Software Requirements Specification (SRS) Autonomous Parking System

Authors: Omar Abdelaziz, Ahmed Maher, Moamen Magdy, Ahmed Ashraf, Mostafa Zakaria

Customer: Ahmed AbdEl-Raheem EL-Kady, Mahmoud Mohamed Ali

1. Introduction

The **Autonomous** Parking System (APS) is an intelligent parking assistance system that can quickly and safely drive the vehicle into a parking space automatically, it recognizes the parking space by sensing the surrounding environment information of the vehicle through ultrasonic and IR sensors, and according to the relative position information of the vehicle and the parking space, follow the corresponding parking algorithm to complete the automatic parking. A schematic diagram of the parallel parking process of the automatic parking system is shown in the Figure below.

1.1. Purpose

This document is intended to give an in-depth view of the components, key elements and key functions of the **Autonomous** Parking System. Components of the system will include IR sensors, Ultrasonic modules, a Human-Machine Interface, and the **Autonomous** Parking System. Key Elements will include the phases involved the execution of the system.
Key functions are the tasks to be executed for the elements to communicate with each other.
The system is developed to provide the Auto Parking feature for automotive manufacturers to add to new automobiles. This document will provide a detailed outline of the requirements of the software system for developers intending to implement the system.

1.2. Scope

The goal of this document is to define a common set of basic requirements for the software design for the **Autonomous** Parking System project.
These requirements shall be adopted and refined for the specification of Basic SW modules. that will deliver the required functionality that the specified by the customer.
The documents include specification High Level Design, Hardware and basic Software.

1.3. Definitions, Acronyms, and Abbreviations

Following are terminology, acronyms, and abbreviation used throughout this document.
- The **Driver** is the person who is sitting in the seat behind the steering wheel and controls the vehicle.
- **Autonomous Parking System (APS**) is the system to be designed and implemented.
- **The Human Machine Interface (HMI)**, is the interface which the Driver interacts with the Autonomous Parking System
- **Sensor** refers to a device that primarily detect distance between itself and an object.

1.4. Overview

Software Requirements Specification consists of 7 sections.
Section 1 details the document and the project by which this document was created. The following sections form the specification.
Section 2 gives the context of the system, constraints defining interfaces andbehavior of the system are provided. A detailed description of the functionality of the system is given. Section 2 forms a depiction of the product in the context of its use.
Section 3 is an integral part of the Active Park Assist System. The requirements of the product are listed in this section. Each requirement was used when defining the constraints and the functionality and is used in later sections.
Section 4 provides models representing Active Park Assist. Sequence diagrams, a use case diagram, and the domain model diagram are included.
Section 5 is the prototype. It is a representation of how the final product should interact with the Driver and its immediate observed environment.
Section 6 has the references used in this document.
Section 7 contains contact information for the project organizer.

2. General Description

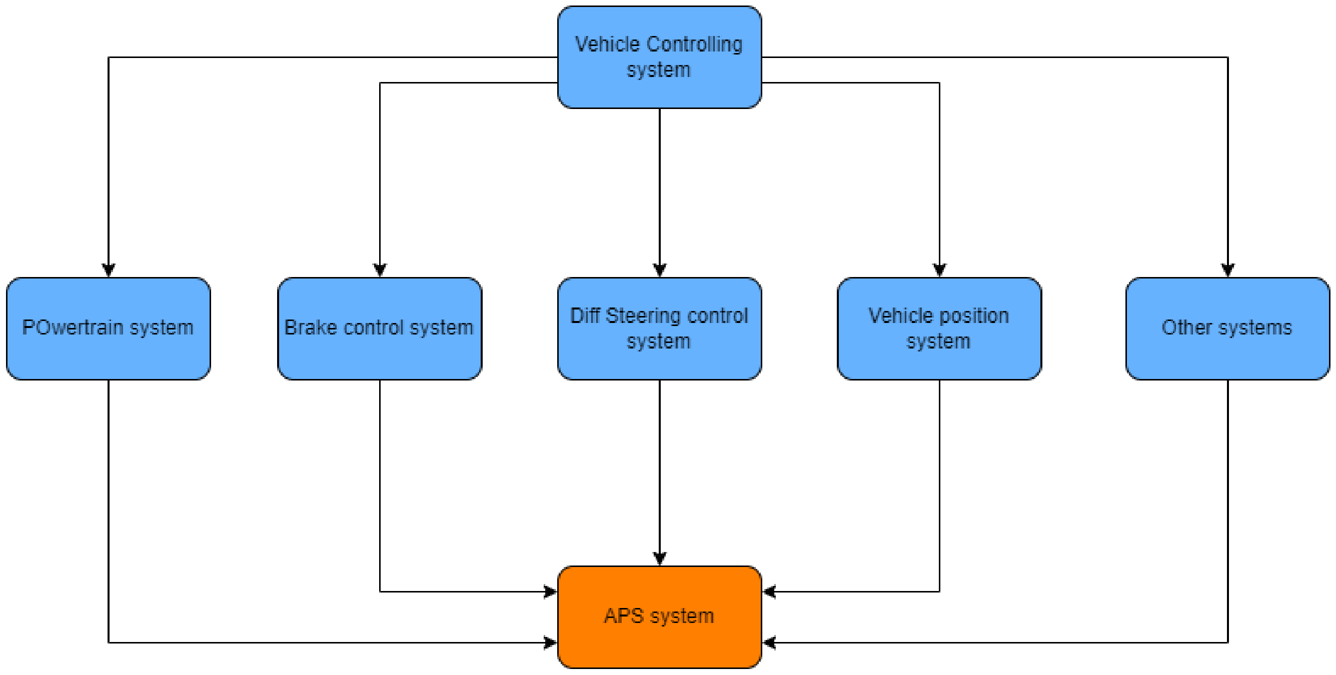
This section includes the description of the APS system, the constraints of the system, the functions it provides and the user characteristics requirements.

2.1. Product Perspective

Context:
The Autonomous Parking System (APS) system is a subsystem in the control system of a vehicle that assists the driver to automatically park on the vehicle. During the APS operation, it will work with multiple subsystems and hardware to achieve the goal of automatically parking the vehicle. The hardware and subsystems include but are not limited to the following:

- powertrain management subsystem.
- brake control subsystem.
- steering control subsystem.
- vehicle position subsystem.
- Other auxiliary systems.

Diagram:



Complete description:
In order to activate the APS system the driver needs to specify the parking preference through the HMI interface. When the above operation are completed, the vehicle will start detecting available parking spots with the sensors. The parking spot will be confirmed only when the dimensions of the detected parking area fit the preset value in the system.
When the APS system finds a suitable parking spot, the driver will confirm the parking spot and press a button to start the automatic parking.
After the confirmation, the APS system will take control of the vehicle and start the parking with the help of our implemented parking Algorithm.
During this process, the parking will be aborted if driver presses the Abort button on the HMI or an obstacle is detected during the parking process, automatically. Both abort actions will display a message on the HMI to notify the driver.
When the parking process is completed, the APS system will notifies the driver of the completion on the HMI and turn the APS system off.

2.2. General Constraints

System interfaces:
The auto parking is mostly based on powertrain system, brake control system, steering control system and vehicle position system, any of these system's failure will causes the APS system loses its primary function.
User interfaces:
The APS should only be accessed by the driver, any other person in the vehicle should not access the APS system during the APS operations.
Hardware interfaces:
The APS system will take control of the sensors/steering/brake through the subsystems such as Sensor Control System/Steering Control System/Brake Control System.
Software interfaces:
The APS system interacts with the driver through the HMI, there will be buttons on

the touch screen for the driver to either start detecting the parking spots or abort the parking process, etc. The LCD will display updated status during the parking process. any failure in HMI will causes the APS system loses its connection with the driver.

2.3. Assumptions and Dependencies

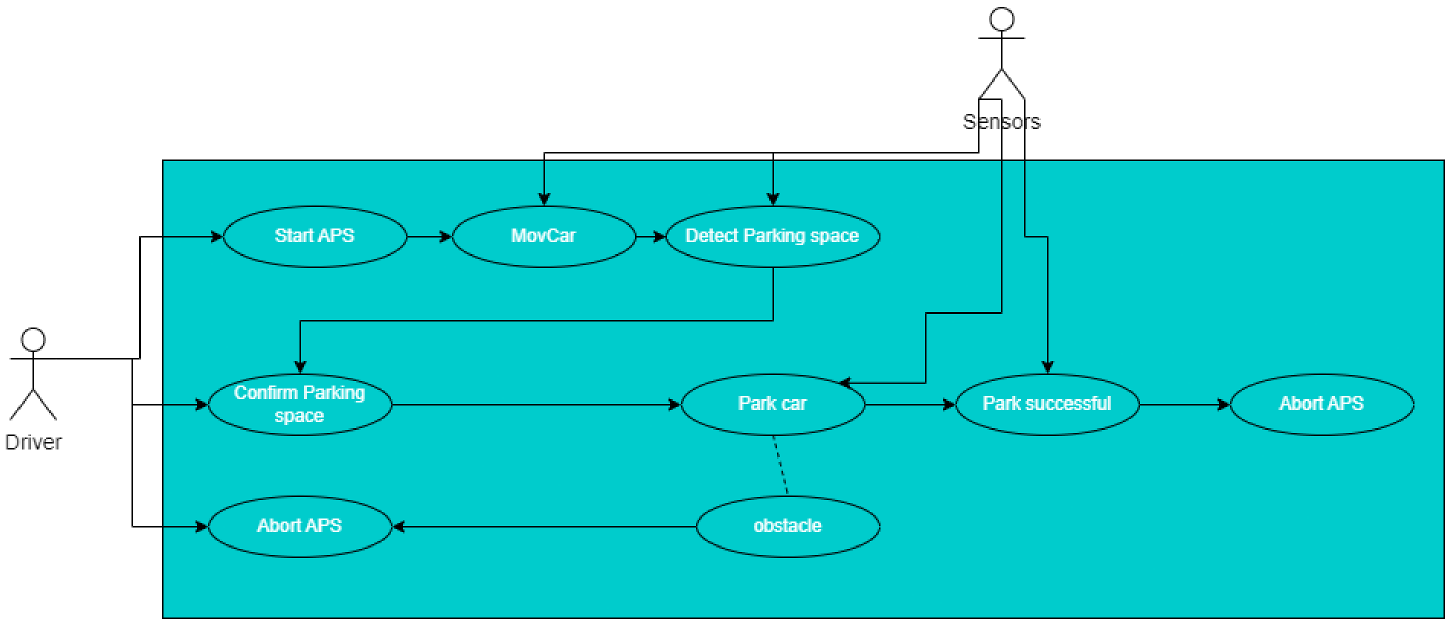
Most sections of this system assume that the car is fully functional. It also assumes the speed of the car is constant through the process of APS as well when the APS system is commenced the car is 5~10cm away from the side object (wall or parked car) in a platoon.
Furthermore, The parking space is on right side of the car, surface of the parking space is not bright for the sensors to evaluate its readings correctly.

3. Specific Requirements

[APS_100]. The Driver must select parking mode before commencing the APS.
[APS_101]. The System must confirm that all system components are online and no faults present or abort the system.
[APS_102]. The system displays an updated status of the car.
[APS_103]. System provides only parallel parking.
[APS_104]. Driver needs to confirm the Chosen parking area.
[APS_105]. while parking.The Driver can abort APS in any time.
[APS_106]. The car comes to a complete stop when APS is Complete before exiting APS Mode.

3.1. Modeling Requirements

Use Case Diagram



3.1.1 Use Cases

1

Use Case:	Start APS
Actors:	Driver
Description:	Driver starts APS by tapping a button on the HMI
Type:	Primary, Essential
Includes:	none
Extends:	Move car
Cross-refs:	[APS_100] [APS_102]
Use cases:	none

2

Use Case:	Move car
Actors:	none
Description:	Car starts moving with constant slow speed
Type:	Primary, Essential
Includes:	none
Extends:	Start Parking space
Cross-refs:	[[APS_101]
Use cases:	1

3

Use Case:	Detect parking space
Actors:	sensors
Description:	Car takes data from the sensors and use it with our algorithm to detect the suitable parking space
Type:	Primary, Essential
Includes:	none
Extends:	Confirm parking space
Cross-refs:	[APS_102] [APS_103]
Use cases:	none

4

Use Case:	Confirm parking space
Actors:	Driver
Description:	Driver confirms the detected parking space by tapping a button on the HMI
Type:	Primary, Essential
Includes:	none
Extends:	Park car
Cross-refs:	[APS_102] [APS_104]
Use cases:	none

5

Use Case:	Park car
Actors:	Sensors
Description:	Car starts parking automatically according to the algorithm we build
Type:	Primary, Essential
Includes:	none
Extends:	Park successful
Cross-refs:	[APS_102] [APS_103] [APS_106]
Use cases:	6, 7

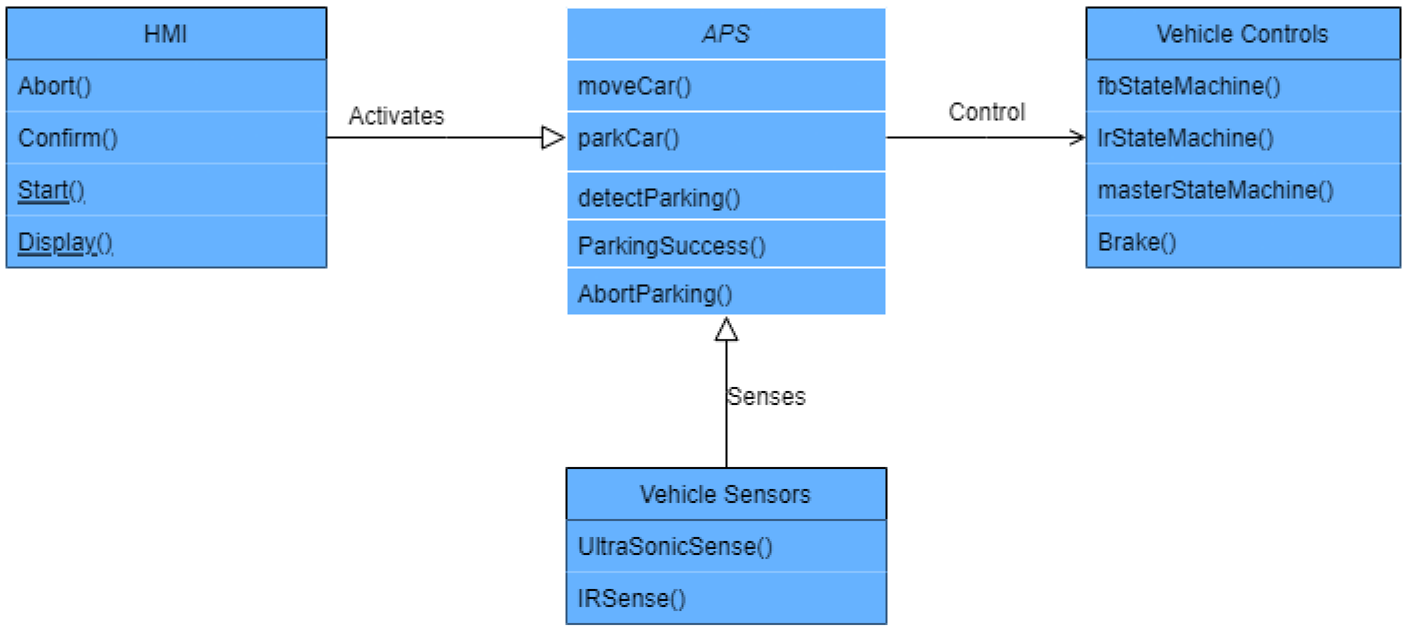
6

Use Case:	Park successful
Actors:	sensors
Description:	Car parking is successful according to the algorithm we build and notifies the user on the HMI then abort the system
Type:	Primary, Essential
Includes:	none
Extends:	Abort system
Cross-refs:	[APS_103] [APS_106]
Use cases:	7

7

Use Case:	Abort APS
Actors:	Driver , Sensors
Description:	This module is applied when the user presses a certain button on the HMI or when the car while in parking mode detects an obstacle
Type:	Primary, Essential
Includes:	none
Extends:	none
Cross-refs:	[APS_102] [APS_105]
Use cases:	none

Domain Model



State machine diagrams

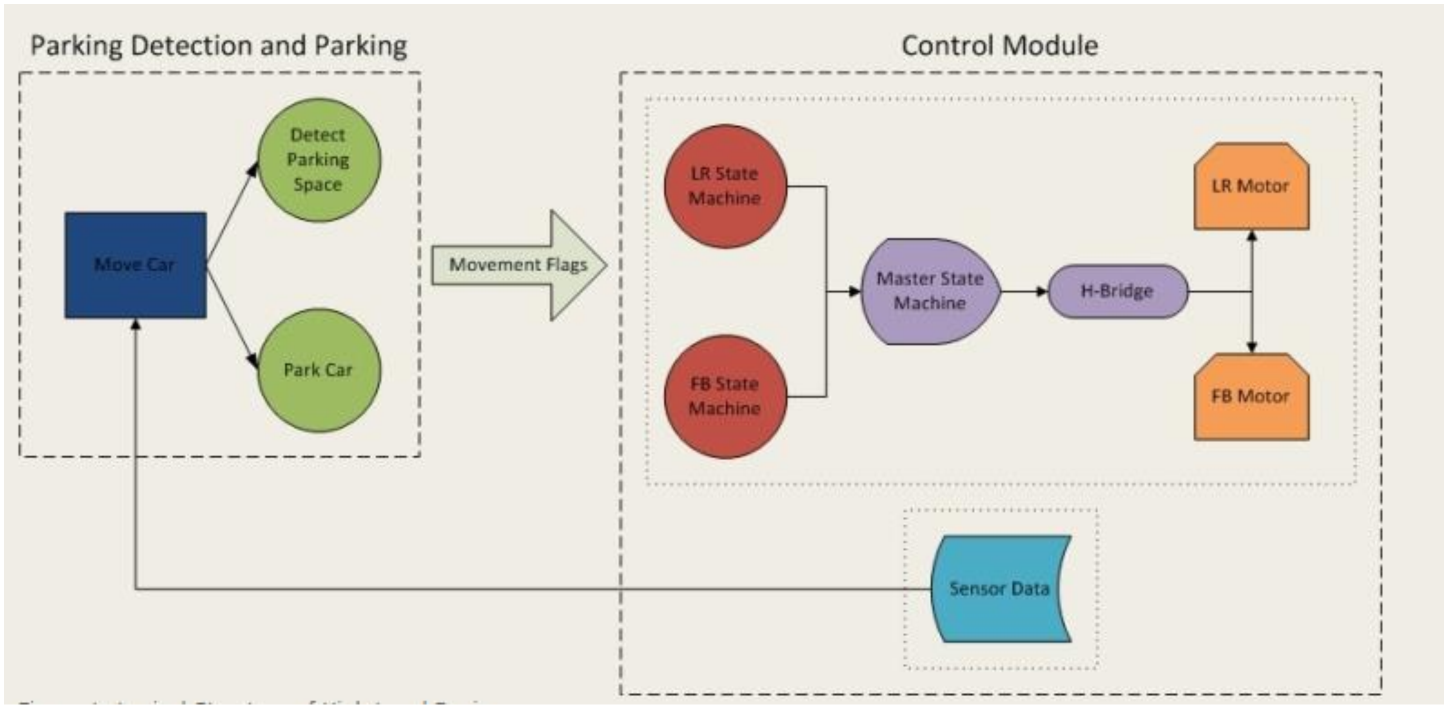


Figure 1: Logical Structure of High Level Design

Control System

The control system contains all the hardware and its associated software. It allows the parking and parking detection algorithms to interface with the car. The software in this module is broken up into three major sections: the Left-Right/Front-Back (LR/FB) state machines, master state machine, and distance calculations. The LR/FB state machines determines which direction to move the car based on flags set by the detect parking space and park car algorithms. Once the LR/FB state machines decides which direction to move the car, the master state machine implements this movement by sending the correct input and enable signals to the H-Bridge. The distance calculations implemented independently every millisecond.

Move Car

Move car contains the detect parking space and parallel parking algorithms. All functions in move car interface with the control module by setting movement flags. The parking space detection and parking algorithms use information from the distance sensors to set these movement flags and guide the car.
Move car works by initializing the movement flags of the car. It sets the car on a default trajectory and then calls detect parking space. Once a parking space has been detected, the parking algorithm is called. After the car has successfully parked, it idles until it is reset.

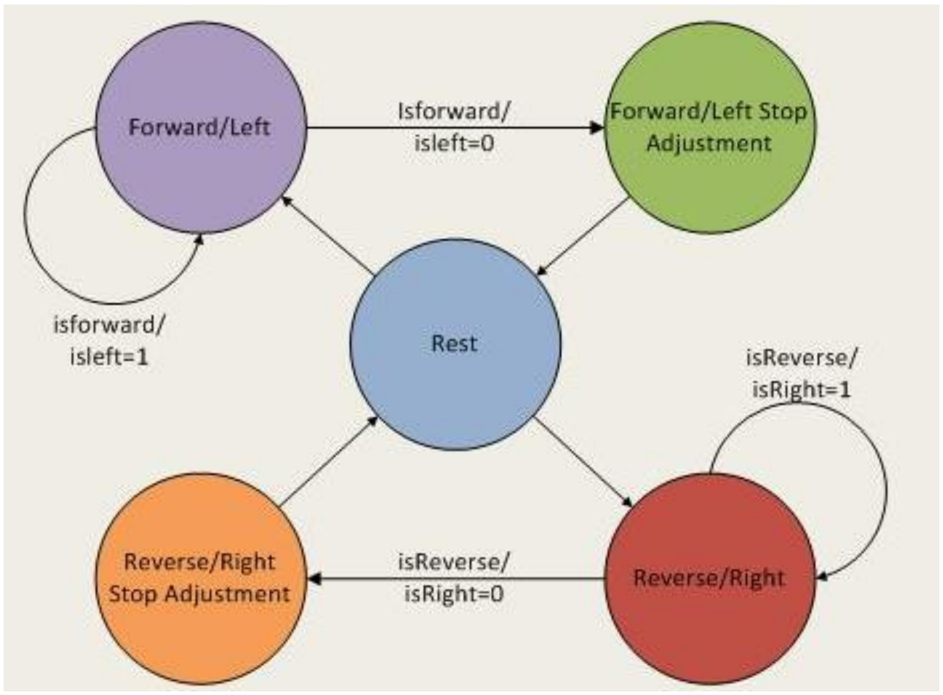


Figure 2: FB/LR Motor State Machine

fbStateMachine()

Function:

The fbStateMachine controls the motor for Forward-Backward operations. It is controlled by the is Forward and is Reverse flags. These flags serve as indicators as to whether the car should be traveling forward or reverse. In order to control the velocity of the forward-backward motion we anded the enable bit with a PWM signal.

Working:

In State 0, the motor is at rest. The corresponding FB control bits are 00. When the algorithm requires the car to go forward or reverse, the corresponding flags (is Forward and is Reverse) are set, and the FB state machine switches states to 1 or 3 respectively.
In State 1, the motor rotates to drive the car forward. The state machine remains in this state while is Forward is equal to 1. Once is Forward is de-asserted, the state machine moves to a buffer state to stop the car from moving forward due to inertia.
After is Forward is set to 0, leaving state 1 and stopping the motor isn′t enough. The wheels might continue to rotate due to inertia, and so a buffer state, State 2, is required. It makes the motor go in Reverse for 1 cycle (50ms) of the FB State Machine, before going back to the rest state, State 0.
If is Reverse is asserted, the state machine jumps to State 3. The state machine remains in this state while is Reverse is equal to 1. Once is Reverse is de-asserted, the state machine moves to a buffer state to stop the car from moving in reverse due to inertia.
After State 3, a buffer state, State 4, is needed to stop the wheels from continuing to rotate in reverse due to inertia. This is a 1 cycle Forward motion, similar in function to State 2 reverse functionality. Once done, the FB State Machine goes back to its rest state, State 0.

lrStateMachine()

The lrStateMachine() works the same way are the fbStatemachine. A forward corresponds to a left turn and a right corresponds to a reverse.

masterStateMachine()

Function:

This uses the FB and LR control bits to call the required functions in order to send the appropriate input signals to the H-Bridge and make the motors rotate in the appropriate direction.
Working:
In this function, the 2 FB and LR control bits are combined to create 4 master control bits by left shifting the FW bits by 2 and adding it to the LR bits.
Therefore,

```
fbBits = fb.controlBits; // (FB FB)
lrBits = lr.controlBits; // (LR LR)
masterBits = (fbBits<<2) + (lrBits); // (FB FB)(LR LR)
```

As a result, each of the 7 car movements (stop, forward, forward-left, forward-right, reverse, reverse-left, reverse-right) have a unique masterBits combination associated with them. The master control bits are then used in the function to decide which motor control function is to be called.

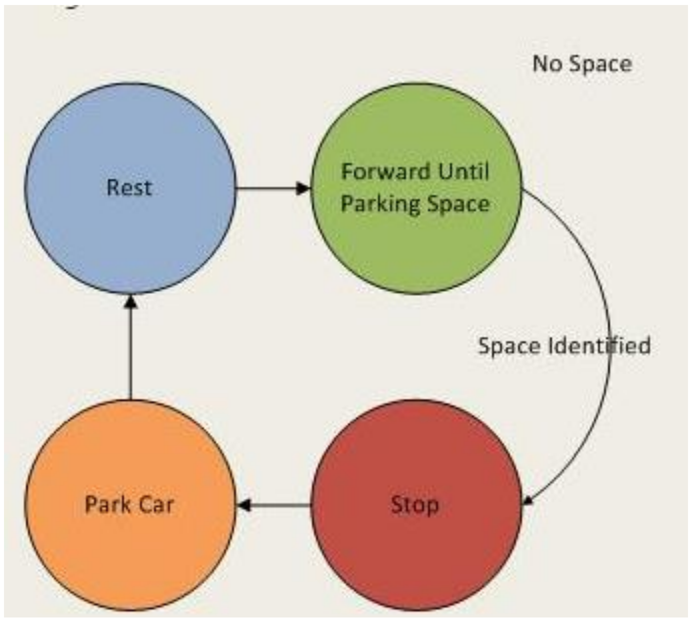


Figure 3: Move Car Motor State Machine

moveCar()

Function:

This is the master state machine of the algorithm module. It decides which mode the car is in, i.e., whether the car is moving forward to detect a parking spot, aligning itself once a parking spot has been detected, or actually going through the motion of parking.

Working:

This is a 5 state linear state machine, as can be seen in the diagram above.
It starts off in State 0. In this state, the car is at rest. It gives enough time for all transients in the car to stabilize. Once everything is stable, it moves to State 1.
In State 1, car moves forward till it detects a parking spot. While in this state, the car invokes the detectParking state machine each time the moveCar state machine is called in the Control Module. Details of how the detectParking state machine works are explained in the next section.
Once a parking lot has been detected, the state machine moves into State 2. It remains in State 2 until the car has parked itself. The parkCar state machine is invoked for each cycle that the moveCar state machine is in State 2. Once the car has been parked by parkCar state machine, the isParked flag is asserted, and moveCar moves onto state 3.
When we reach State 3, the car parked itself. The car will eternally remain in this state hereafter, since the car has parked itself and is at rest.

In addition to serving as a state machine as described above, moveCar also makes available 2 values rsDist and rrsDist to its sub-state machines, detect Parking and parkCar. rsDist stores the values of the side distance in the previous clock tick of the moveCar state machine, while rrsDist stores the value 2 clock cycles earlier.
Timing:
The moveCar state machine is invoked every 100ms. The moveCar state machine also serves as a clock for the detectParking and parkCar state machines. When in State 1, each clock tick of the moveCar state machine serves as a clock tick for the detectParking machine. When in State 3, each clock tick of the moveCar state machine serves as a clock tick for the parkCar machine.

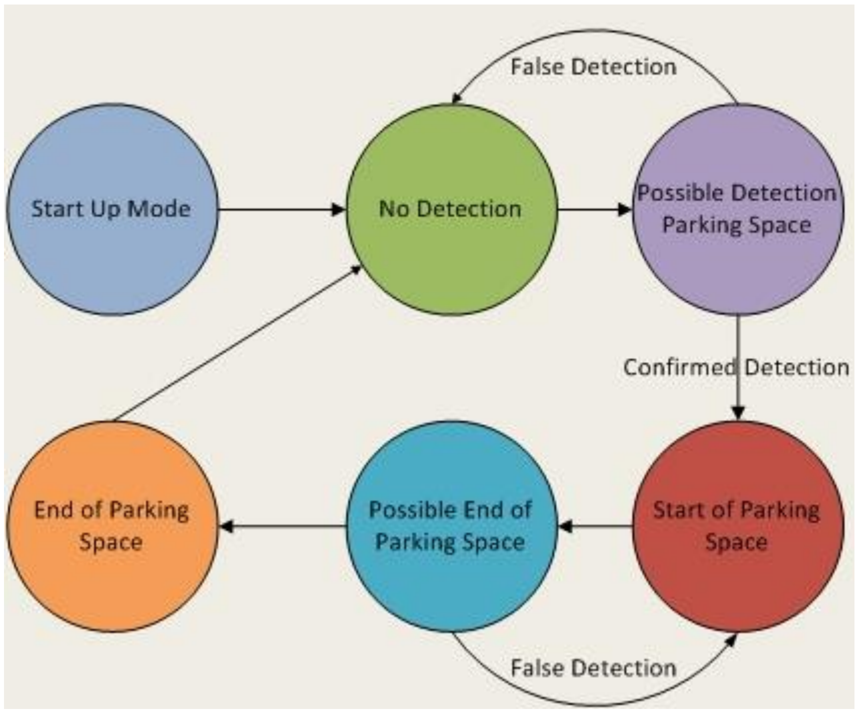


Figure 4 : Detect Parking Space State Machine

DetectParking

Function:

The function of detectParking state machine is, as its name suggests, to detect a parking space to park in. It accomplishes this by continuously polling the distance values from the side distance sensor.
Working:
detectParking is a 6 state state machine, as can be seen in the diagram above.
State 0 serves as a start-up. This is essential because the first few cycles of the detectParking take place while the side distance sensor is still calibrating itself. Once the wait state is done, the state machine enters state 1.
State 1, essentially, searches for a sudden increase in the side distance value. A sudden increase corresponds to the beginning of a parking space. It does this by checking the (sDistance rsDist) value.

If there is a sudden depression, sDistance will increase and so its difference from its own previous value (rsDist) will be a large number. When this does occur, the state machine goes onto State 2.
In State 2 it attempts to confirm that it indeed is detecting a valid depression, by calculating (sDistance rrsDist). Since State 2 is invoked 1 clock tick after the depression was last detected in State 1, rrsDist will store the value of the side distance before the depression began, i.e., from 2 clock cycles earlier. If (side distance rrsDist) is still a large number, we can confirm that a depression has been detected, and we move to State 3.
In State 3, we keep track of how long the depression is. This is done by incrementing the detect.controlBits for each state machine clock tick that we are still in the depression. When there is a sudden decrease in the value of the side distance, we move to state 4, since it signals a probable end of the parking lot.
State 4 confirms that the possible end of the parking space, as detected in State 3, is indeed the end of the space. This is done in a manner similar to the confirmation done in State 2 using the rrsDist variable.
Once a parking space has been detected by the above states, the state machine moves into State 5 wherein it checks the control Bits (which kept track of how long the parking space was by incrementing for each cock tick while in the depression) to make sure the parking space is large enough. If large enough, then the isParkingLot flag is asserted which would direct moveCar to stop and start the parking sequence.

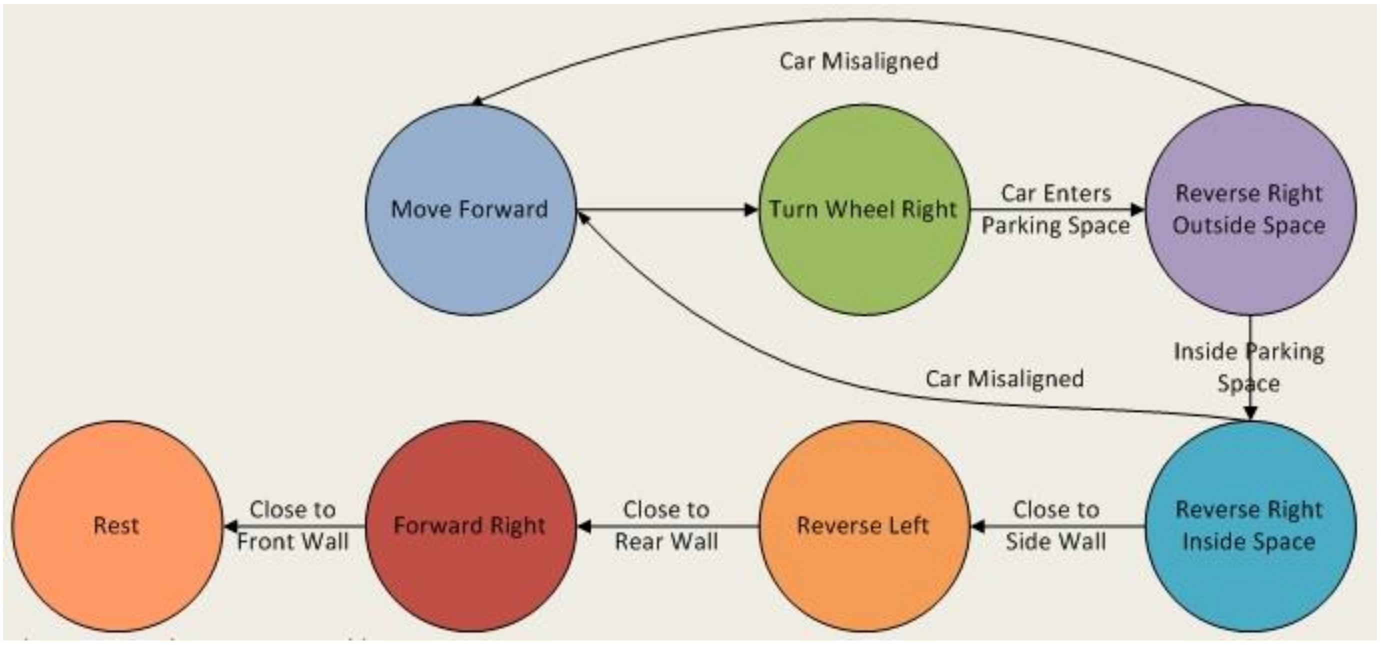


Figure 5: Park Car State Machine

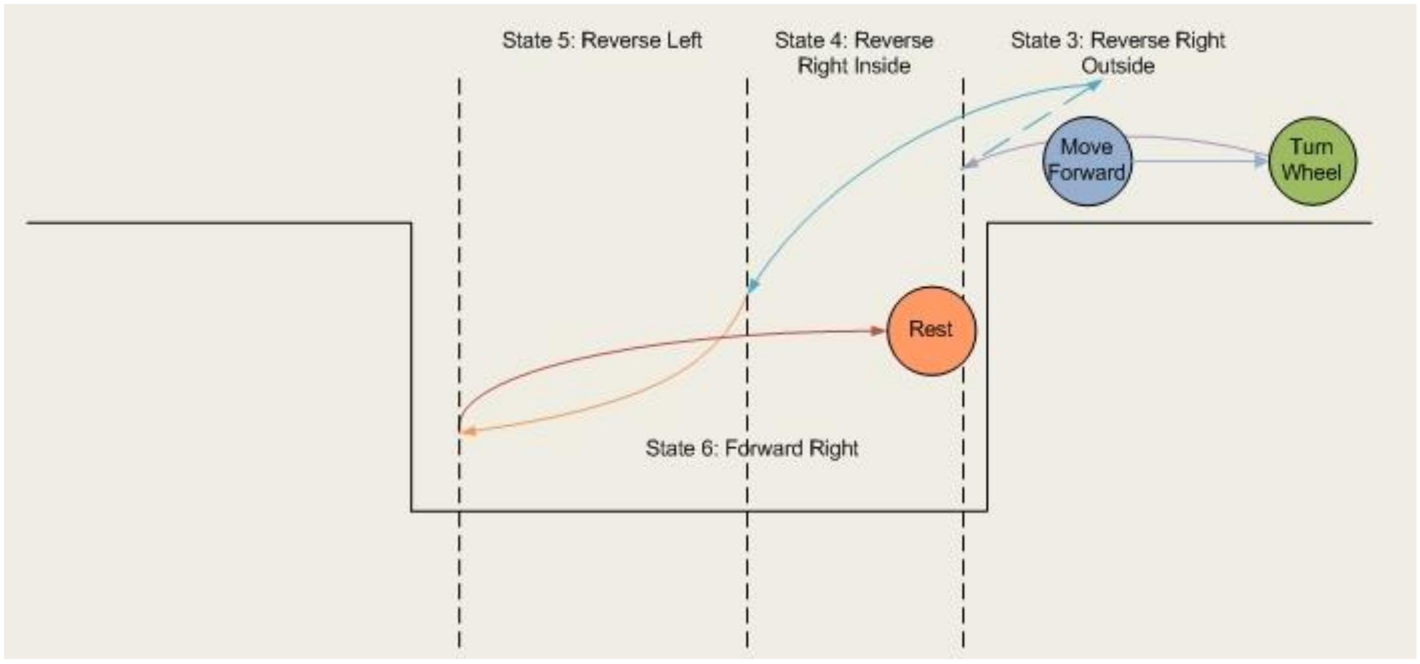


Figure 6: Parking Motion of Car, colors correspond to state of the Park Car State Machine

parkCar()

Function:

The function of the parkCar state machine is to park the car once a parking spot has been identified. The algorithm to park the car continuously interacts with its surroundings through the forward, side and rear sensors

Working:

The parkCar function tries to simulate how a human would parallel park. It is, essentially, just the following 4 motions:
Reverse Right until you are inside the parking lot.
Go Forward and redo 1. if the car is not aligned.
Reverse Left until the car is fairly straight and close to the back wall.
Forward Right until the car is straight and close to the front wall.
The above routine is accomplished using a 7 state machine.
State 0 makes the car move forward by a certain amount. The idea is to give the car enough space to move and rotate into the parking space.
State 1 simply turns the front wheels to the right. We turn the wheel before reversing the car so as to not lose turning radius by turning as the car reverses. Once the wheel is turned, the state machine moves onto state 2.
State 2 commands the car to go reverse right for a specified amount of time until the car has passed the edge of the parking space. Once past the edge of the space, it moves to state 3.
In State 3, the car continues in reverse right until it is either a certain distance from inside of the parking space, or the rear distance is close to the edge. These conditions, as can be seen from the figure above, are checks to verify that the car is deep enough inside the parking lot to be able execute the reverse left maneuver. Once the conditions are met, the car stops and the state machine moves to state 4.
NOTE: If at any point in states 1, 2 or 3 the cars AI decides it is not in a position to go through with the parking, it will go back to State 0, and redo the whole procedure.
In State 4, the car moves reverse left. It does this until the rear of the car is close to the side wall of the parking space, which can be judged by the rear distance sensor value. Once close enough to the rear value, it stops and moves to state 5.

State 5 commands the car to go forward right. This attempts to straighten out the car completely and to align it nicely inside the spot. It goes forward right until it is close to the side wall of the parking space, as judged by the forward distance sensor. Once aligned, the car is parked and it moves to state 6.
State 6 is a 1 cycle stop before progressing back to state 0. Also, here the isParked variable is set so that the moveCar state machine can move out of parking mode to rest mode.

3.2. Design Constraints

AVR is limited in resources
No steering mechanism due to RC car design

4. Change Management Process

Any changes is proceeded by source tree on our github repo and will type in the beginning of the SRS the changed or added things in the next versions if needed.