
- User Interface Development: The Application Layer designs and develops user interfaces, such as touch screens, displays, or mobile apps, to interact with the self-parking car system. These interfaces allow users to initiate parking maneuvers, monitor system status, and adjust parking parameters according to their preferences.
- Integration with Sensor and Actuator Drivers: The Application Layer integrates with lower-level components, such as sensor drivers, actuator controllers, and communication interfaces, to access sensor data, control vehicle movements, and exchange information with external devices. It abstracts the hardware details and provides a unified interface for higher-level functionalities.
- State Machine Implementation: The Application Layer implements state machines to model the different states and transitions in the parking process, such as initialization, obstacle detection, trajectory planning, and parking execution. State machines provide a structured approach to managing the system's behavior and coordinating task execution.
- Fault Detection and Recovery: The Application Layer monitors system health, detects faults, and initiates recovery actions when anomalies are detected. It implements error handling mechanisms, such as exception handling, fault tolerance, and error recovery strategies, to maintain system integrity and robustness.
- Performance Optimization: The Application Layer optimizes performance by fine-tuning algorithms, adjusting parameters, and optimizing resource usage to enhance system efficiency, responsiveness, and reliability. Performance metrics, such as processing time, energy consumption, and accuracy, are monitored and optimized to meet project requirements.

7.6 Summary

Software is the most important part and is the controlling mind of the project. Without it, it would be like a model without benefiting from it. It determines the function of each part, divides the tasks, makes decisions, and reaches the goal.

In MCAL Layer, the RCC peripheral is essential for configuring and controlling system clocks in STM32 microcontrollers by managing clock generation, power management, peripheral configuration, and clock monitoring, RCC ensures the stability, reliability, and performance of the microcontroller-based system. The GPIO pins are essential components in the self-parking car project, providing a versatile interface for interfacing with sensors, controlling actuators, facilitating user interaction, and signaling system status. By leveraging GPIO pins effectively, the microcontroller can interact with its environment, control system components, and provide a seamless user experience. External interrupts are essential for enabling real-time event handling, efficient multitasking, and simplified control flow in the self-parking car project. By leveraging external interrupts to respond promptly to sensor inputs and user interactions, the system can achieve enhanced responsiveness, reliability, and user experience. Timers play a vital role in the self-parking car project, providing precise timing, event generation, PWM control, and system synchronization capabilities essential for autonomous navigation, sensor integration, and actuator control. By leveraging timers effectively, the project can achieve robust functionality, real-time responsiveness, and efficient operation. communication protocols such as UART and I2C, configured and used for debugging and sending or receiving data from sensors and actuators.

In HAL Layer, the ultrasonic sensor driver is essential for obstacle detection, safety, navigation, parking assistance, and user experience enhancement in the self-parking car project. By integrating ultrasonic sensors into the system and leveraging the driver effectively, the project can achieve reliable obstacle detection, precise navigation, and safe parking maneuvers, ultimately contributing to its success in autonomous parking applications.

RTOS is essential for managing multitasking, scheduling, and resource allocation in the self-parking car project. By leveraging RTOS effectively, the project can achieve real-time responsiveness, modularity, scalability, and reliability, ultimately contributing to its success in autonomous parking applications.

The Application Layer serves as the brain of the self-parking car project, responsible for implementing high-level logic, user interaction, decision making, error handling, and system integration functionalities. By leveraging the capabilities of the Application Layer effectively, the project can achieve autonomous parking capabilities, user-friendly interfaces, fault tolerance, and seamless integration with external systems, ultimately delivering a safe, efficient, and user-friendly self-parking car solution.

Chapter 8

Experimental Methods and Results

8.1 Overview

In this chapter, we will talk about an important part and stage of completing any project, which is testing and approval, as it cannot be said that any product fulfills its purpose unless it is tested and known whether it has defects or is free of them. We will first discuss the ways and methods used for testing and how to synchronize them with the development and analysis of the project, how to test and ensure the readiness of each of the software, components, and the conceptual path for parking the car through special programs for that, and the actual testing of them. The preliminary results and final results will be explained, including how to detect and address errors. In the end, a summary of all the results and analyzes achieved will be presented.

Verification and Validation Methodology, Software testing is the process of validating and verifying that a SW program/app meets the requirements, works as expected, and can be implemented with the same characteristics.

- Error: refers to difference between actual output and expected output.
- Faults: it's condition that causes the SW to fail to perform its function. Also known as defect/bug.
- Failure: it's the inability of a system/component to perform its required function.

Verification is the SW should conform to its specification. "Does it work?". Validation is the SW should do what the user really requires. "Did we build the right thing?".

Verification & Validation must be applied at each stage in the SW process. SW testing is a way to increase the quality and reduce the risk of failure. SW testing is a process which includes many different activities, not equal only test execution.

SW Development Life Cycle (SDLC), It's the sequence of activities carried out by developers to design and develop high quality SW.

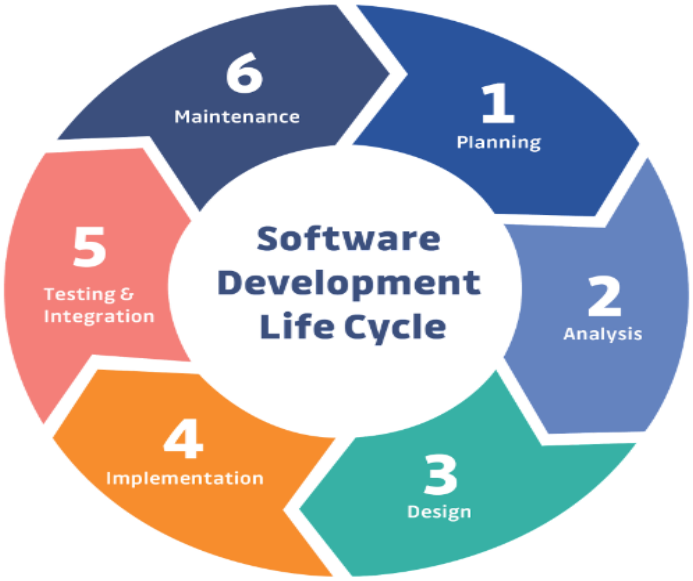


Fig 8.1. software development life cycle

Software Testing Life Cycle (STLC) consists of a series of activities carried out methodically by Testers methodologically to test your software product. The software part had the largest part of testing and verification, and this was done by testing each drive for each microcontroller feature, sensors, and motors by using Keil Micro Vision program and physical components.

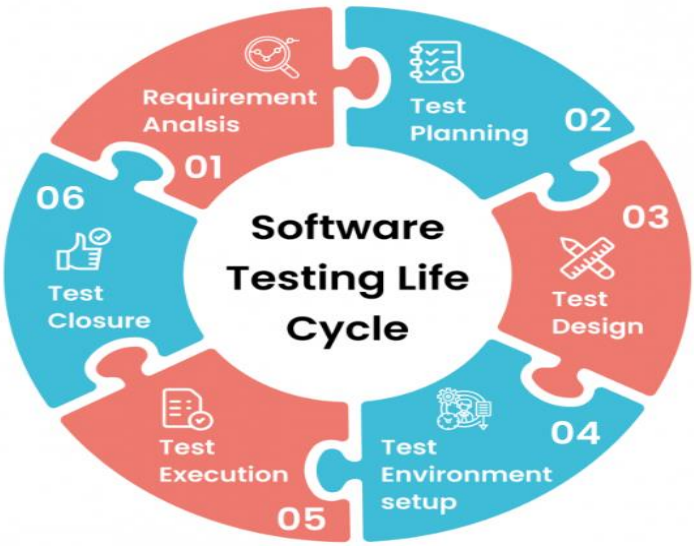


Fig 8.2. software testing life cycle

We have adopted a specific approach in this part, which is Upon completion of a certain part (studying, implementing, and testing), one moves to the next part. It's called Waterfall Model.

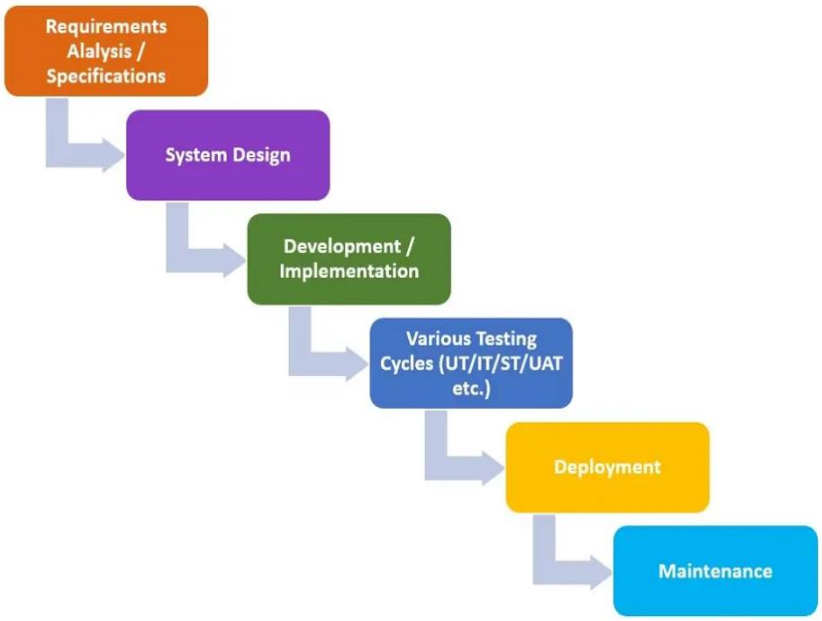


Fig 8.3. Waterfall model

8.2 Software Experimental Procedure and Results

At this point we start to test Microcontroller Abstraction Layer (MCAL); that includes MCU features. then Hardware Abstraction Layer (HAL); that include ultrasonic sensors and motors. We used Keiluvision5 and the physical components for testing.

- MCU Features and Peripherals (RCC, GPIO, Interrupts, timers, ...etc).
- Ultrasonic Sensor
- DC motors
- Servo motor

8.2.1 Test Scenarios and Variations

Firstly, Microcontroller Abstraction Layer (MCAL) testing has begun:

- ✓ Started with RCC (Reset and Clock Controller) driver; Our goals are: an easy way to enable and reset each peripheral, control the frequency of each bus, and provide more than one frequency for our MCU. When we tested the first version, we succeeded in achieving the first and second goals, but then our MCU was operating at the frequency of 8MHZ, so in the second version we were focusing on increasing the flexibility of the frequencies available to our system and in the second version we reached the ability to provide three frequencies 8, 32 and 36 MHz.

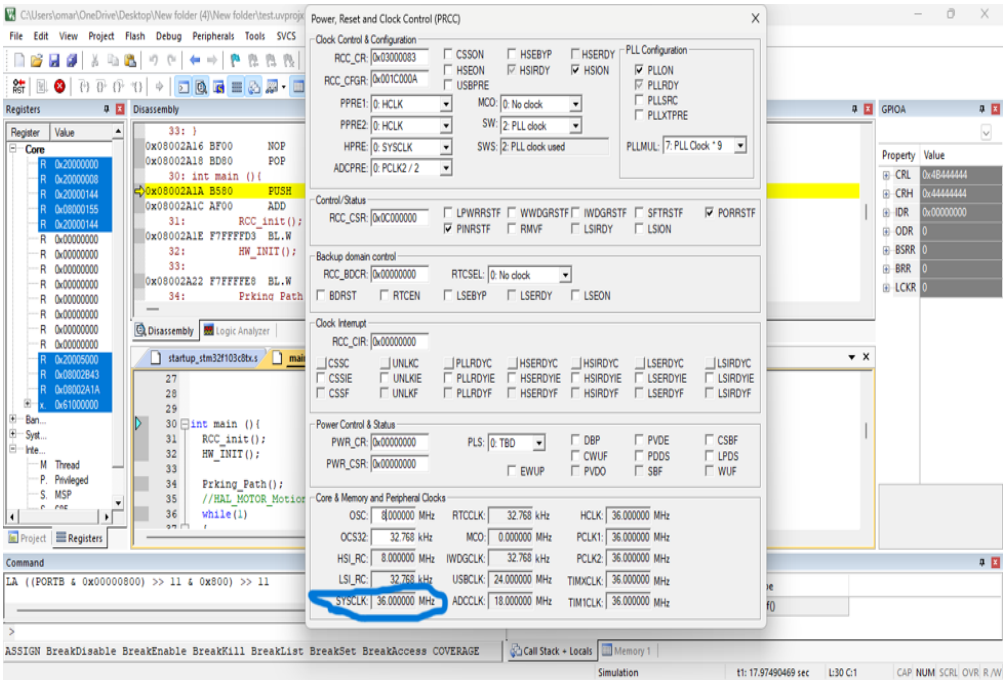


Fig 8.4. Example for a 36MHZ system bus frequency

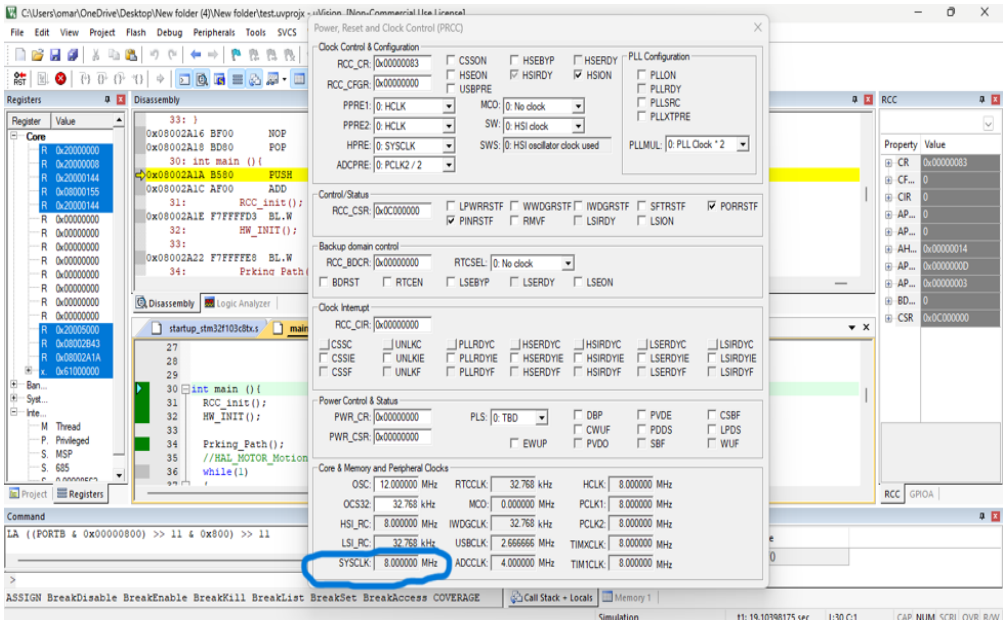


Fig 8.5. Example for a 8MHZ system bus frequency

✓ Next the GPIO (General Purpose Input Output) driver was tested; Our goals are: to control each mode and speed of the GPIO pin, write logic 0 or 1 to any pin or port, and read data from any pin or port (analog or digital). When we tested the first version we found a problem with pin configuration, so we fixed this problem in the second version.

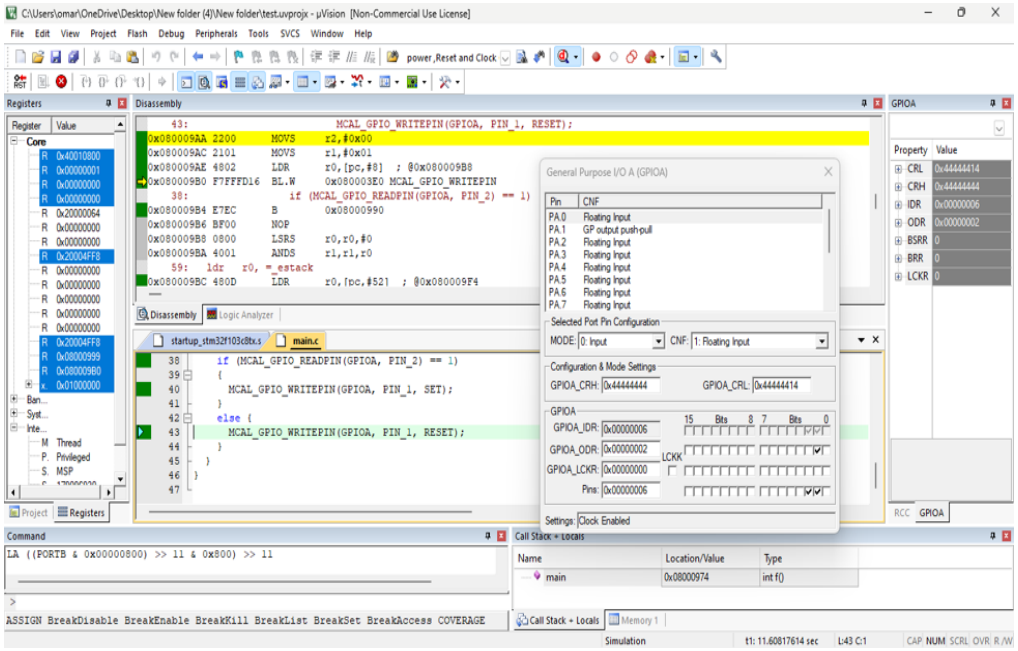


Fig 8.6. Example for configuration, Read and write functions

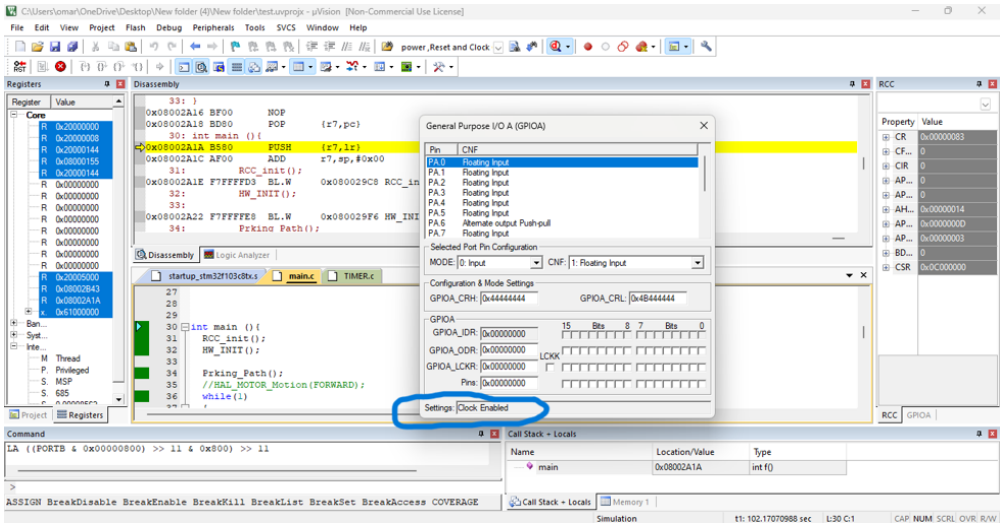


Fig 8.7. Example for Clock enable function (GPIOA)

✓ The interrupt driver was then tested; Our goal is: to control interrupt modes, and to control what happens when an interrupt occurs. When we tested this driver, we found no problems.

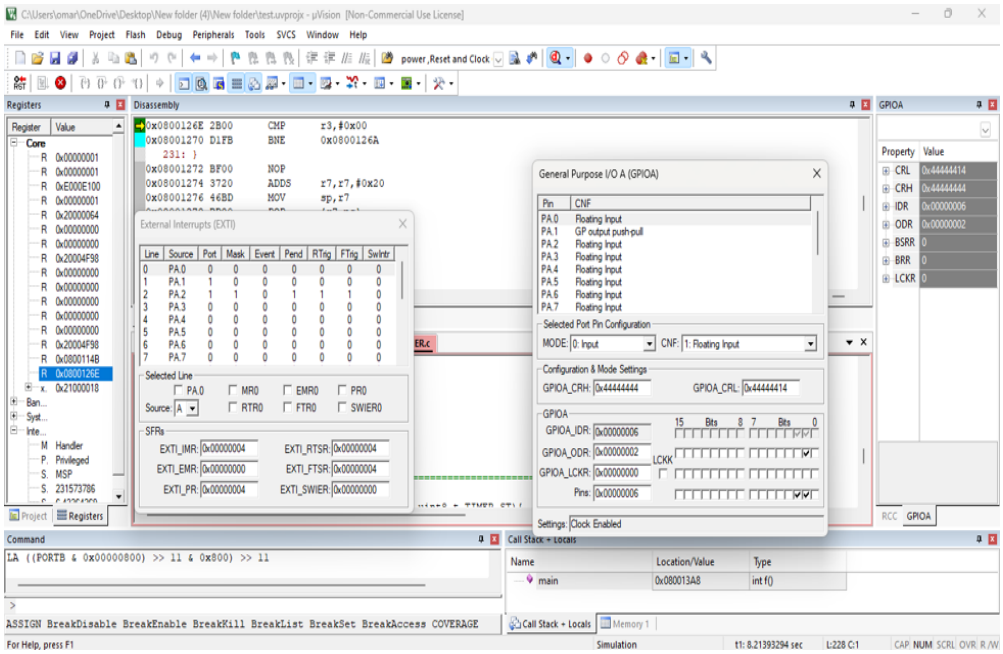


Fig 8.8. Example for interrupt

- ✓ Finally, the timer driver was tested; Our goals are: to create a function to generate PWM (Pulse Width Modulation), to create a delay function that can give microsecond and millisecond delays, to create a function to calculate time, and to create a function to use the input capture mode for use with the ultrasonic sensor. In the first version we found that only the delay function worked well so we worked on the next version to find out the problems in other functions. In the second version, we focused on the PWM function until we fixed it. After that, we worked on the input capture function, but it took a lot of time, so we replaced it with the counter, which performs the same function, but in a different way of working.

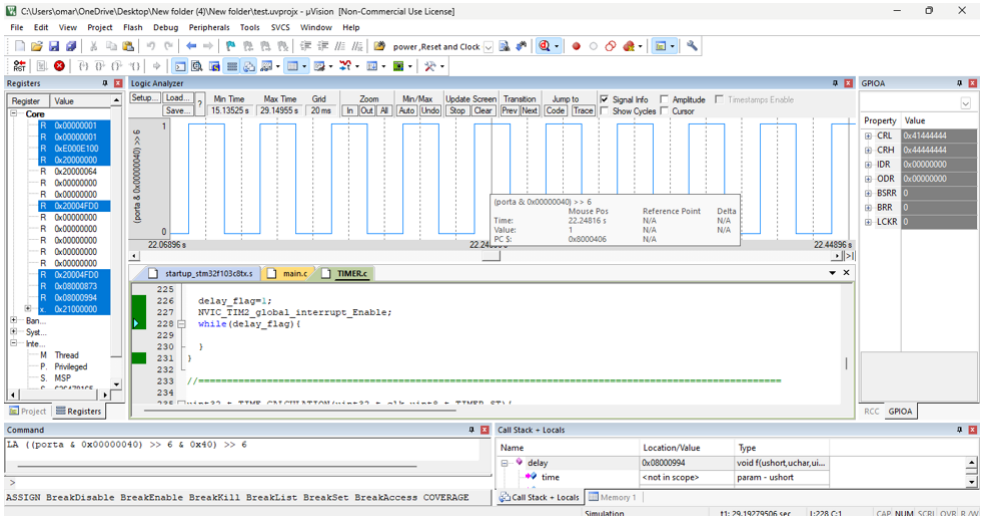


Fig 8.9. Example for PWM

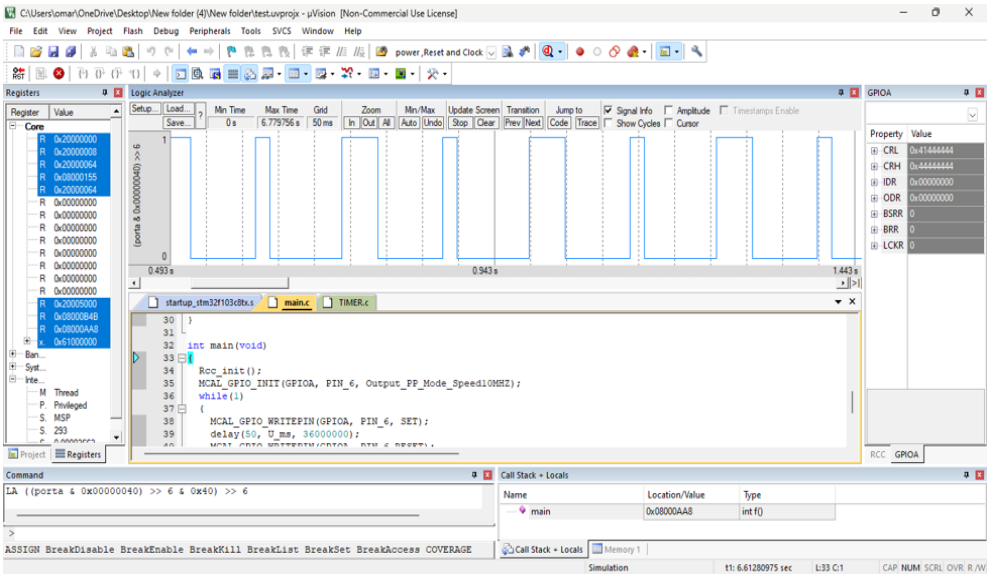


Fig 8.10. Example for delay

Then, HAL layer (hardware abstraction layer) testing has begun:

- ✓ The first driver we worked with was DC Motor driver. And it didn't have any problem with it because it's based on GPIO driver only so after the GPIO worked well this driver worked well also.
- ✓ Then we started to test Servo motor driver and the problems that we found at it, were solved when the PWM function was solved because it based on this function only.
- ✓ The last driver we tested was Ultrasonic sensor and it was the driver that take most of testing time the first problem we found was the problem in input capture mode that we talked about before, After many failed attempts we decided to use counter function and change the algorithm that driver was written with and finally the we succeeded to made the sensor work but this version was only deal with one sensor so in the next and final version we was focusing on make four sensors work together to made suitable with our application.

8.2.2 Software Performance Analysis

As we mentioned previously, the program went through several stages until it reached acceptable performance. The simulation was not limited to using only, but the components were tested by giving them some tasks and ensuring that they were completed efficiently. The systems were interfaced gradually to see and identify defects and then ensure that they worked correctly.

8.3 Path Planning Results and Performance

In Chapter 2, the most important points for moving and directing the robot were discussed and presented, including planning the schematic path for self-parking. The restrictions that were worked on in planning the route for self-parking are:

- Planning a collision-free path
- Shortest path
- Possibility of implementing the path

8.3.1 Visualization of Results

The results were simulated using the MATLAB program before implementing them practically and verifying the accuracy and quality of the path. The robot data and the appropriate parking environment data were entered, the path data was determined, and then the data was combined to link the robot to the parking spot through the path.

8.3.1.1 Path Planning for Parallel Parking

1- Car's specifications based on its design:

Length	Width	Wheelbase	F_Overhang	R_Overhang	Steering angle
41.5	18	27	6.25	8.25	45

In the final stage of parking, a may has collision with AB. The minimum distance between ab and AB is (d_2)

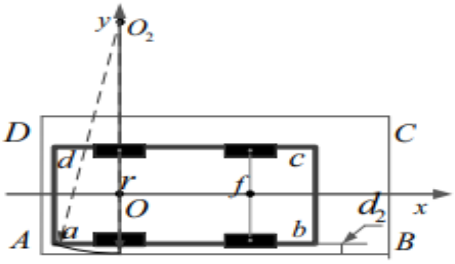


Fig 8.11. Collision circumstance1

In the initial stage of entering parking space, b may have collision with C. The minimum distance is (R_b)

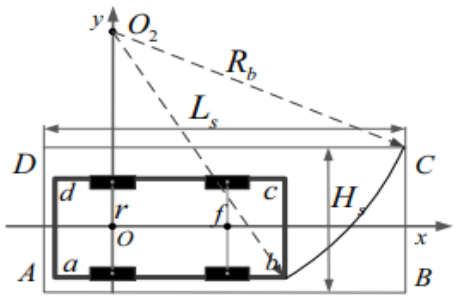


Fig 7.12. Collision circumstance2

In the initial process of reversing, the body may have collision with C because of the narrow distance between ab and CD. The minimum distance between ab and CD is (d_3)

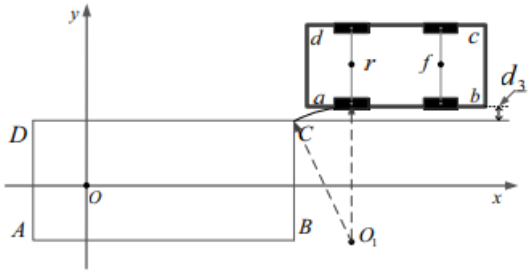


Fig 8.13. Collision circumstance3

In the initial process of reversing, c may have collision with lateral obstacle. The minimum distance between cd and lateral obstacle is (d_4)

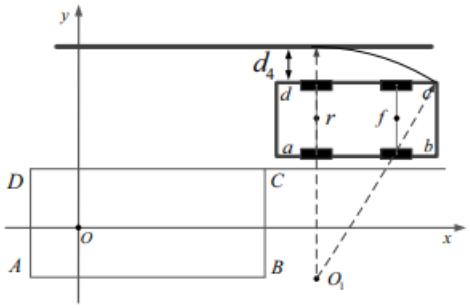


Fig 8.14. Collision circumstance4

2- From previous considerations, we can specify the parking environment:

Length	Width	d2	d3	d4	Rb
51.5	20	1	5	13	49

3- The path above is planned under the following assumptions; the start and end positions are given. The starting point will be determined by the software algorithm.

Now we can simulate the path by using MATLAB program to verify that information.

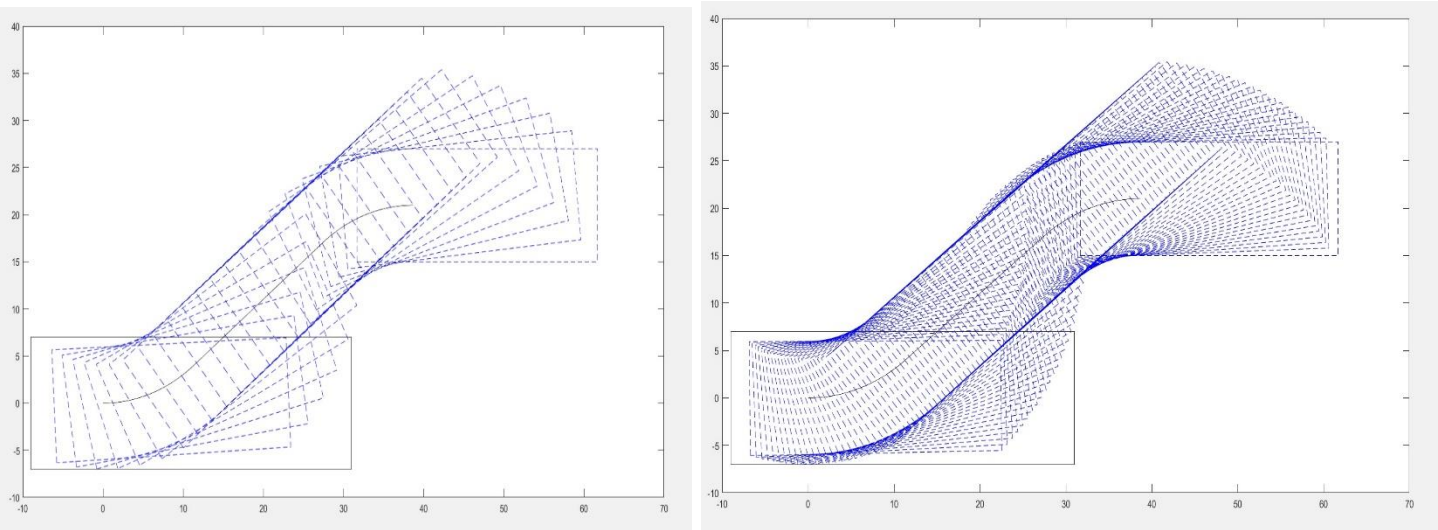


Fig 8.15. Path Simulate using MATLAB

To compute how to avoid the collision in the line path of reversing motion and in narrow spaces with the previous specifications, the shortest path is compared with the connecting line of rO

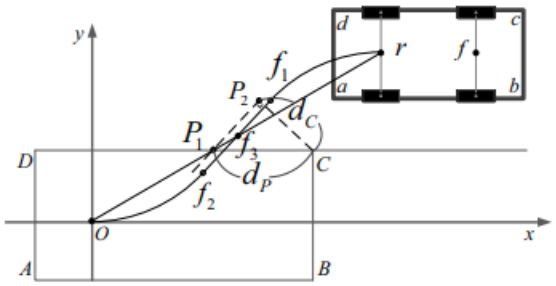


Fig 8.16. Collision circumstance5

Where rO and f1f2 intersect in f3, and rO and CD intersect in P1, P1P2 parallel to f1f2, CP2 perpendicular to P1P2, the length of CP2 is dc, the length of CP1 is dp. If the dc has the range dc>W/2, and f3 is beyond the CD, the body will have no collision with C in this circumstance.

8.3.1.2 Path Planning for Perpendicular Parking

1- Car's specifications based on its design:

Length	Width	Wheelbase	F_Overhang	R_Overhang	Steering angle

41.5	18	27	6.25	8.25	45

In the final stage of parking, delta1 may has collision with P3P4. The minimum distance is ($delta_1$).

In the initial stage of entering parking space, delta2 may have collision with P2. The minimum distance is ($delta_2$).

In the initial process of reversing, the front left corner may have collision with lateral obstacle. The minimum distance is ($delta_3$).

In the initial stage of entering parking space, delta4 may have collision with P1P4. The minimum distance is ($delta_4$).

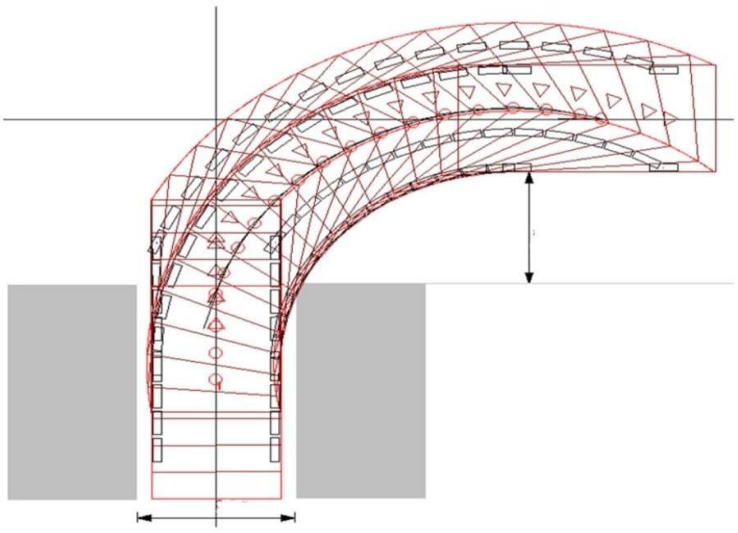


Fig 8.17. free collision Perpendicular path

2- From previous considerations, we can specify the parking environment:

Length	Width	$delta_1$	$delta_2$	$delta_3$	$delta_4$
20	51.5	1	3	9.24	33.24

3- The path above is planned under the following assumptions; the start and end positions are given. The starting point will be determined by the software algorithm.

Now we can simulate the path by using MATLAB program to verify that information.

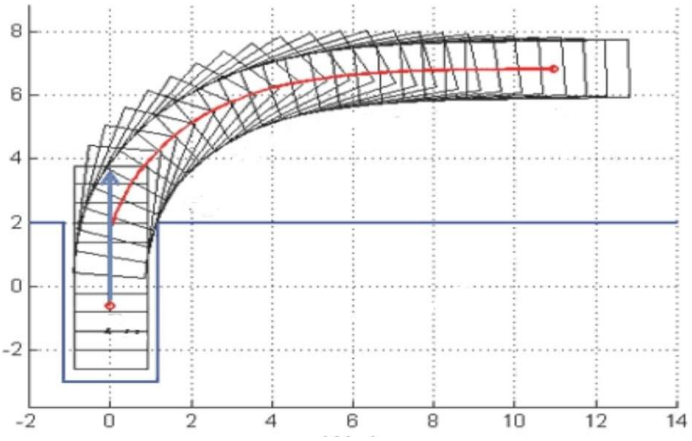


Fig 8.18 Path Simulate using MATLAB

8.3.2 Path Planning Performance

After simulating the path, ensuring its efficiency, and completing the design and assembly of the robot, it is ready for practical experience in reality.

8.3.2.1 Path Planning for Parallel Parking

Now we can test the path with real experiments based on the previous information. It consists of three phases:

- First, the robot moves forward along the length of the rear overhang of the car to reduce the path length.
- Phase 1: from start point (r) to the first point of the straight line (f1). Servo motor will change its angle to 50 and DC motor will work for time (t1).
- Phase 2: from the first point of the straight line (f1) to the last point of the straight line (f2). Servo motor will change its angle to 90 and DC motor will work for time (t2).
- Phase 3: from the last point of the straight line (f2) to the end point. Servo motor will change its angle to 130 and DC motor will work for time (t1).

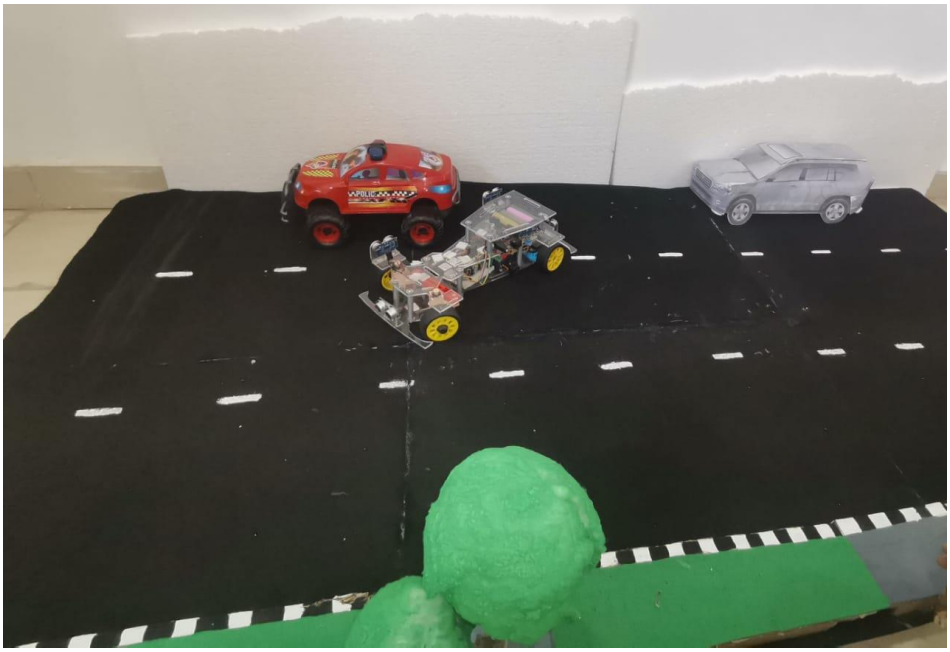


Fig 8.19 Parallel Path Planning (Start and End Point and the Path)

8.3.2.2 Path Planning for Perpendicular Parking

Now we can test the path with real experiments based on the previous information. It consists of two phases:

- First, the robot moves forward along the length of the rear overhang of the car to reduce the path length
- Phase 1: from start point (V_s) to the first point of the straight line (V_{be}). Servo motor will change its angle to 45 and DC motor will work for time (t1).
- Phase 2: from the first point of the straight line (V_{be}) to the end point of the straight line (V_d). Servo motor will change its angle to 90 and DC motor will work for time (t2).

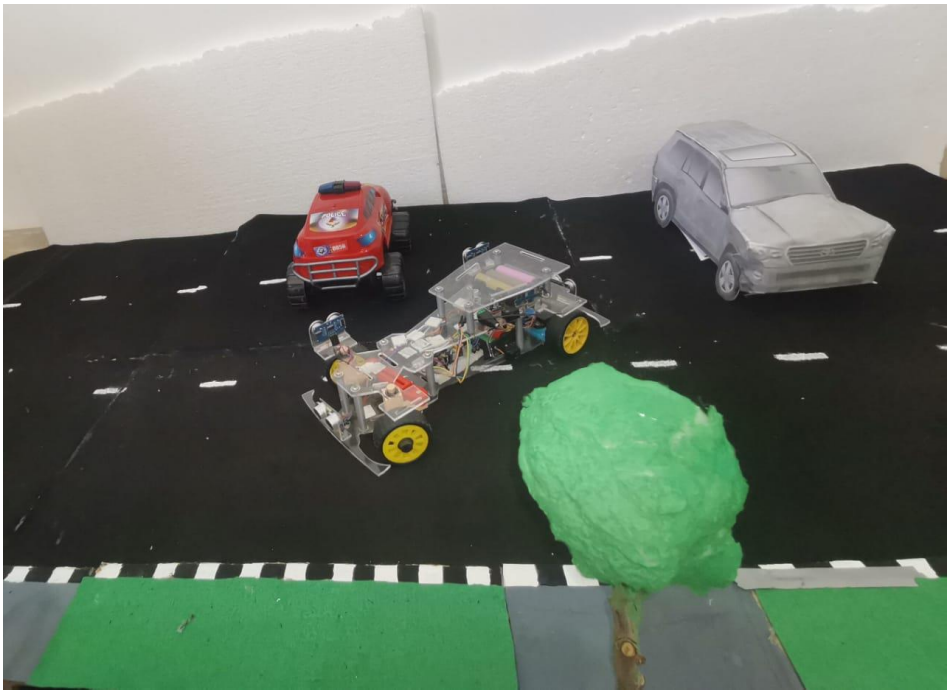


Fig 8.20 Perpendicular Path Planning (Start and End Point and the Path)

8.4 Application Results

To create an algorithm that matches the project requirements with some other requirements to reach the goal in the best way and most accurate results, there are four states in the general parking procedure. The change of each state depends on the differences in sensor data. Two ultrasonic sensors were mounted on the right side of the robot and the anterior one was identified as US1 and the posterior one was identified as US2. Any front obstacles are also avoided through the ultrasonic sensor installed in the front of the car UF1. we will assume valid width of parallel or perpendicular width is w.

8.4.1 Description of chosen algorithm

State 1: In this case, the parking robot detects the environment around it and begins to calculate the length of the first car or wall parallel to it. If any obstacles appear in front of it, the robot will stop moving and activate the alarm. Assuming that the robot moves straight, the robot moves at a constant steering angle.

If (US1 and US2 < w) state=1;

State 2: The robot detects the end of the first car and begins to calculate the length of the parking lot. After that, the robot will make the decision to stop the car or not. There is also a calibration in Case 2 based on the length of the parking lot. First, the US1 difference will be greater than the width of the parked car. The parking robot was supposed to be on top of the first car. If the difference US1 is less than the width of the

parking car again before the difference US2 is greater than the width of the parking car, the robot will decide that there is not enough parking space and go to state 1. If rear sensor US2 is on top of the first vehicle and front sensor US1 is still in the parking space at the time of calibration. The robot will again decide to stop the car or not when the front sensor US1 detects the second car, and at that time, the calibration will stop. Whether or not there is enough parking space can be known by comparing the actual detected length of the parking space and the minimum length of the parking space. If the actual parking length is greater than the minimum parking length, the robot will park the car. If not, the robot will find the next parking spot and go to state 1. During this stage, if any obstacles appear ahead, the robot will stop moving and trigger an alarm.

If (US1 > w && state==1) state=2;

If (US1 > w && US2 > w) Calibrate on;

If (actual parking length < minimum parking length) not parking, state==1;

If (actual parking length > minimum parking length) state3;

State 3: The car will stop when the rear sensor detects the second car, and the parking system has been introduced in the event that if there is no car or other side obstacle, the car can park (park in the space).

If (US2 difference > w && state==3) state=4;

If (US2 > Parallel width and length) state4(Parallel_Parking);

If (US2 > perpendicular width and length) state4(perpendicular_Parking);

State 4: path shape is formed based on ultrasonic sensor values, is it suit for parallel parking or perpendicular parking.

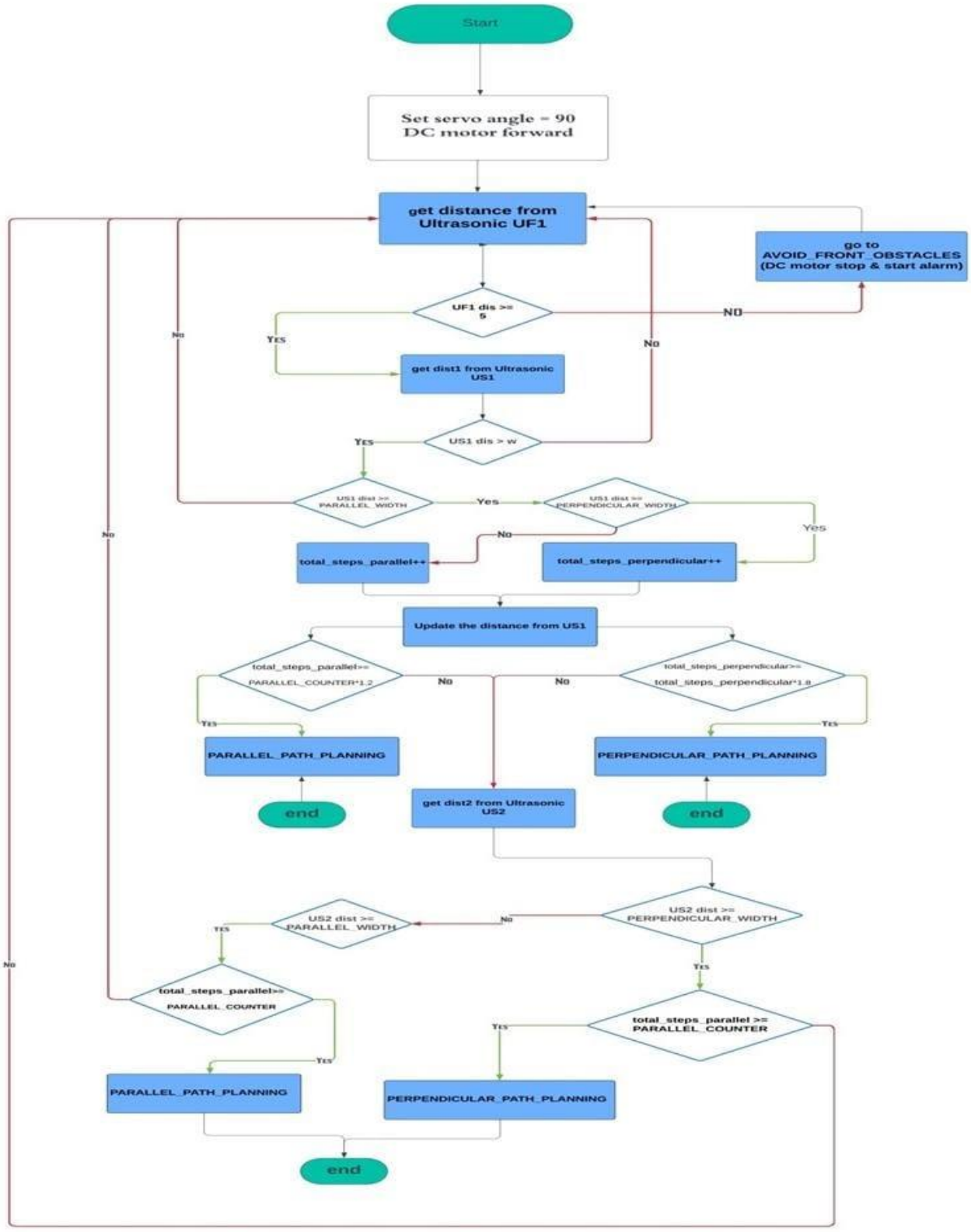


Fig 8.19 System Flow Chart

8.4.2 Testing in Real-World Environment

A multi-obstacle environment was created to test all stages of the application and ensure its efficiency and correctness.

8.4.3 Discussion of Effectiveness and Limitations

In this section, we will discuss the effectiveness of the application and what limitations we encountered.

The effectiveness of the application is centered on identifying whether the empty area suitable for parking is perpendicular, parallel, or inclined, and then implementing the appropriate path for the parking, which is what is implemented correctly through the application by comparing the width and length of the empty area through the ultrasonic sensor. In the event that the garage is empty of cars or there are no side obstacles, the application must determine the appropriate area for it, the best path to implement it, and park the robot; which is what has been implemented to avoid the robot continuing to move and its inability to recognize the environment surrounding it. One of the basics of safety is to avoid colliding with any obstacle. The application identifies obstacles that appear in front of the robot, then stops and activates the alarm. During the implementation of the parking path, work was done to create a collision-free path and ensure its effectiveness.

Among the limitations encountered are the limitations of the ultrasonic sensor. This type of sensor has many weak points as it cannot measure a distance of less than 3 cm and can only measure the distance of only one point at a time, and its efficiency is not very good. So, when we started working on mapping we were trying to do full autonomous navigation to have the CPU build a complete picture of the environment but while studying this method we found that the data we had from the sensors was not enough to use this complex method. But in this method we can use LIDAR or camera to provide enough information for this algorithm. It is also not possible to use the sensor located on the back side of the car because while parking the car, the car will be very close to the objects surrounding it, especially the back side, which makes the sensor give random values, which may spoil the parking process. When we try to implement the parking method (corner parking), the sensors cannot give enough information to determine whether there is a suitable place or not because there is always a blind spot and this may lead to damage to the car while parking and for this reason we dispense with the use of this one.

8.5 Summary

The most important stage in project implementation is testing and experimentation. It enables us to verify the validity and effectiveness of algorithms, theories, and ideas that have been reached through research and study, review the results, and work to improve them.

Software testing of components was done through Keiluvision5 and the physical components.

The path was tested through simulation using MATLAB and in reality using the robot.

The application works through an algorithm consisting of 4 states for self-parking:

- The first state is movement, guidance, discovering the environment surrounding the robot, and identifying any obstacles that hinder the robot's movement (if any obstacle in front of the robot is identified, it will stop and activate the alarm).
- The second state is to identify empty places and test those places whether they are suitable for parking the robot or not. If there is no other car, wall, or obstacle on the side of the car, a system has been created to park the car in the empty space.
- The third state begins if there is an empty space between two cars or two obstacles suitable for parking the robot. At that stage, it is determined whether the robot will park vertically, parallel, or inclined.
- The fourth state is completing the appropriate parking path and parking the robot.

Chapter 9

Future Developments

9.1 Overview

Some additions will be mentioned that will increase efficiency and accuracy and make the project more comprehensive in terms of actual usage requirements, given the time and effort expended to reach those results. How to overcome the limitations encountered in order to avoid them will be discussed.

9.2 Future Developments

- Developing sensors with more efficient sensors, such as using Lidar sensors, or better, using computer vision with cameras to draw maps.
- Adding a parking angle to the system after changing the sensors using ultrasound sensors or cameras that are able to detect the interior blind spots of an empty space.
- Add un-parking state.
- Create an application on the mobile phone so that the user can determine the position in which he wants to park the car or un-park the car.

References

[1] Zhenji Lv, Linhui Zhao, Zhiyuan Liu, “A path-planning algorithm for parallel automatic parking”, Third International Conference on Instrumentation, Measurement, Computer, Communication and Control 2013

[2] Dainis Berjova, “RESEARCH IN KINEMATICS OF TURN FOR VEHICLES AND SEMITRAILERS”, Latvia University of Agriculture, Faculty of Engineering 2008

[3] American International Journal of Sciences and Engineering Research, “Autonomous 4WD Smart Car Parallel Self-Parking System by Using Fuzzy Logic Controller”, American Center of Science and Education 2019

[4] Noor N.M, Z Razak and Mood Yamani, —Car Parking System: A Review of Smart Parking System and its Technology‖, Information Technology Journal, 2009.

[5] A.A Kamble and A Dehankar —Review on Automatic Car Parking Indicator System‖, International Journal on recent and innovation trends in computing and communication, Vol 3 no.4 pp 2158-2161.

[6] K Sushma,PRaveendraBabu and J.Nageshwara Reddy, —Reservation Based Vehicle Parking System using GSM and RFID Technology‖,International Journal of Engineering Research and Applications Vol 3 no.5 2013.

[7] Xiaochuan Wang and Simon X. Yang, “A Neuro-Fuzzy Approach to Obstacle Avoidance of a Nonholonomic Mobile Robot,” in 2003 lntemalional Conference on Advanced Intelligent MechatronicS (AIM 2W3, 2003 IEEEIASME.

[8] Gustavo Pessin, Fernando Osório, Alberto Y. Hata and Denis F. Wolf, “Intelligent Control and Evolutionary Strategies Applied to Multirobotic Systems” in 2010 IEEE.

[9] Soe Yu Maung Maunga , Yin Yin Ayeb , Nu Nu Winc, “Autonomous Parallel Parking of a Car-Like Mobile Robot with Geometric Path Planning”, American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)

[10] STM32F10Xxx Technical Reference Manual

[11] STM32F103c8 Datasheet

[12] HCSR04(Ultrasonic sensor) Datasheet

[13] User guide L298N Dual H-Bridge Motor Driver

[14]https://www.embitel.com/blog/embedded-blog/what-is-autosar-mcal-software-architecture