IoT Auto-Parking System

A hardware project submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

In Electronics and Communication Engineering

at Alexandria University

Under the supervision of

Prof. Dr. Nour El-din Ismail

Alexandria University, Egypt, July 2018

Team members

Alaa Adel Hussein Mohamed Nour

Abdulmaguid Eissa Abdulmaguid

Abdelrahman Gamil Ali Hassan

Ehsan Ali Elkoumy

Heba Gaber Sayed

Moustafa Moustafa Roushdy

Mahmoud Alaa El-welily

Mohamed Sobhy Mohamed

Rana Mahmoud Moghazy

Shahd Ahmed Mohamed El-kabbary

Abstract

Internet of Things (IoT) plays a vital role in connecting the surrounding environmental things to the network and made easy to access those un-internet things from any remote location. And generally, people are facing problems with parking vehicles in parking slots in a city. In this study, a Smart Parking System (SPS) is designed which enables the user to find the nearest parking area and gives availability of parking slots in that respective parking area. And it mainly focuses on reducing the time in finding the parking slots and also it avoids the unnecessary travelling through filled parking slots in a parking area. Thus it reduces the fuel consumption which in turn reduces carbon footprints in an atmosphere.

Nowadays, the rate of car market development is strikingly high and this leads to serious traffic problems especially in middle cities full of various services and malls. One of these problems people face in their daily life is parking in an appropriate slot easily and as fast as they can in order to save their time. IoT auto-parking system is presented as a prototype with the aid of IoT technology and autonomous cars to serve a fast and reliable product, saving time, fuel consumption and effort.

List of figures

List of tables

Abbreviations

API	Application Programming Interface
DC	Direct Current.
EMF	Electromagnetic Force.
FBRD	Float Baud Rate
FR	Flag Register
FSM	Finite State Machine.
FIFO	First In First Out
GPIO	General Purpose Input Output.
GUI	Graphical user interface
IBRD	Integer Baud Rate
ICR	Interrupt Clear Register
IM	Interrupt Mask
IoT	Internet of Things
IPv6	Internet Protocol version 6
IR	Infra-Red
ISR	Interrupt Service Routine
NVIC	Nested Vector Interrupt Controller
PAN	Personal Area Network
PWM	Pulse Width Modulation.
PAN	personal area network
IM	Interrupt Mask
PLL	Phase Locked Loop
RFID	Radio Frequency Identification
RF	Radio Frequency
RTIC	Reception Timeout Interrupt Clear
RX	Reception
RXIC	Reception Interrupt Clear
SLA	Service Level Agreement
SDK	Software Development Kit
TM4C123X	TivaC Launchpad
TX	Transmission
UART	Universal Asynchronous Receiver Transmitter.
UUID	Universal Unique Identifier
WiFi	Wireless Fidelity
XML	Extensible Markup Language
CCTV	Closed Circuit Television

GPS	Global Positioning System
MEOs	Medium Earth Orbits
SPS	Smart Parking System

Contents

CH.1

INTRODUCTION

1.1 System overview

In the development of traffic management systems, an intelligent parking system was created to reduce the cost of hiring people and for optimal use of resources for car-park owners. Currently, the common method of finding a parking space is manual where the driver usually finds a space in the street through luck and experience. This process takes time and effort and may lead to the worst case of failing to find any park space if the driver is driving in a city with high vehicle density.

The alternative is to find a predefined car park with high capacity. In recent years, research has used vehicle-to-vehicle and vehicle-to-infrastructure interaction with the support of various wireless network technologies such as radio frequency identification (RFID), wireless mesh network and the Internet. This study aimed to provide information about nearby parking spaces for the driver and to make reservation minutes earlier using supported devices such as smartphones or tablets.

Using the IoT technology, where it has created a revolution in many fields of life as well as in SPS technology. The present study proposes and develops an effective cloud-based SPS solution based on the IoT.

This system constructs each car park as an IoT network, as an IoT network which counts the number of free slots, counting the time spent by parking cars in each slot and calculating the cost according to this time.

The data centre serves as a cloud server to calculate the costs of a parking request, and these costs are frequently updated and are accessible any time by the vehicles in the network. The project is based on several innovative technologies and can automatically monitor and manage car parks.

Furthermore, in the proposed system, each car parking area can function independently as a traditional car park.

This project implements a system prototype with wireless access in an open-source physical computing platform based on Raspberry Pi with RFID technology using a smartphone that provides the communication and user interface for both the control system and the vehicles to verify the feasibility of the proposed system and to represent its ability to be applicable in our life with many developed ways in the scope of hardware/devices and scope of software techniques.

It is preferable to use friendly and easily used devices as the aim of the project is to have a complete prototype working reliably with accepted efficiency without taking much care of using more complex devices as complexity was not in the scope in the project.

This system can collect information about the state of occupancy of the car parks, and can direct drivers to the nearest vacant parking spot by using a software application.

In this project, designing a complete architecture of a large-scale parking system was not the aim. The aim of this project is seeking for implementing a small prototype to represent the complete idea of IoT parking system. This architecture provides drivers with information about parking slots availability and tells them about the most convenient parking slot at their destination before their departure.

Using RFID and IR sensors in this system, RFID tag is used for checking and authentication user information about car's number and color.

As a protection procedure, a smoke detector sensor is added to the garage and connected to the Raspberry Pi to alert it if the concentration of fuel gases exceeded a certain value as an indication that there is a fire in the garage.

When a car parks or leaves the parking slot, the IR sensor at each parking slot detects the action and send this information to the unit controller to update the information on the car park status. A part of this system is implemented in the IR sensors which send urgent information to the Raspberry Pi (garage controller).

The Raspberry Pi acts as a gateway controller which is responsible for gathering data from these different sensors (RFID-IR sensor-smoke detector), pushing this data to the cloud and performing some actions according to this kind of data. It sends this data to the cloud server where it is stored, analyzed and represented to the coordinator who supervises the garage situation to be aware of the new updates of its parking slots and the identity of the parking cars.

The status of the overall parking slots in the system is always updated in the real time in the system database.

On the other hand, any driver should be aware of this system to get benefit from it, also should download the mobile application on his mobile. This application has the advantages of a web service where he can quickly access the system server using the 3G/4G connection and gather all the parking information for each parking slot.

Once the driver arrives at the entrance of the garage, he can easily choose any empty parking slot he wants by just clicking on it on his application.

To reach a complete model of smart parking system based on IoT, it is better to be as much automated as possible since automation is one of the special characteristics of an IoT system in which devices are working and transferring data in different spaces by themselves with a very small area for human interaction.

Therefore, it is better for the parking car to be a dedicated auto parking car which can park smartly with an acceptable easiness and efficiency. So when the driver at the garage entrance chooses a specific empty slot, he will have the opportunity to order his car to park alone through a wireless command signal sent from his mobile application to the car using a Bluetooth connection between the car and the mobile. Once the car gets the command signal to park and the number of the parking slot, it will immediately move to the required parking slot and begin the parking process.

As a sum up, this project represents a miniature model of an automated car parking system that can regulate and manage the number of cars that can be parked in a given area at any given time, based on the availability of the parking spaces. The automated parking method allows the parking and exiting of the cars using sensing devices. The entry to or the exit from the car park is commanded by an Android-based application.

The system is derived from the idea of IoT that is based on transferring of information from different nodes in different places wirelessly. It uses the RFID technology to authenticate car information and IR sensors to monitor the parking slot state. The use of IR sensor facilitates the implementation of a large-scale system at low cost. The system provides a mechanism to prevent disputes in the car park and helps minimize wasted time in looking for a parking space. After logging into the system, the user can choose a suitable parking space once he comes to the garage.

This system will also calculate the parking time for each parking space in real time and output the parking cost for this time. It will support the business with hourly parking charges. At each car park, the system allows a driver to find information on each parking slot without the need to directly access the local server node by directly accessing the cloud-based server using the mobile application.

1.2 System operations

1. When a user wants to find a parking slot, he must log in to the system. After successful login, a request message is sent to search for a free parking slot. Then the user arrives at the car park, he must be authorized

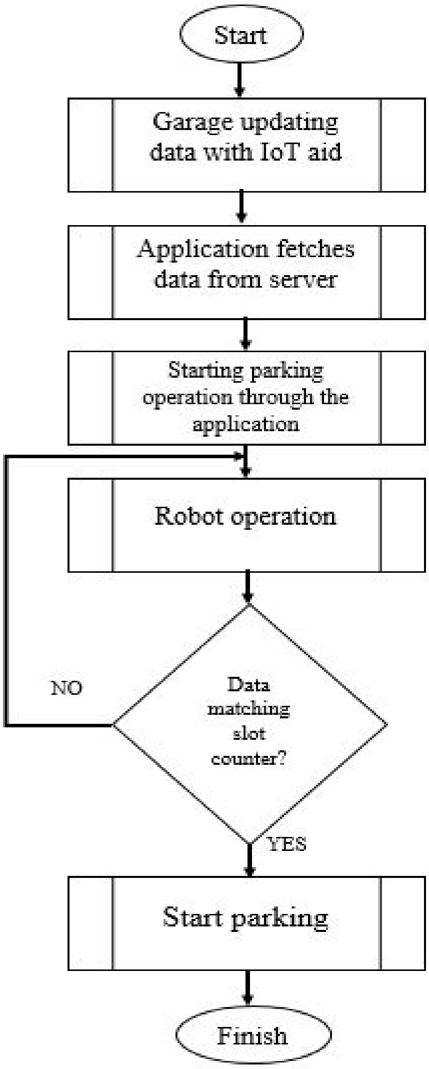

Start
Garage updating data with IoT aid
Application fetches data from server
Starting parking operation through the application
Robot operation
NO
Data matching slot counter?
YES
Start parking
Finish

Fig. (1.1) Flow diagram of the system operations to enter. This authorization is achieved via the RFID technology or by scanning the user card. This mechanism is simple and economical then IR sensors in each slot to detect whether it is full or empty slot and send data to the server to update the database.

2. When detecting any fire through smoke detector circuit, it sends fire alarm immediately to the moderator to appear on server website.
3. The user can connect with the smart parking system with their smartphones or with some browsers.
4. The user chooses the slot and sends the command to the car to park.
5. The car starts parking..

1.3 The project prototype

It presents a prototype of a low-cost and flexible smart parking monitoring system using an embedded micro-web server, with Internet Protocol (IP) connectivity for accessing and controlling devices and appliances remotely. The general approach is to design a usable IoT smart parking system for stakeholders (end users), where the methodology for accomplishing this goal was to apply User-Centered Design (UCD), which included observation, surveys with stakeholders.

This prototype enabled a car to detect entry to the car park and guide the driver to an empty parking space and it also uses a self-parking car to provide more facilities for the user as after the microcontroller receives the number of slot specified by the user application to be parked in parking process, it starts the operation. An algorithm gathering these different states and inputs to perform the task of parking automatically and without the help of the driver.

Figure (1.2) illustrates the design of the project, which is composed of 3 main parts: 1- Autonomous car 2-Web server 3-Mobile application

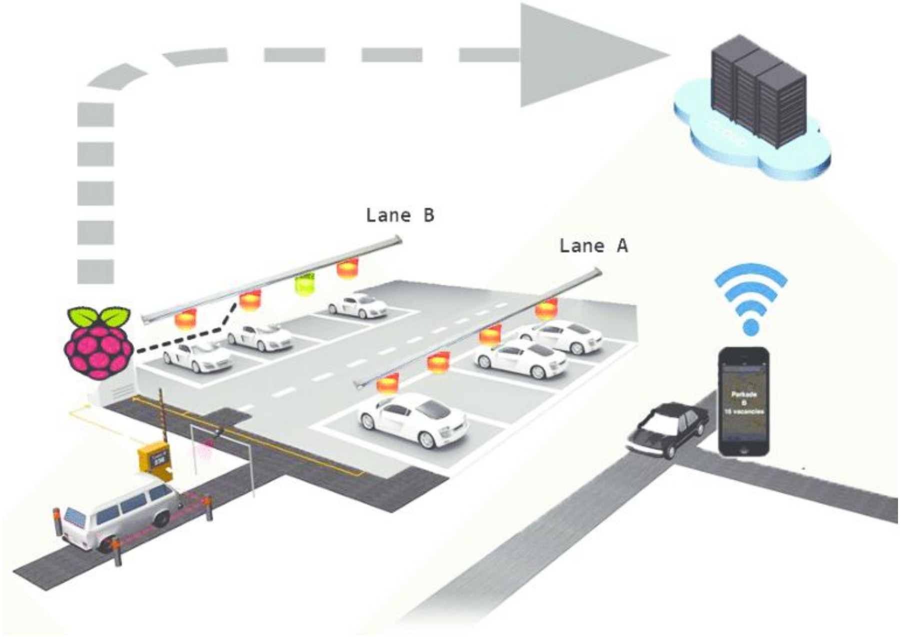


Fig. (1.2) Project design

1.4 Hardware and software requirements

1.4.1 Project main elements

- **Centralized server:** Maintains databases which contain information about parking spaces present in the parking system. 000webhost is used as a free platform which provides the facility to make an accessible website accompanied with its database.
- **Raspberry Pi:** Represents the microcontroller which is used to implement the parking system and it is attached with RFID tags and IR sensors and smoke detector to collect data from them.
- **Website:** Which is displayed as the admin side interface for the coordinator of the parking system so that the coordinator can monitor the real-time updates in the parking area and can deal with any strange or sudden situation like a fire alarm.

- **User application:** An application which can be connected to the smart parking system server to get the state of each parking slot and through which the user can order the self-parking car to park in a specific slot.
- **Self-parking robot:** An Auto parking car which is smart enough to park precisely in the required slot by itself once it receives the order from the application and the number of the parking slot.

1.4.2 Languages used

- **Python:** For the Raspberry Pi, as it is a powerful programming language that is easy to be used with Raspberry Pi to connect it with the real world. Python has different already used and tested libraries to interface it with different external devices.

- **Hypertext Markup Language (HTML):** It is the standard markup language for creating web pages and web applications. HTML elements are the building blocks of HTML pages, with HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, tables, lists, links, quotes and other items. It is used to output any texts and table needed to be represented on the website for the parking system coordinator.

- **Cascading Style Sheets (CSS):** CSS describes how HTML elements are to be displayed on the screen, paper, or in other media. It is mainly used for design purposes in the website. In the project, CSS is just used to modify the format and the appearance of the table that should be displayed on the website to the garage coordinator.

- **Hypertext Preprocessor (PHP):** For the server website which is a server-side scripting language designed for web development, but also used as a general-purpose programming language. It is used to execute the SQL command upon dealing with the database either in case of inserting data into it or fetching data from it. It is used to perform the tasks of receiving the data from the Raspberry Pi, storing it in the database and to analyze some data to output a certain information needed to be represented on the server website and send to the user mobile application.

- **Structured Query Language (SQL):** SQL is used in programming and designed for managing data held in a relational database management system or for stream processing in a relational data stream management system. It is particularly useful in handling structured data where there are relations between different entities/variables of the data. In this project, it is used to directly store the real-time updated data coming from the Raspberry Pi in the virtual storage (database) and to make some operations on it like counting the number of empty slots and calculating the time spent by any car parking in its slot.

- **JavaScript:** JavaScript is used for the connection between the server and the mobile application to make an API through which data is transferred from the server to the mobile application.

- **C:** For the parking car as C is a widely used programming language in the context of embedded systems because of its ease to directly access the low-level hardware registers to make controllers acts as required, since ARM architecture based controller (TM4C123GH6PM) is used.

- **Java:** For the development of the mobile application using Android studio.

1.4.3 Project steps

1. Preparing parking system prototype with IR sensors in each slot to detect whether it is a full or an empty slot.
2. Interfacing Raspberry Pi with IR sensor. IR sensors emit and receive infrared radiation then detect an alarm if an object is close to the sensor.
3. Interfacing Raspberry Pi with RFID sensor for authentication and collect the car’s information.
4. Interfacing Raspberry Pi with smoke detector circuit to send fire alarm to modulator on the server side.
5. Sending all information about parking system including parking slots and the smoke detector taken from different sensors in parking system to cloud.
6. The used cloud server 000webhost (centralized database) will be

periodically updating its database according to the information coming from Raspberry Pi.

7. Developing of the project application, where the user can connect with the system server and choose a specific parking slot for the car to park.
8. Then self-parking car begins parking process automatically depending on received information from the server.

CH.2

GARAGE CONTROLLER USING RASPBERRY PI

2.1 Chapter objectives

- Experiencing the main concepts of IoT and apply them on a practical prototype.
- Dealing with widely used development board like Raspberry Pi and interfacing various sensors to it.
- Building a complete IoT model based on (Sensors-Gateway-Cloud-Application)

2.2 IoT

IoT is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. A thing, in the IoT, can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built-in sensors to alert the driver when tire pressure is low or any other natural or a man-made object that can be assigned an IP address and provided with the ability to transfer data over a network.

IoT was first introduced in 1999 at Auto-ID centre and first used by Kevin Ashton, the co-founder and executive director of auto-ID Center at MIT, As evolving this latest burning technology, it promises to connect all our surrounding things to a network and communicating with each other with less human involvement.

In a presentation he made to Procter & Gamble in this year he explained the potential of the IoT as follows:

“Today computers and, therefore, the internet are almost wholly dependent on human beings for information. Nearly all of the roughly 50 petabytes of data available on the internet were first captured and created by human beings by typing, pressing a record button, taking a digital picture or scanning a barcode.

The problem is, people have limited time, attention and accuracy all of which means they are not very good at capturing data about things in the real world.

If we had computers that knew everything there was to know about things using data they gathered without any help from us, we would be able to track and count everything and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling and whether they were fresh or past their best.”

IPv6’s huge increase in address space is an important factor in the development of the IoT. Because of Ipv6 wide address space range, we could assign an IPV6 address to every atom on the surface of the earth, and still have enough addresses left to do another 100+ piles of earth. In other words, humans could easily assign an IP address to every "thing" on the planet. An increase in the number of smart nodes, as well as the amount of upstream data the nodes generate, is expected to raise new concerns about data privacy, data sovereignty and security.

IoT has evolved from the convergence of wireless technologies, micro-electromechanical systems (MEMS), micro services and the internet. The convergence has helped tear down the silo walls between operational technology (OT) and information technology (IT), allowing unstructured machine-generated data to be analyzed for insights that will drive improvements.

The number of IoT devices increased 31% year-over-year to 8.4 billion in 2017 and it is estimated that there will be 30 billion devices by 2020. The global market value of IoT is projected to reach $7.1 trillion by 2020

2.3 IoT layers perspective

In October 2014, the conference of the world association of IoT was held where CISCO has partnered with companies such as IBM and Intel. In this conference, they proposed a reference model of IoT layers.

This model defines the main functions of each layer and shows the interaction between them. By studying this model, the first thing that comes to mind is the similarity of its multilevel idea with the famous OSI model.

The model breaks down into seven functional levels the dozens of technologies that, all combined, comprise the IoT.

Devices send and receive data interacting with the Network where the data is transmitted, normalized, and filtered using edge computing before landing in data storage and databases accessible by applications which process it and provide it to people who will act and collaborate. See figure (2.1).

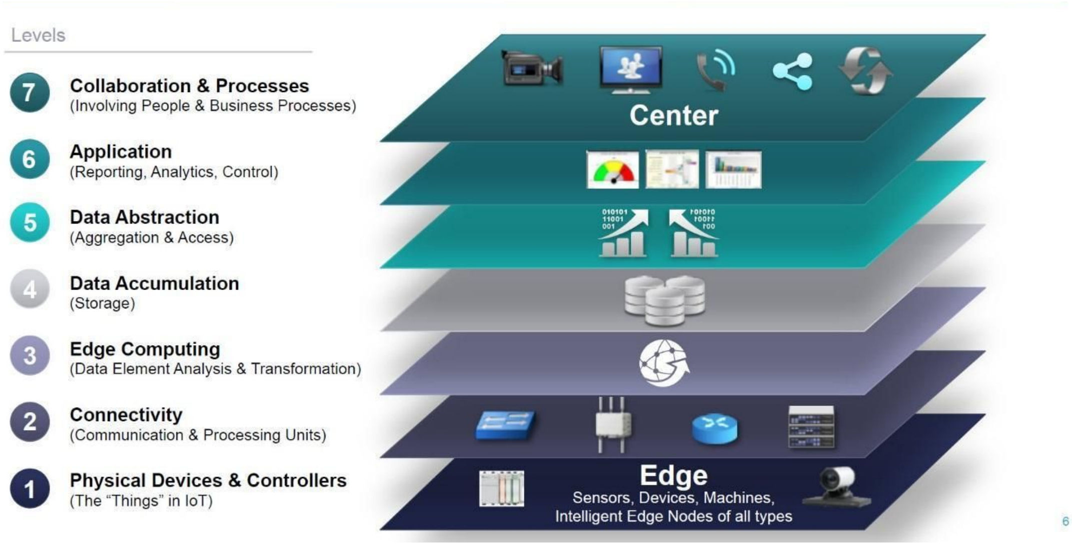

Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types
6

Fig. (2.1) IoT layers. [1]

In this project, It is important to focus on making an application on the main and most important IoT layers which are:

1. **Physical layer**
   It is comprised of devices, sensors, controllers etc. It is the layer where sensors are doing their works to continuously measure data and sending it to the gateway.

2. **Edge computing layer**
   Where the Raspberry Pi acts as a Gateway which gathers all the data from the different sensors, makes some analyzing on it and transform the required data to the cloud (server).

3. **Data accumulation layer**
   Where the server receives the data sent from the gateway and stores it in its database and may represent it to in a website.

4. **Application layer**
   Where the required data are represented to the client through a mobile application connected wirelessly to the system server. Using this application, the user can see the periodic change of the information and the real-time and make some decisions to be performed in the physical layer according to this information.

2.4 Main IoT concepts

2.4.1 Sensors

IoT loses its distinction without sensors. They act as defining instruments which transform IoT from a standard passive network of devices into an active system capable of real-world integration.

The most important hardware in IoT might be its sensors. These devices consist of energy modules, power management modules, RF modules, and sensing modules. RF modules manage communications through their signal processing, WiFi, ZigBee, Bluetooth, radio transceiver and duplexer.

The physical objects that are being connected will possess one or more sensors. Each sensor will monitor a specific condition such as location, vibration, motion and temperature. In IoT, these sensors will connect to each other and to the system that can understand or present information from the sensor's data feeds. These sensors will provide new information to a company's systems and to people.

2.4.2 Gateway

An IoT gateway is a physical device or software program that serves as the connection point between the cloud and controllers, sensors and intelligent devices.

All data moving to the cloud, or vice versa, go through the gateway, which can be either a dedicated hardware appliance or software program. An IoT gateway may also be referred to as an intelligent gateway or a control tier.

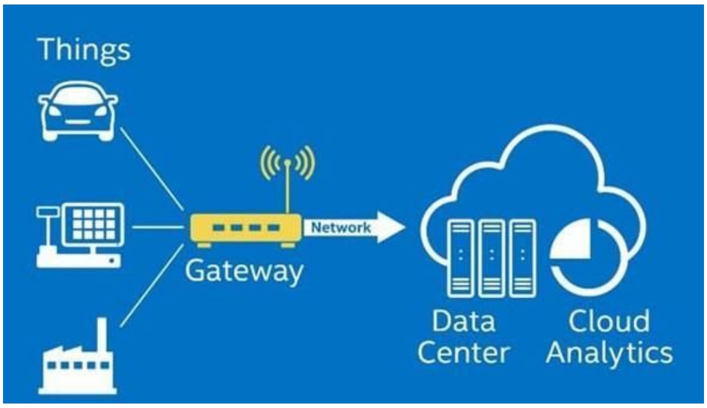


Fig. (2.2) Gateway function [2]

Some sensors generate tens of thousands of data points per second. A gateway provides a place to preprocess that data locally at the edge before sending it on to the cloud.

When data is aggregated, summarized and tactically analyzed at the edge, it minimizes the volume of data that needs to be forwarded on to the cloud, which can have a big impact on response times and network transmission costs.

Another benefit of an IoT gateway is that it can provide additional security for the IoT network and the data it transports. Because the gateway manages information moving in both directions, it can protect data moving to the cloud from leaks and IoT devices from being compromised by malicious outside attacks with features such as tamper detection, encryption, hardware random number generators and crypto engines [2].

2.4.3 Edge computing, fog computing and cloud computing

- **Edge Computing**

  Edge computing is a method of optimizing cloud computing systems by taking the control of computing applications, data, and services away from some central nodes (the "cloud") to the other logical extreme (the "edge") of the Internet which makes contact with the physical world.

  In this architecture, data comes in from the physical world via various sensors, and actions are taken to change physical state via various forms of output and actuators; by performing analytics and knowledge generation at the edge, communications bandwidth between systems under control and the central data centre is reduced. Edge Computing takes advantage of proximity to the physical items of interest also exploiting relationships those items may have to each other.

  In another word, edge computing is considered as a “mesh network of micro data centres that process or store critical data locally and push all received data to a central data centre or cloud storage repository.

  It is typically referred to in IoT use cases, where edge devices would collect data and send it all to a data centre or cloud for processing. Edge computing triages the data locally so some of it is processed locally, reducing the backhaul traffic to the central repository.

Typically, this is done by the IoT devices by transferring the data to a local device that includes computing, storage and network connectivity in a small form factor. Data is processed at the edge, and all or a portion of it is sent to the central processing or storage repository in a corporate data centre.

Edge computing allows data produced by the IoT devices to be processed closer to where it is created instead of sending it across long routes to data centres or clouds. [3]

Doing this computing closer to the edge of the network lets organizations analyze important data in near real-time a need of organizations across many industries, including manufacturing, healthcare, telecommunications and finance.

“In most scenarios, the presumption that everything will be in the cloud with a strong and stable fat pipe between the cloud and the edge device that’s just not realistic,” says Holder Antunes, senior director of corporate strategic innovation at Cisco.

Edge computing is characterized by its (Basic data visualization-Basic data analytics and short-term historian features-Data caching, buffering and streaming-Some data aggregation-Device to Device communications and M2M).

Some terms sometimes used to describe edge computing that it includes ‘fog’ computing while others see some differences between them telling about Fog computing as follows

- **Fog computing**

Fog refers to the network connections between edge devices and the cloud. Edge, on the other hand, refers more specifically to the computational processes being done close to the edge devices. So, fog includes edge computing, but fog would also incorporate the network needed to get processed data to its final destination.

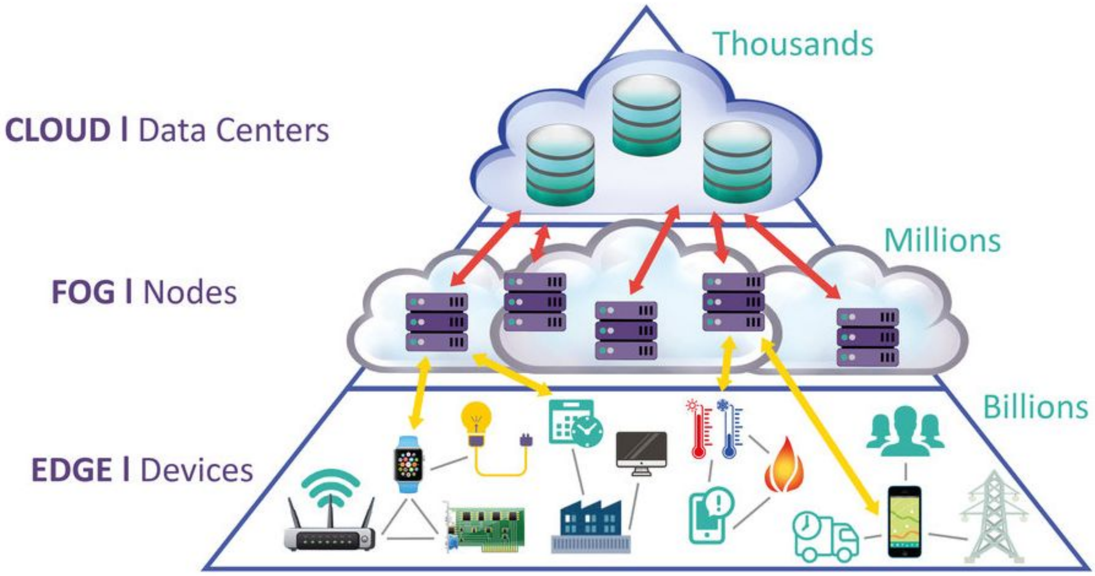


Fig. (2.3) IoT system architecture in a large scale.[4], [5]

Considering an industrial application where different kinds of sensors are used such as pressure sensors, flow sensors, and control valves to monitor an oil pipeline. The traditional cloud computing model would send all the readings to the cloud, analyze them using machine learning algorithms to detect abnormalities, and send appropriate fixes down to the end devices.

But in fact, not all of these reading should be sent to the cloud because some of these data are very critical that they must not need much that time to be transferred to the cloud to be analyzed and stored and to decide an action to be performed according to these readings. In the time it takes to send an abnormal reading from a sensor, categorize as a potential leak in the cloud, and send a downlink message to notify personnel or stop the flow, the leak might have turned into a major spill. So, in that case, using cloud computing is not a proper method to deal with these kinds of serious readings.

While on the other hand with a fog computing infrastructure, sensors will send data to inexpensive local fog nodes where abnormality detection can happen locally and send commands to shut off the leaky valves within milliseconds, instead of minutes. This example illustrates how fog nodes can extend the role of the cloud down to a fog level for added benefit.

Fog works with the Cloud, while the Edge by definition excludes the Cloud. Fog works hierarchically, whereas Edge is limited to a small number of layers. Besides calculation, Fog also deals with networking, storage, control and acceleration. An interesting and inexpensive intermediate solution to the Cloud can be created with Fog Computing. With Fog, computing power is pushed to the periphery of the networks and is executed for a group of "Things" between Edge and Cloud.

• **Cloud computing**

Cloud computing is the delivery of computing services- servers, storage, databases, networking, software, analytics and more over the Internet ("the cloud"). Companies offering these computing services are called cloud providers and typically charge for cloud computing services based on usage, similar to how you are billed for water or electricity at home.

Organizations nowadays are turning to cloud computing services for some reasons such as:

1. **Cost**
   Cloud computing eliminates the capital expense of buying hardware and software and setting up and running on-site data centres, the racks of servers and the IT experts for managing the infrastructure.

2. **Speed**
   Most cloud computing services are provided self-service and on demand, so even vast amounts of computing resources can be provisioned in minutes, typically with just a few mouse clicks, giving businesses a lot of flexibility and taking the pressure off capacity planning.

3. **Global scale**
   The benefits of cloud computing services include the ability to scale elastically. In cloud speak, that means delivering the right amount of IT resources -for more or less computing power, storage, bandwidth—right when it's needed and from the right geographic location.

4. **Productivity**
   On-site data centres typically require a lot of "racking and stacking"-hardware setup, software patching and other time-consuming IT management chores. Cloud computing removes

the need for many of these tasks, so IT teams can spend time on achieving more important business goals.

5. **Performance**
   The biggest cloud computing services run on a worldwide network of secure data centres, which are regularly upgraded to the latest generation of fast and efficient computing hardware. This offers several benefits over a single corporate data centre, including reduced network latency for applications and greater economies of scale.

6. **Reliability**
   Cloud computing makes data backup, disaster recovery and business continuity easier and less expensive because data can be mirrored at multiple redundant sites on the cloud provider's network.
   Cloud computing is characterized by its (complex analytics-Business logic-Big Data mining-Machine learning rules-advanced visualization-long term data storage).

Note

Edge computing, fog computing and cloud computing are all complementary architectures that come together to create powerful IoT platform where one does not replace the other. The best IoT solution for any business based on the technology of IoT will likely be a mix of the three architectures. Deciding which computing tasks should happen in the cloud, in the fog or at the edge requires careful analysis of the business needs. [6]

2.4.4 IoT Cloud Server

The IoT servers have different purposes, like administration, monitoring, data gathering and analysis. They are completely modular, based on open-source enterprise platforms that provide the capabilities that are needed for the server-side of the IoT architecture, to connect to devices. The data that is transmitted through the server gateway is processed and stored securely using big data analytics. This data is then used when performing intelligent actions on devices, in essence, making them smart.

Reliability is another aspect of IoT servers, being vastly more reliable than traditional servers due to the sheer availability of servers. If and when there are problems with one server, the resources will automatically be shifted to another server which will take the place of the malfunctioning one.

So because activities like storage and data processing take place in the cloud rather than on the device itself, this has had significant implications for IoT.

Many IoT systems make use of large numbers of sensors to collect data and then make intelligent decisions.

Using the cloud also allows for high scalability. When you have hundreds, thousands, or even millions of sensors, putting large amounts of computational power on each sensor would be extremely expensive and energy intensive. Instead, data can be passed to the cloud from all these sensors and processed therein aggregate.

For much of IoT, the brain of the system is in the cloud. Sensors and devices collect data and perform actions, but the processing/commanding/analytics typically happens in the cloud.

Cloud storage services may be accessed through a co-located cloud computer service, a web service application programming interface (API) or by applications that utilize the API, such as cloud desktop storage, a cloud storage gateway or Web-based content management systems.

2.5 Hardware in use

In this project, it is aimed to show an application based on the main components of any complete IoT system knowing that any system to be characterized as a complete IoT system needs:

1. **Hardware**
   Such as sensors or devices. These sensors and devices collect data from the environment (e.g. state of parking slot-concentration of fuel gases in the air-the parking car information) or perform actions in the environment according to this data (e.g. a led is on once car parked in a slot).

2. **Connectivity**
   The hardware needs a way to transmit all that data to the cloud or needs a way to receive commands from the cloud. For some IoT systems, there

can be an intermediate step between hardware and connecting to the cloud, such as a gateway or router. This connectivity can be in a form of cellular, satellite, Wifi, Bluetooth, RFID, NFC, LPWAN, or Ethernet connections.

The perfect connectivity option would consume extremely little power, have a huge range, and would be able to transmit large amounts of data (high bandwidth) but unfortunately, this perfect connectivity doesn’t exist. Each connectivity option represents a tradeoff between power consumption, range, and bandwidth and the proper choice among them depends directly on the nature of each IoT system.[8]

In the scope of connectivity for this project, two types of connectivity were chosen based on their power consumption, range, and bandwidth they provide and how these parameters make a kind of connectivity is more preferable than another one.[9][10]

- **Cellular**

Cellular connectivity is known for its high Power Consumption, High Range and High Bandwidth it provides. But despite these characteristics, it was chosen to be used for the mobile application downloaded in each user cell phone to be his interface with the system. It was chosen for the connectivity between the application and the cloud for some reasons such as:

The availability of cellular (3G/4G) towers in any part of a town. That makes the application connect easily with the cloud server in anywhere without getting worried to be in a specific area which has a 3G/4G connectivity.

Since the car driver searching for a parking slot is moving with his car in different random spots, then this connectivity type is a good choice for his application be connected with the cloud server.

The small size of data fetched by the mobile web application, in cellular connection, high power consumption and bandwidth are all consequences of using applications which are used to push or fetch a huge amount of data such as in case of watching or streaming videos. But in the IoT system application, a small amount of data is pushed to the application in an intelligent manner by the server which does its duty to discard any useless data and transmit the only useful and critical data the application. And the application itself is designed to do some actions caused by transmitted a few command signals to the server which does not a high power to consume or a high bandwidth to transmit these command signals through.

- **Wifi:**

Wifi connectivity has the same high bandwidth as the cellular connectivity but unlike cellular, it is characterized by its low range and power consumption. In the project, Wifi is used by the gateway through which the data is transmitted from it to the cloud server. Despite its small range, it is usually used in the steady place in which many devices can be connected to the internet as long as they have existed in this place. And that is why it is used in a place like a car parking garage where all sensors are implemented and connected to the gateway to transmit any measured data to it and then the gateway will use Wifi connectivity as a path through which it will transmit the measured data to the server.

It also consumes a small amount of power which makes it an economical connectivity type especially for any environment having a steady gateway which is supposed to work all the time.

3. **Software**

This software is hosted in the cloud and is responsible for analyzing the data it's collecting from the sensors and making decisions. This software is considered as the brain of the system which receives all the important information from the gateway, analyzes it, stores it in the database and make a different kind of decisions according to the received data by sending commands to the actuators in the environment. This software is supposed to be executed all the time and do its work by itself without any interference from the human in order to achieve one of the most main purposes of IoT which is Automation.

4. **User interface**.

To make all of these components useful, there is a need to a provided way for the users to interact with the IoT system (e.g. a mobile web application with a dashboard that has some kind of connectivity with the system server in order to show the updated data in real time and allow users to choose some action to be performed according to their views based on this data).

In the scope of hardware, it is prefered to use friendly and easily used devices as it is required to have a complete prototype working reliably with accepted efficiency without taking much care of using more complex devices as complexity was not in the scope of the project.

2.5.1 Raspberry Pi

Raspberry Pi is a small, single-board computer that was originally developed for computer science education and has since been popularized by digital hobbyists and makers of IoT devices.

In this project, recently launched Raspberry Pi 3 is used, included built-in WiFi and Bluetooth making it the most compact and standalone computer. Based on a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor and 1GB RAM, the Pi is a powerful platform. The Raspberry Pi 3 is also equipped with 2.4 GHz WiFi 802.11n and Bluetooth 4.1, in addition to the 10/100 Ethernet port. The HDMI port makes it further easy to hook up A/V sources.

There are different popular development boards which are used in the IoT applications and prototyping such as (Arduino Uno-Beagle Board-particle photon- Udoo Neo and others), but Raspberry Pi is chosen in this project as it has a very wide internet community so that it can be a good reference in searching for it in case of having some technical problems in interfacing and connection. Also, Raspberry Pi, as previously said, is a small computer with a high-speed CPU (1.2 GHz) and with this speed it can handle many tasks efficiently like gathering data from the different sensors, pushing it to the cloud and taking actions according to this kind of data.

2.5.2 IR Sensor

Infrared sensors consist of two elements: an infrared source and an infrared detector. Infrared sources include an LED or infrared laser diode. Infrared detectors include photodiodes or phototransistors.

The energy emitted by the infrared source is reflected by an object and falls on. When the IR transmitter emits radiation, it reaches the object and some of the radiation reflects back to the IR receiver. Based on the intensity of the reception by the IR receiver, the output of the sensor is defined. According to this operation, IR sensor is used to act as an object detection sensor.

Noted that the IR rays should be projected to the object with an inclined angle in order to be reflected from the object to the IR receiver while if they are transmitted in a perpendicular direction to the transmitter surface,the rays will be reflected to the IR transmitter and the receiver will not sense the transmitted IR rays. See figure (2.5).

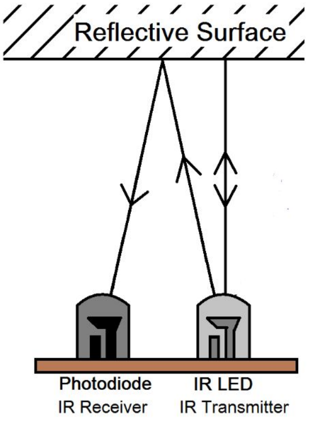


Fig. (2.4) Transmitted IR rays in the inclined and perpendicular angles

In this project, this type of sensors is used in each parking slot to detect the availability of any slot to park in it. According to the parking status measured by the sensor, it will send a data signal to the gateway (Raspberry Pi) in order to push it to the cloud.

2.5.3 RFID (RC522)

RFID uses electromagnetic fields to automatically identify and track tags attached to objects. The tags contain electronically-stored information. Passive tags collect energy from a nearby RFID interrogating radio waves. Active tags have a local power source (such as a battery) and may operate hundreds of meters from the RFID reader. Unlike a barcode, the tag need not be within the line of sight of the reader, so it may be embedded in the tracked object. RFID is one method for Automatic Identification and Data Capture (AIDC).

In this scenario, it is assumed that every parking car has already tag attached to its body so once it enters the garage, the RFID reader will capture the data stored in the car tag. This data is supposed to include any important information about the car and its driver such as (the car plate number-car colour-car driver name-car driver license number).

All of this data will be transmitted immediately to the garage controller (Raspberry Pi) which acts as a gateway to gather all the needed data to send it wirelessly to the cloud to be stored in the virtual required database.

2.5.4 Smoke detector (MQ-2)

The Grove-Gas Sensor (MQ2) module is useful for gas leakage detection (home and industry). It is suitable for detecting H2, LPG, CH4, CO, Alcohol, Smoke or Propane. Due to its high sensitivity and fast response time, measurement can be taken as soon as possible.

Fig. (2.5) Smoke detector

The sensitivity of the sensor can be adjusted by a potentiometer.
This sensor is used as an additional sensor that can be implemented in any IoT garage for the purpose of protection and fire incident avoidance. As it previously said, it can measure the concentration of any fuel gas derivatives in the air so that if its concentration exceeds a certain safety value then it can figure a sign of fire and trigger a signal to the garage controller (Raspberry Pi) as a fire alarm to tell the garage supervisor that there is a fire in the garage to let him deal immediately with it.

2.5.5 Raspberry Pi Role as a Garage Controller and a Gateway

In the prototype, Raspberry Pi is supposed to acts as regular garage controller which handles any required tasks in the operation of the garage as lighting green led at each empty parking slot, lighting red led at each filled slot and takes the initial precautions needed in case of the fire alarm.

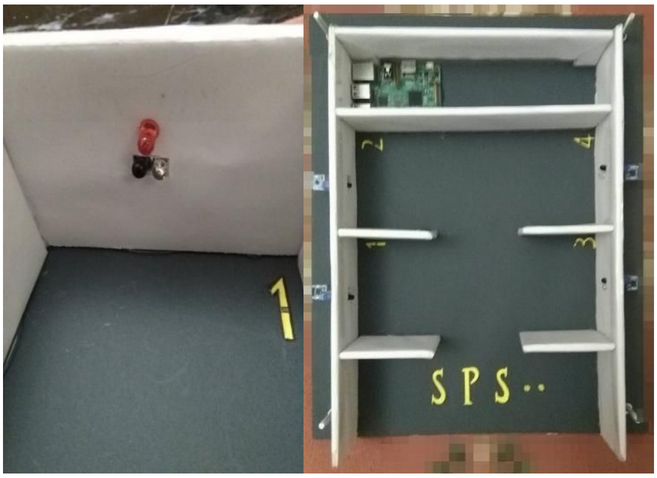


Fig. (2.6) IoT garage prototype

It also should act as a central data gathering point to which all the data from (IR sensor-RFID-smoke detector) should be transmitted. This central point should act as an IoT gateway which after collecting the data will do some analytic operation on it in order to send only the needed important data to the virtual database in the Cloud.

Raspberry code is written in python language in the main code to deal with the interfacing tasks with the different sensors using General Purpose I/Os (GPIOs) and SPI library, collecting data from them and pushing the data wirelessly to the database using request HTTP library.
Code for the Raspberry Pi is in the appendix (A) page (99).

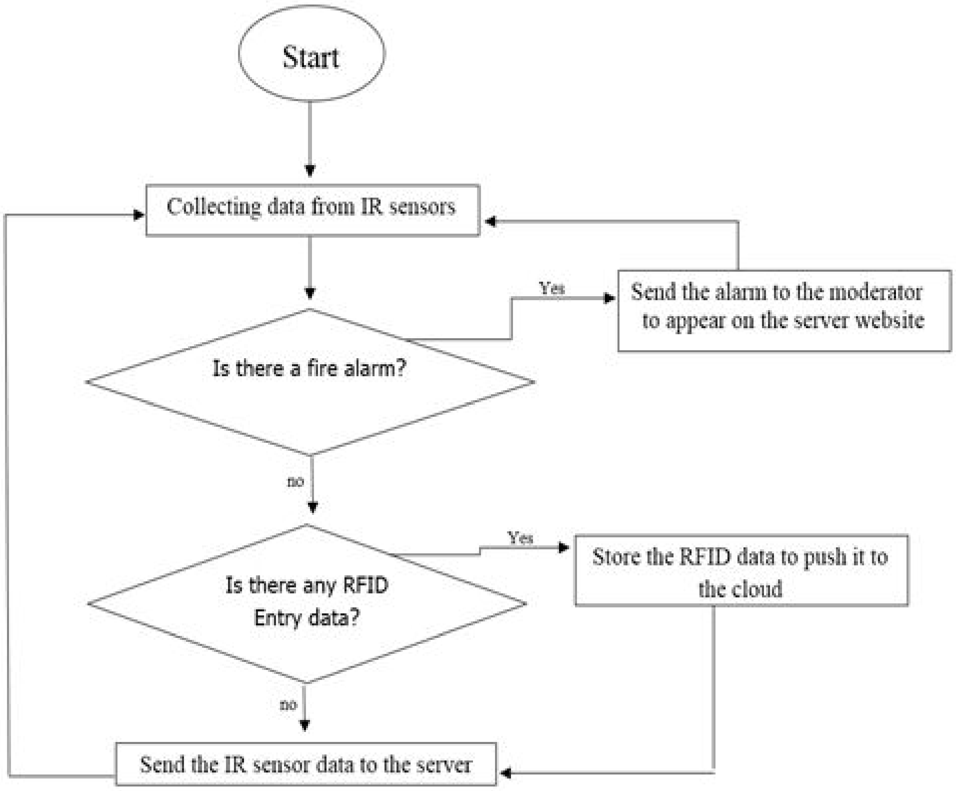

Start
Collecting data from IR sensors
Yes
Send the alarm to the moderator to appear on the server website
Is there a fire alarm?
no
Yes
Store the RFID data to push it to the cloud
Is there any RFID Entry data?
no
Send the IR sensor data to the server

Fig. (2.7) flow diagram of the system

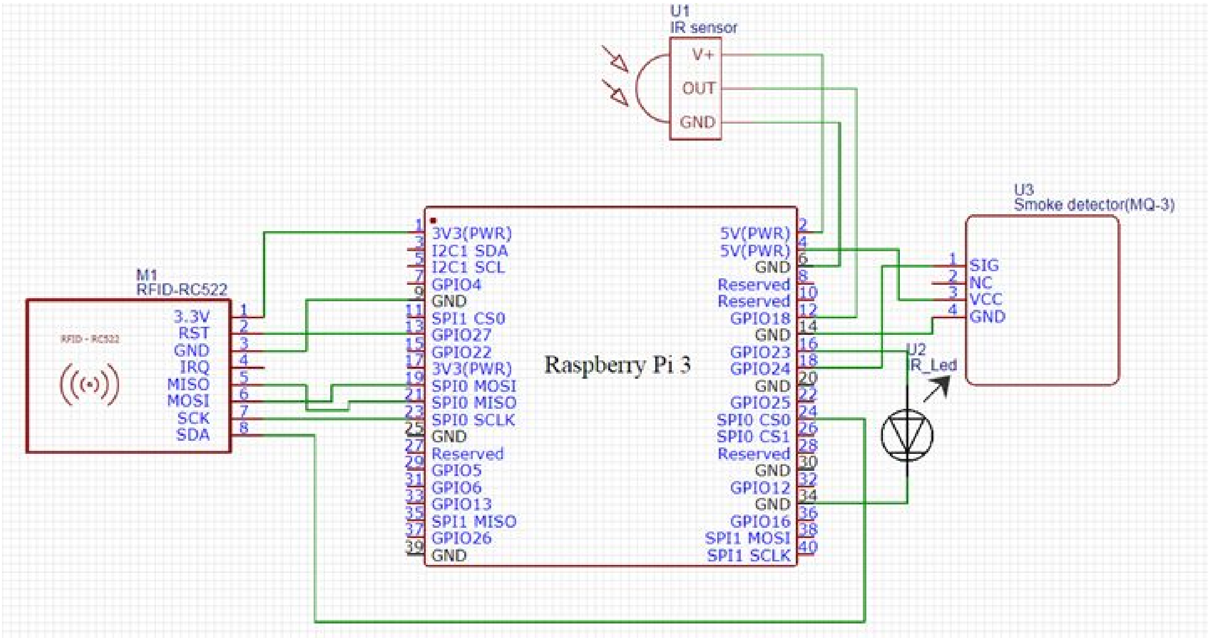


Fig. (2.8) Simplified schematic connection for the Raspberry Pi (the garage controller)

2.5.6 Server and database

A Server is a computer, a device or a program that is dedicated to managing network resources. Servers are often referred to as dedicated because they carry out hardly any other tasks apart from their server tasks.

There are a number of categories of servers, including print servers, file servers, network servers and database servers.

In theory, whenever computers share resources with the client machines they are considered servers.

Nearly all personal computers are capable of serving as network servers. However, usually, software/hardware system dedicated computers have features and configurations just for this task. For example, dedicated servers may have high-performance RAM, a faster processor and several high-capacity hard drives. In addition, dedicated servers may be connected to redundant power supplies, several networks and other servers. Such connection features and configurations are necessary as many client machines and client programs may depend on them to function efficiently, correctly and reliably.

In order to operate in the unique network environment where many computers and hardware/software systems are dependent on just one or several server computers, a server often has special characteristics and capabilities, including:

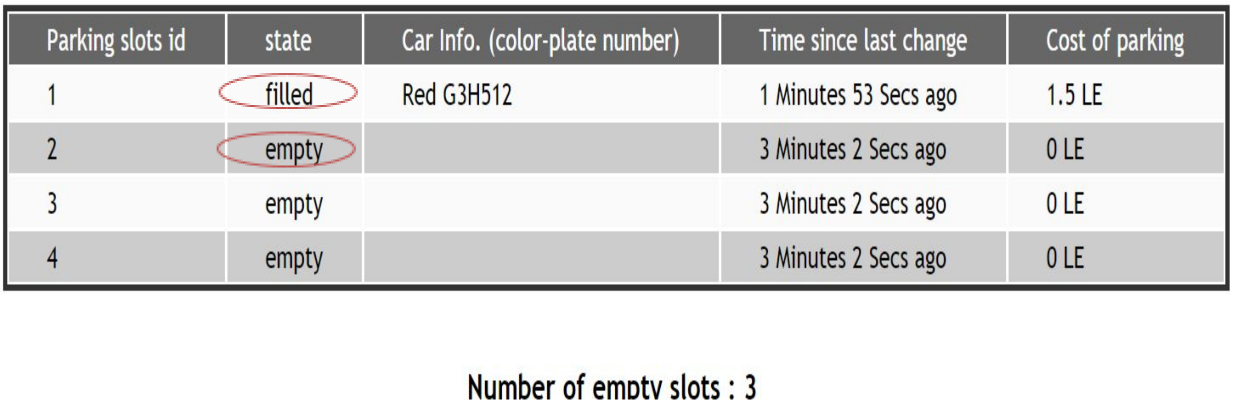
- The ability to update hardware and software without a restart or reboot.
- Advanced backup capability for frequent backup of critical data.
- Advanced networking performance.
- Automatic (invisible to the user) data transfer between devices.
- High security for resources, data and memory protection.

Server computers often have special operating systems not usually found on personal computers. Some operating systems are available in both server and desktop versions and use similar interfaces. However, an increase in the reliability of both server hardware and operating systems have blurred the distinctions between desktop and server operating systems.

000webhost is used as a free platform which provides the facility to make our own accessible website accompanied by a dedicated database for data storage.

The aim of this website is to act as a display of the garage for the garage coordinator which continuously monitors its parking slots state and update the coordinator with any change. This kind of information is received wirelessly from the garage controller (Raspberry Pi) which already fetched this data from the surrounding IR sensors in the parking area.

Table (2.1) Representation of the slot states (empty/filled)



Parking slots id	state	Car Info. (color-plate number)	Time since last change	Cost of parking
1	filled	Red G3H512	1 Minutes 53 Secs ago	1.5 LE
2	empty		3 Minutes 2 Secs ago	0 LE
3	empty		3 Minutes 2 Secs ago	0 LE
4	empty		3 Minutes 2 Secs ago	0 LE

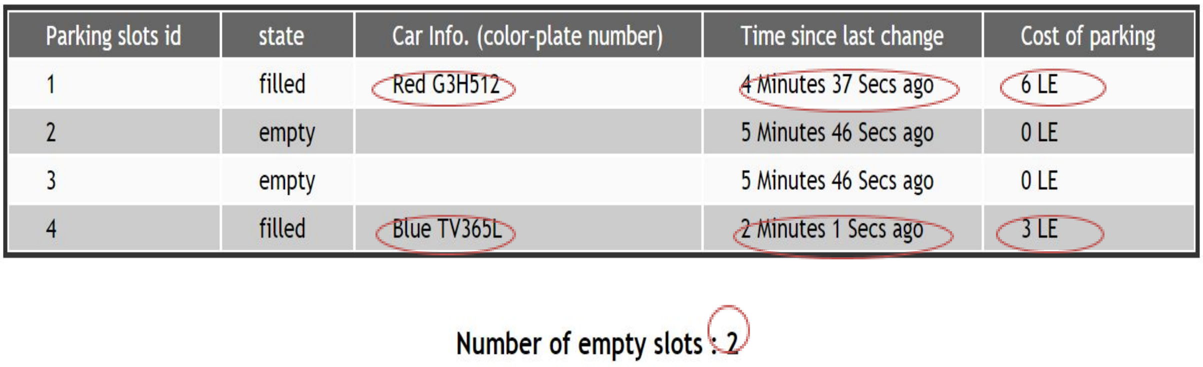
Number of emptv slots : 3

It also shows some important information about the parking car and its driver It

It also shows some important information about the parking car and its driver such as (car colour-car plate number-car driver license number). This kind of information is received wirelessly from the garage controller (Raspberry Pi) which already fetched this data from the RFID reader at the entrance of the parking area.

Besides, it counts the time spent by the car in its slot until it gets out from the garage and according to this time, it calculates the cost which should be paid for the parking and notify the driver with a coordinator with it to receive the payment from the driver when he leaves.

Table (2.2) Representation of the parking car information, time spent in the slot and the cost of parking



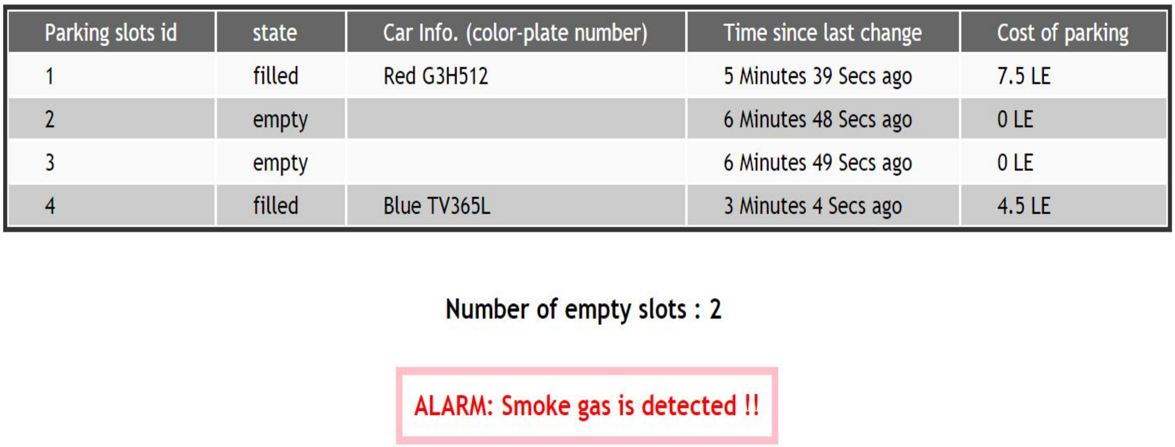
Parking slots id	state	Car Info. (color-plate number)	Time since last change	Cost of parking
1	filled	Red G3H512	4 Minutes 37 Secs ago	6 LE
2	empty		5 Minutes 46 Secs ago	0 LE
3	empty		5 Minutes 46 Secs ago	0 LE
4	filled	Blue TV365L	2 Minutes 1 Secs ago	3 LE

Number of empty slots : 2

This server also can act as an alarm display to notify the coordinator of any possibility for a fire to occur. This kind of information is received wirelessly from the garage controller (Raspberry Pi) which already fetched this data from the smoke detector implemented in the parking area to measure the concentration of fuel gases in the air and to trigger the Raspberry Pi to send an alarm signal to the server if the fuel gas concentration has exceeded a certain value.

Therefore, once the garage moderator is notified from the server about a fire alarm, he will immediately make the required procedures to eliminate the fire.

Table (2.3) Declaration of a fire alarm when a fuel gas is detected



Parking slots id	state	Car Info. (color-plate number)	Time since last change	Cost of parking
1	filled	Red G3H512	5 Minutes 39 Secs ago	7.5 LE
2	empty		6 Minutes 48 Secs ago	0 LE
3	empty		6 Minutes 49 Secs ago	0 LE
4	filled	Blue TV365L	3 Minutes 4 Secs ago	4.5 LE

Number of empty slots : 2

ALARM: Smoke gas is detected !!

The code of the server depends mainly on three separate codes. Each one has a certain task to perform and they cooperate together at the same time to make the server do what is supposed to do. These tasks are as follows:

1. The first code is the essential code for any connection needed between the server and its database upon executing the SQL which deals directly with the data in the database in different operations like pushing it to the database, fetching it from the database and any other analytic operation performed on this kind of data.

This code is written in PHP and SQL languages. The code is in Appendix B page (101)

2. The second code is the main server code which is supposed to receive the data from the gateway, store it in the database and make some analytic operations on it in order to be displayed on the web page for the garage in an appropriate way by giving him only the important parking information needed to be known such as (the state of each parking slot-parking car info as car plate number and its number-the time spent by the parking car in the slot-the cost should be paid according to the parking time-number of empty slots-The fuel gas concentration in the garage).

This code is written in PHP and SQL languages. The code is in Appendix C page (101).

3. The third code is code which is responsible to represent the data extracted from the second code in a form of a table with a special design made by some lines of CSS languages. This table appears in a webpage which is periodically updated every 5 seconds to cope with the continuously transmitted data from the gateway in order to the represent it immediately to the garage coordinator.

This code is written in PHP, CSS and HTML and it is in the Appendix D page(103).

CH.3

ANDROID APPLICATION

3.1 Android

An android application is a mobile software application developed for use on devices powered by Google's Android platform. Customers in today's world are on the move and they're using mobile application platforms to get there. Whether they use mobile phones, tablets, or other mobile devices they have all the information they need. We live in an era where information is readily available no matter where we go, thanks to mobile phones and the proliferation of apps for every possible need or interest. Mobile applications are more accessible and easier to use. Smartphones are everywhere, as are tablets. at the end of 2015, the Pew Research Center reported that 68% of Americans have smartphones and 45% have tablet computers.

3.2 Application software development

Applications are usually developed in Java, C++ or Kotlin programming language using the Android software development kit (SDK), Third party tools, development environments and language support have also continued to evolve and expand since the initial SDK was released in 2008.

There is many tools to develop an Android application , here it has been used Android studio SDK. Building an Android application comes down to two major parts Java and XML. Java is the language used in Android, but the Android part encompasses learning XML for the design of the application , learning the concepts of Android, and using the concepts programmatically with Java. Thus, our application is divided into two main parts:

1. The graphical user interface (GUI) using extensible markup language (XML) it is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. This part is considered the window for the user to interact with the application.

2. The dynamic part through which there is a response and interaction between the user and the application also the objects, threads, functions, intents and syncs interactions in the application it self this is done using Java, it is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, the language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of

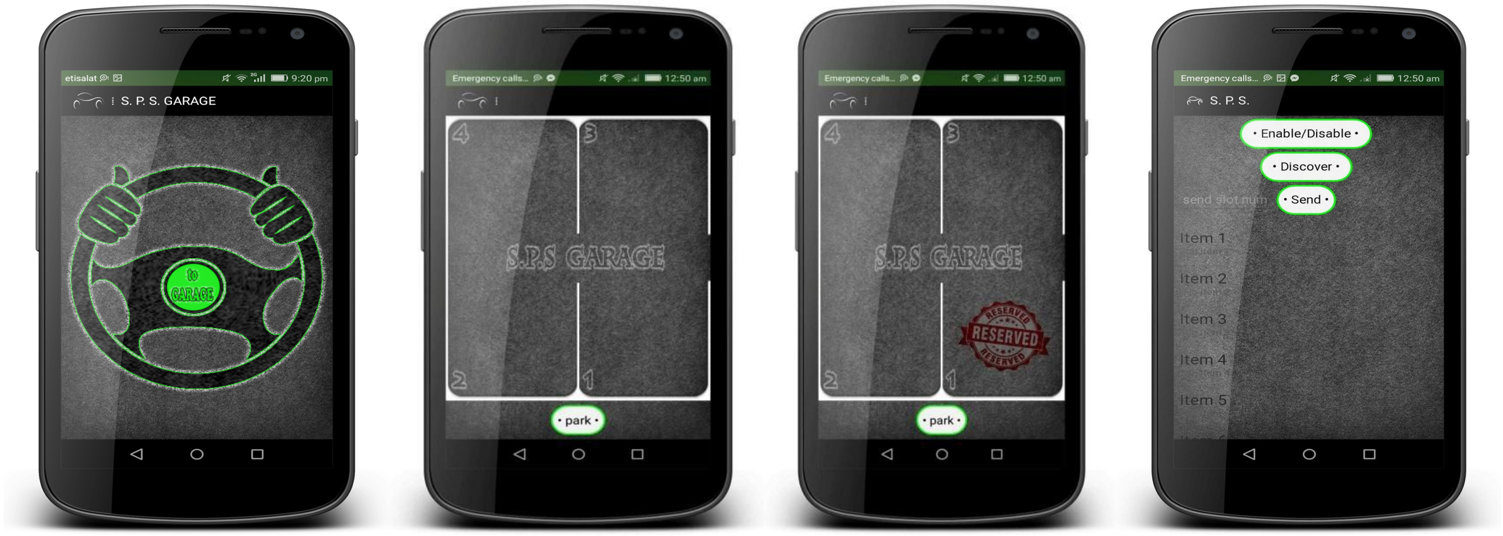
them, it is needed to write with C++ when interactions are needed with the phone memory. This part is the brain of the application, it contains all data and logic of the application, actually it’s where all the magic happens.

3.3 SPS Android application

“Smart Parking System using Android Application” provides user an easy way of checking the parking slots through an application, choose an empty slot and send its number to the car to auto-park. To avoid the problem of traffic conjunction that unnecessarily consumes time, in this application the user can view various parking slots and check for their availability. Whenever a parking slot is occupied it will be simultaneously marked with a “Reserved” sign in the android application. Hence this application reduces the user’s effort and time of searching the parking slot and also avoids conjunction of traffic.

3.4 SPS user interface

- Layout 1 (to Garage): This is the layout that opens when the user click on the application icon, it contains a button view that takes the user to the next layout “layout 2”, as shown in figure (3.1).
- Layout 2 (Garage simulation 1): This part’s user interface shows the garage slots status (empty/occupied), when one slot -or more- is occupied at the garage, a red ‘reserved’ sign will appear simultaneously on this slot in the application as shown in layout 2 (Garage simulation 2), figure (3.1) this is what is seen when slot one is occupied for example.
- Layout 3 (Bluetooth): this layout is mainly in charge of the connection between the user and the car (TivaC), the user needs to communicate with the car to send it an empty slot number to auto-park in, this connection is done through this layout.



(a)Layout of the first activity (to Garage)

(b)Layout of the second activity (Garage simulation 1)

(c)Layout of the second activity (Garage simulation 2)

(d)Layout of the third activity (Bluetooth)

Fig. (3.1) A screenshot of the application activities

3.5 Programming procedure

- Programming the GUI , this part is done using XML also Android Studio provide a more simple way to program the GUI like drag and drop some items , but it is better using XML it provides more features and better control , simply to design such an app all the images used -the background, the red reserved sign...etc- should be saved in the project file called res/drawable, using XML a Layout which will contain whatever the app has will be defined, everything the application contains -backgrounds, buttons… etc- is defined till the app is ready to make it interactive and this is done through declaring some functions to make specific objects interactive for example to make the app close when a button is pressed,this should be defined in the xml file using onClick, this will declare a function in the java file corresponds to this xml file where we can write the code for this function to close the app when it’s clicked.

For SPS application there are three main .xml files: activity_main1.xml, activity_main2.xml and activity_main3.xml and three relevant .java files: Main1Activity.java, Main2Activity.java and Main3Activity.java they are in charge of making the application interactive as mentioned previously, now let’s focus mainly on how the GUI design was made (.xml files):

1) Activity_main1.xml

   This .xml file if responsible for creating layout (a) shown in figure (3.2) this is the first layout the user views when they click on the application icon, this layout has only a button view shaped as a car steering wheel, when this button is clicked its color is changed from green to red then the second layout opens immediately. to make this two car steering wheel images -green and red- were added in the application drawable folder and to make the button color change when it is clicked this is done through some instructions written in a sub .xml file called imagebutton.xml.

2) Activity_main2.xml

   This .xml file if responsible for creating layout (b) shown in figure (3.2) this is the second layout, it’s considered the main project layout where all the work is done. it mainly consists of a group of LinearLayouts that’s divided into a group of ImageViews and one button view. In general a layout defines the structure for a user interface in any application, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with -the manner of this interaction is specified through the .java file relevant to the .xml file being worked in- Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects. The View objects can be one of many subclasses, such as button, TextView or ImageView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout.. etc.

The most important views in this layout are the four ImageViews on the four parking slots; each one of these ImageViews is given an id so they can be called in the .java file to change its slot status from filled to empty and vise versa simultaneously with the garage. And the button view that will open the Bluetooth layout -will be explained later- when clicked.

3) Activity_main3.xml

This layout interfaces the parking operation through application with Bluetooth, it mainly contains:

- three Buttons:

1- Enable/disable button to enable the Bluetooth or disable it

2- Discover button to discover all the nearby and visible Bluetooth devices

3- Send button that passes the slot number -the user texts- to the output Stream buffer sending it to the other device.

- A clickable listview called device_adapter_view, this is the view at which the nearby devices will be shown, and on clicking on a device pairing and connection with it will happen.

- A textView at which the user enters the slot number.

4) Additional details:

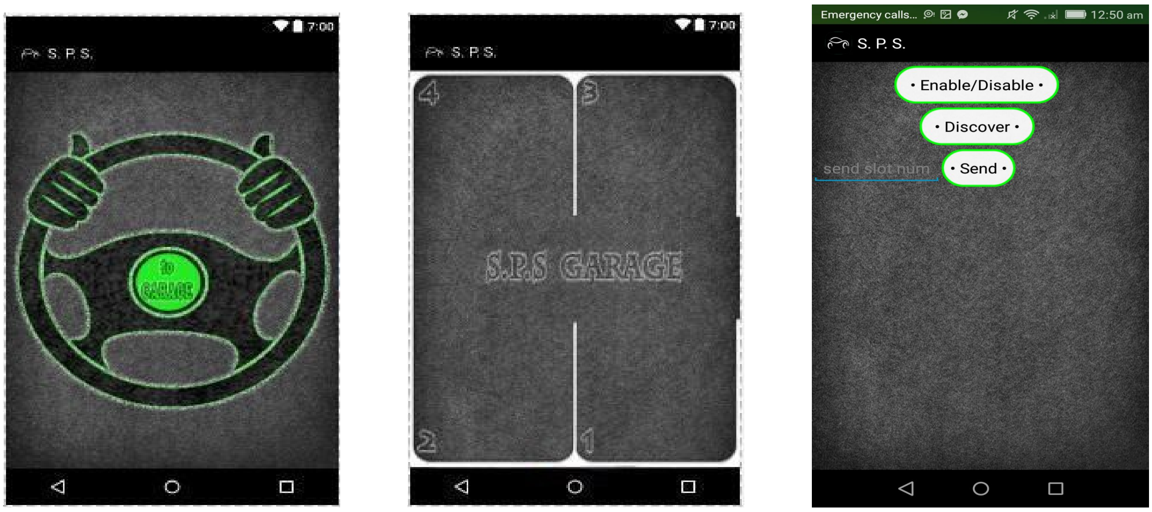
Other design details were edited through sub .xml files, .java files and other methods in Android studio, for example:

- The action bar design:

Its color was changed through some instructions in styles.xml file, its title and logo were changed through some instructions in main_activity.java files.

- The application icon:

It were edited through the Image Asset in the app folder and its label was edited through an instruction in AndroidManifest.xml file.



a- Activity_main1.xml layout

b- Activity_main2.xml layout

c- Activity_main3.xml layout

Fig. (3.2) A screenshot of .xml files layouts

- Now the interactive part of the application , How to access the internet from the application ? , the project for example when using the internet a permission must be added asking the user to allow accessing the internet from his phone , then the developer will add some libraries to use when coding, there are many libraries which make it much easier for developers , in this project one library called Ion is used, this library makes it easier to fetch data from internet .

  So after adding the permissions and the libraries begin to declare each object you’re going to use, then initialize this variables or objects in the onCreate function , onCreate function is one of the life cycle functions of the application , this function is called when the application opens for the first time .

  Also those objects can be declared in any function you write as long as it’s a field variable .how to get from one Activity to another whenever we click on something -button for example- receiving and sending data… etc? all these stuff can be done here with Java programming Language, first some permissions should be added to the Manifest file in

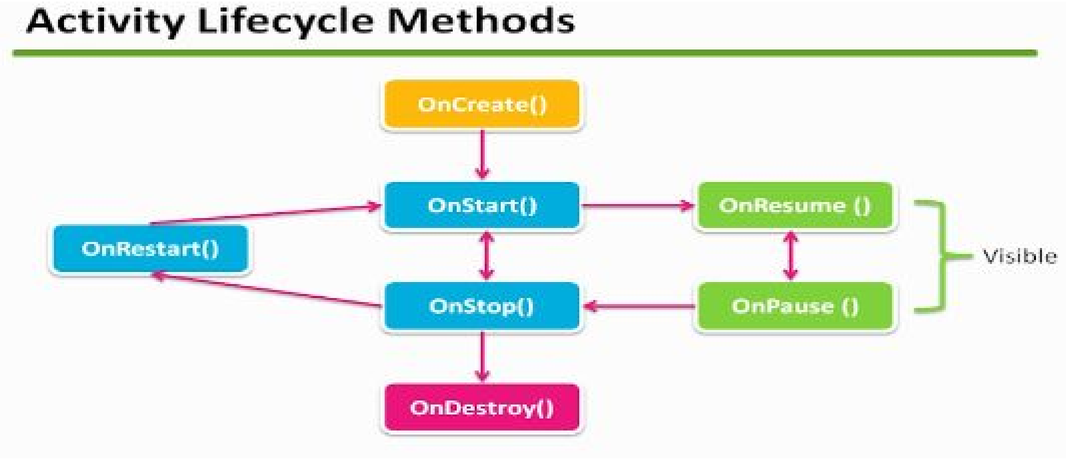


Fig. (3.3) Activity Lifecycle methods [20]

As shown in figure (3.3) the life cycle of the activity this part will focus mainly on three of this function onCreate() , OnResume() and OnPause() , OnPause is called when you get out from the application for a while like when you swap between applications that’s where the application is paused here some variables need to be saved or some data so as not to lose it or rebuild it again here a function onSaveInstanceState (Bundle bundle) is used, this function takes one argument which is Bundle , Bundles are generally used for passing data between various Android activities. It depends on what type of values you want to pass , here the states of the parking slots are saved whether they are empty or filled . now when you back to the application after pausing it onResume() is called so the bundle which has all the saved data which was passed here to restore, in our case this data is the parking slots states writing onRestoreInstanceState (Bundle bundle) method.

Fetching data from internet is done through APIs ,Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets, which also is a Service Level Agreement (SLA) to specify the functional provider and expose the service path or URL for its API users , so instead of interacting with details of server or a web page we need only its API for certain information for example an API for slots state which we can know if it is empty or filled , so in the application we just send a request and have a response this what is called API simply the response is a file formatted in XML or in JSON , here JSON is used and simply JSON is just an objects inside each others , using Ion library we could fetch the data from the server which the Raspberry pi sent the data to, there are four slots each slot has its ID number and its state.

Java provides many features like initializing a JSON object which will contain the response API, get the value of the state which is string of "empty" or "filled" and the ID of the slot which is int from 1 to 4 , now we can set each slot to its corresponding state.
The responded JSON file which is the response from the web may be a null object, null means NOTHING it indicates that the value reference to nothing , here in our code if the null case wasn't handled the application will crash so a NullPointerException was added, it will handle this case logging to the monitor that there is a null , also in this case nothing in the application have to be changed like the slots or states everything remains as its before simply we ignore the null when it is received.

Finally for the app to be auto refreshed and always updated the application contains a handler the main thread of the android application handles the operations and the events done through the UI, but in this case synchronization of the refreshing task with the main thread is needed thus a handler is declared and write the refreshing code inside the runnable method which is called every 5000 milliSeconds as it is specified.
Also there is a refresh button which is set to refresh and get the data when clicking on it.

To make the car park the TivaC controller do this in another way it could be easier for the user to just send the slot number which is empty from the mobile application to the car (Tiva C), as the data is small (just an int) representing the number of slot. As shown in figure (3.3) (b) there are only 4 slots , so it is efficient to send this data with Bluetooth, first interfacing the Bluetooth module with the Tiva C (it is well explained in its part) then programming the application to send data with Bluetooth.

3.6 Programming the application with Bluetooth

1) Create a layout which will contain the buttons textViews and any GUI needed like enable/disable button, discover button, send Button and a TextView to write the slot number will be sent, all of this is done using XML as explained before.

2) There are many libraries to implement and code the Bluetooth application but hardly to find projects that operated Bluetooth application with Tiva C , so in this case it's better to understand how it works , with the help of Android developers official page provided by Google . Bluetooth works somehow like a finite state machine, to start making operations and coding with Bluetooth a BluetoothAdapter object have to be declared, with this BluetoothAdapter the application can enable disable Bluetooth in this device with respect to having intents and broadcast receivers, intents are like delivery guy which going to deliver a message to the user or pass a message from an Activity to another while the BroadCastReceiver is something to catch the changes that will happen as you tell the application to do something on changing the Bluetooth state for example.

Discovering devices and pairing with them, simply BluetoothAdapter.StartDiscovery() will make the Bluetooth discover all the nearby devices but if the Android version is more than lollipop you should first check some permissions if you don't know this you may waste months not knowing what's going on!
once Bluetooth discovers a device it will show this device address and name in the screen through a clickable layout (device_adapter_view) which is a developer made layout we just copied it to the application project.

Now when clicking on a device it should pair with it then create the RF communication channel it's like hand shaking or establishing the channel at which the app and the other device will communicate in our case it is a TivaC and it's always in a server mode let's explain this, the Bluetooth works in threads or states it's in the accept thread waiting for any paired device to connect to it , after connection it is a connected thread at which the communication occurs taking inputs from input stream and passing outputs through output stream it's like a buffer or a register, as mentioned the Tiva C is programmed to be in a server mode waiting for some device to connect to it , for the communication to occur a bluetooth Socket is needed then create the RF channel with the other device UUID , UUID is something like an address but it is related to software and a single device may have more than one UUID then we call btSocket.connect() which will connect the two devices to each other now the connected thread could be started at which the communication will occur writing data to the buffer and sending it through the channel to the other device.

“surrounding any part of code with a try catch is to prevent the application from crashing informing the developer what’s the error if happened “

CH.4

SELF-PARKING ROBOT

4.1 Chapter Objectives

- Experiencing the needed hardware.
- Interfacing with launchpad peripherals.
- Demonstrating a self-parking robot.
- FSM algorithm implementation.

In this chapter, we are going to demonstrate a simulation to a Self-Parking Robot using basic concepts of robotics such as Direct Current (DC) motors and servo motors driving according to specific inputs, these inputs serving different situations of the robot position in a garage. Inputs are being served from Ultrasonic Sensors, detecting empty slots of a garage, and IR Sensors, detecting number of slots, on both sides of the robot. A software program to manipulate these inputs form sensors is based on FSM , serving different states of the robot and drivers to interface the external hardware as well as playing a rule of IoT, as a good IoT application requires only about 0-5% of the human attention, therefore the user interferes only by a bottom press in a mobile application to start parking, by receiving the number of slot specified by the user application to be parked in. A microcontroller is receiving the counts through a Bluetooth module and counting slots through an IR sensor. Once the counter is matched with the slot number that user sent, parking process starts operation. FSM is the algorithm gathering these different states and inputs to perform the task of slot counter and parking process.

Interpreting in more details with figures in the upcoming points, declaring the hardware used, functionalities and the software design of the robot.

4.2 Hardware in use

Finding a choice of the hardware with good performance and low cost is always an issue in any Engineering application and consume a counted time and effort, therefore spending the time to choose is not a waste in order not to redesign later based on better choices according to new circumstances.

- **DC motors**

  A couple of DC motors are acting as two-wheel drive with torque and speed that can handle the weight of the Robot with the other components and circuits.

- **Servo motor**

  A servo motor playing the driving wheel rule of a real car, rotating the robot with different degrees according to the sensors inputs and the software design.

- **Steering mechanism**

  Steering mechanism is being derived from servo motor, turning the robot right and left to perform the parking process accurately.

- **DC motor driver circuit**

  DC motor is a coil circuit needs a large Electromagnetic Force (EMF) and generates a large back EMF as well in case of shutting down the voltage source which puts other circuit components in danger of exposing to large amount of current flow that may zap the entire system, particularly when dealing with highly current sensitive components such as microcontrollers, so a good protection to the circuit is a need.

To isolate the power supply of DC motors, from the other circuit parts, a CYTRON circuit driver is used achieving previously mentioned reasons, CYTRON is dual driver serving the couple of the DC motors as well as isolatin them from the rest of the system.

- **Microcontroller**

  TivaC (TM4C123X) ARM architecture based is intelligence part, handling the system decisions and inputs getter from outside the system, outputs server from inside the system.

As ARM architecture is the most used in several applications such as mobile processors, Automated Cars, Smart Homes, ... etc. the ARM Cortex M4 microprocessor was the selection to get in touch with a standard used architecture, on the other hand, TM4C123X is a rich peripherals board introducing several useful devices and registers such as Timers, General Purpose I/Os GPIOs , Pulse Width Modulation (PWMs), Universal

Asynchronous Receivers Transmitters (UARTs), Interrupts, Phase Locked Loop (PLL), System Clock, ... etc. that make building a large system with different hardware interfaces a lot easier in implementation and faster and accurate in performance and execution.

- **Bluetooth module**

  The IoT part is a Bluetooth module that connects a mobile application, User Interface, with the automated part of the system which is Robot. A user sends a request of the desired empty slot to be parked in through the application and the module receives that request forwarding it to the microcontroller, microcontroller accordingly starts its operation.

- **IR and ultrasonic sensors**

  Sensors are extremely important devices in any embedded systems application, as they connect the external world to an artificial part which accordingly takes actions based on the coming inputs. Here, one IR sensor is attached to the robot side sensing slots and increments a software counter on each one passing by, once they matched the number received from the user mobile application through a Bluetooth module, the robot starts parking in the specified slot. ultrasonic sensors tracking the distances surrounding the robot body sending feedback to the controller about any obstacles in front of the robot and distances readings in the specified slot as inputs to the parking algorithm.

In the upcoming section, more detailed points about the software written interfacing those variance devices and hardware pieces.

4.3 Hardware interfacing

In this section you will find the software written to interface the previously mentioned hardware devices referring to the important peripherals and accessed registers to manipulate the varying inputs.

4.3.1 DC motors PWM interface

DC motors are electrical mechanical devices meaning that taking electrical applied voltage as an input and proportionally generating mechanical output represented in Torque τ and speed restricted by the equations:

$$\textit{Shaft torque}\ \tau = 60\ \frac{\textit{Output in watt}}{2\pi N}\ ,\ N \textit{ in rpm.} \qquad (4.1)$$

$$N = \frac{Eb}{\varnothing}\quad Eb \textit{ is emf and } \varnothing \textit{ flux.} \qquad (4.2)$$

To control speed of a device such as DC motor, we need PWM to deliver a specified period for the high cycle and the low cycle. See figure (4.1) and refer to equation (4.3) Ratio between the high cycle to the total period is called duty cycle that controls motor speed.

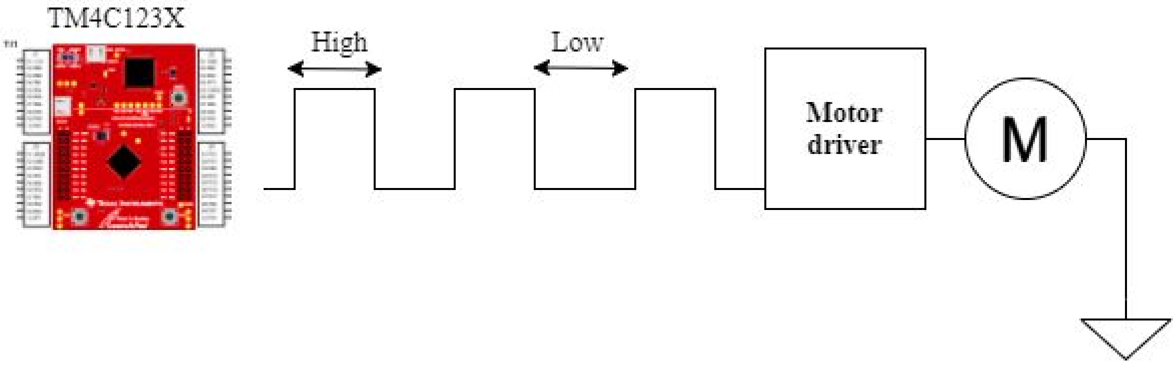


Fig. (4.1) PWM feeding a DC motor

$$\textit{Duty cycle} = \frac{high}{high{+}low} \qquad (4.3)$$

Implementation of PWM in hardware does not require an interrupt, only using PWM registers in TM4C123X in count up/down mode, see figure (4.2).

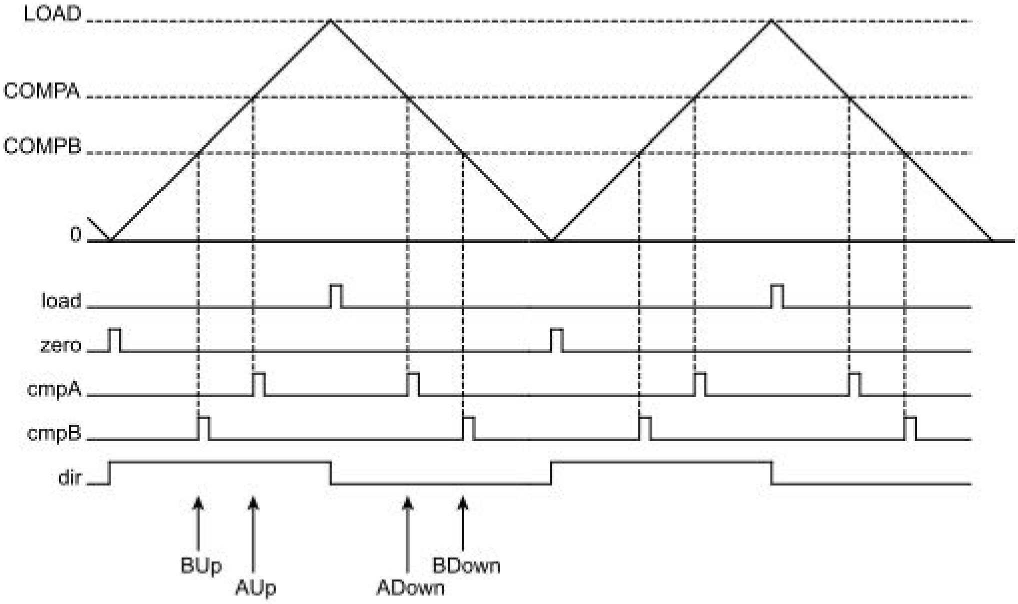


Fig. (4.2) Count up/down mode

To operate any module in TM4C123X , first initialize its registers giving them the proper value for the application and set the register clock properly, here using the PWM module needs the system cock to be set, as it is a timer triggered internally. The first line in the initialization activates module number 0, each module has two generators A and B, this activation requires about a clock cycle to be done so this clock is wasted by a dummy volatile variable called delay. Each module generates its output to specific GPIO port, here PORTB is used as an output so the clock of this port needs to be activated in the same manner with the delay issue. The next five lines of code initiates PORTB with the mentioned pins to be in the alternative functions mode, as each pin in each port may has many functions besides the GPIO, as well as disabling the analog functionality of the port and putting the specified pins in PORTB in digital output mode. After finishing the GPIO initializations, begin with putting the desired values in the proper register such as the total period of the PWM in the LOAD register, the duty cycle in the COMP register and specifying the mode of operation whether count down or count up/down mode in the GEN register with writing 0 value in the CONTROL register before writing any value in the other registers then writing back 1 value after finishing, finally we enable the mentioned module. See Appendix C for initializations of motor direction control. Previous initializations setting up the PWM port clock and duty cycle for both motors to start driving and initialization for the direction of the DC motors either forward or backward drive to gain more flexibility in robot motion.

4.3.2 Servo-Steering mechanism interface

Servo motors are simply consisting of a DC motor with a control circuit to derive it in an angular motion that is controlled by a periodic PWM. Period of the PWM is a servo manufacturer based value, mostly 20 milliseconds, and the high portion duration of the total period is responsible for moving the motor with specific angle by a countdown PWM, see figure (4.3).

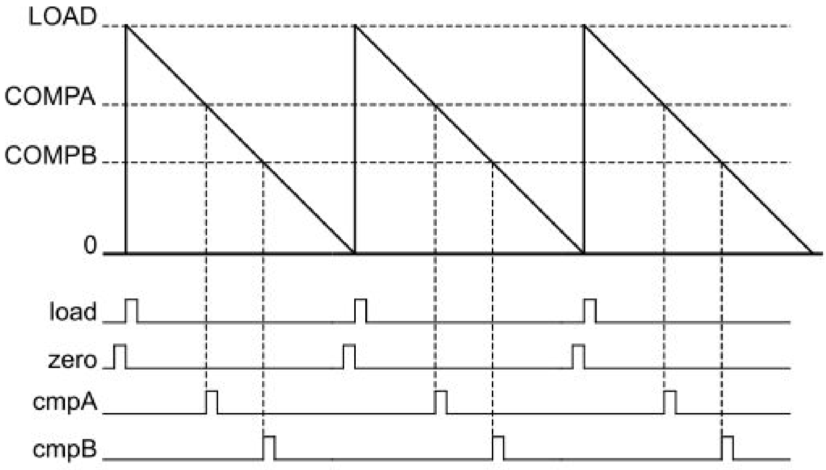


Fig. (4.3) Countdown mode

In case of MG996R servo, 1 millisecond means zero degree and 2 milliseconds means 120 degrees. A code interfacing the servo using another PWM module in Launchpad in the same manner as DC motor. Delivering the appropriate angle to servo is handled in a c function taking the angle as an input and manipulates this input to load the COMPA register with a value that represents the pulse period, refer to Appendix C for initializations and codes.

With both servo and DC motors we obtain a full motion control over the robot body and serve the different cases for the robot most likely to be in.

4.3.3 Ultrasonic sensor interface

The principle of operation of ultrasonic sensors is to emit short, high frequency sound pulses at regular intervals. These propagate in the air at the velocity of sound. If they strike an object, then they are reflected as echo signals to the sensor, which itself computes the distance to the target based on the time-span between emitting the signal and receiving the echo. See figure (4.4).

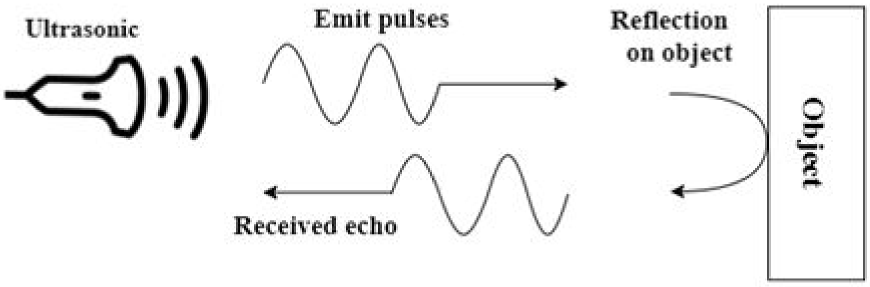


Fig. (4.4) Ultrasonic pulse emission and echo

In a case such as parking, the object to be detected is the space between the garage wall and the robot simulating the car motion to identify whether a slot is available or busy by a vehicle; once the ultrasound waves hit the wall, a feedback of the distance is being measured and serves its role in parking process, see Figure (4.5).

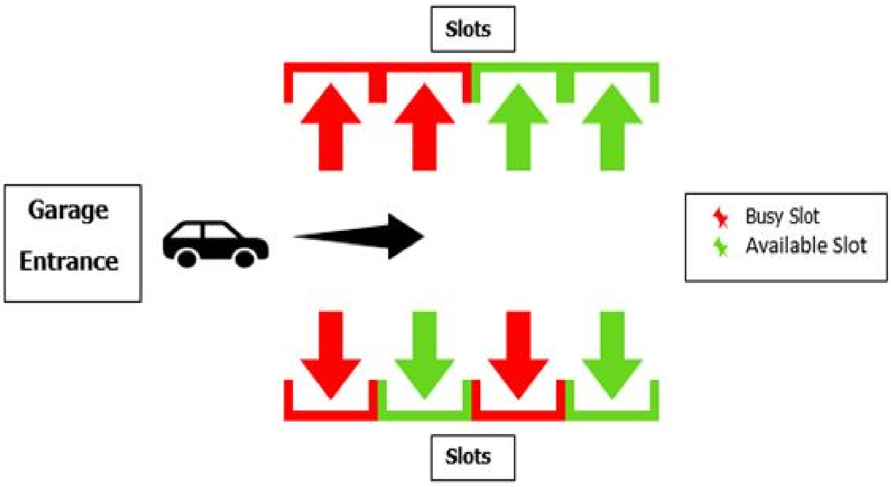


Fig. (4.5)Garage slot simulation

Pulses serving the sensors are being generated using timer A in TM4C123X microcontroller. For timer activation in microcontroller, firstly we need to activate the alternative functions of the desired pin of the chosen port, as previously mentioned that each port has multiple peripherals distributed on its pins, then loading the control register with a value corresponding to timer peripheral usage, disabling analog function of the pin, setting pin direction whether input or output and lastly enabling the digital functionality of the used pins. These steps are done to four ultrasonic sensors to cover the working area efficiently. Initialization of timer peripheral can be found in Appendix C. A sequence of steps to be done accurately to get distance, firstly turn off the counting register, send a logic zero pulse to sensor trigger pin then wait about 20 microseconds followed by a logic one with the amount of delay, a logic zero pulse again all to the same sensor pin, wait until status of register indicates new input, read the input from the counter register, convert the reading of the received echo pulses into useful information which is the distance and return that distance to be used later. C definition serving that purpose with the conversion equation provided by ultrasonic manufacturer's datasheet can be found in Appendix C.

4.3.4 IR sensor interface

IR sensor role is to count the number of slots as each slot is attached by a dark piece to be detected by sensor. Each time it senses a slot, a software counter is incremented until it matches the number sent from the user mobile application, once the matching happens the parking process starts its operation. Figure (4.6) interprets the process graphically.

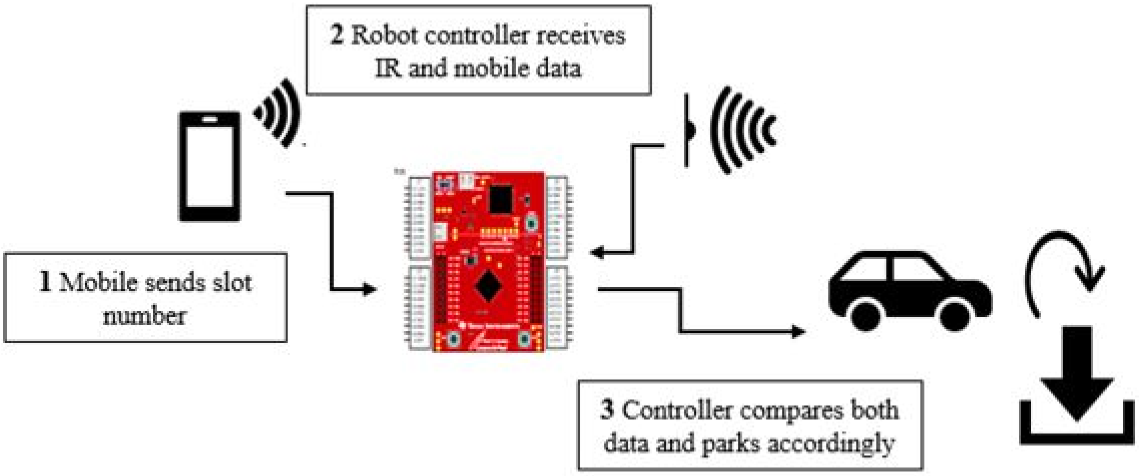

2 Robot controller receives IR and mobile data
1 Mobile sends slot number
3 Controller compares both data and parks accordingly

Fig. (4.6) Graphical illustration of app-IR related process

Interfacing IR sensor is considered easier than previous hardware as it is only needed to use one of the available GPIOs, using one pin of PORTE as an input from the IR sensor counting slots in a garage, this pin to be initialized as input pin, disabling the alternative functions, disabling the analog and activating the digital functionalities of that pin. A routine that takes the input of the IR input and manipulates it with a flag to be set in case of a new reading then incrementing the counter based on it and clears the flag if it is already set with Null reading. Initializations and a c definition returning the counts in Appendix E.

4.3.5 Bluetooth module interface

Bluetooth is a wireless medium and a data protocol that connect devices together over a short distance shorter than 10 meters. If a computer or a phone in the network provides a bridge to the internet, the Bluetooth-Connected device becomes part of the IoT . Bluetooth is classified as a personal area network (PAN) because it implements communication within the range of an individual person. Alternatively, devices within a Bluetooth network are usually owned or controlled by one person, when two devices on the network are connected, we often say that the devices are paired. From another perspective, Bluetooth network is considered as a client-server paradigm, where the Embedded System is the server and the Bluetooth-connected device is the client that requests information from the server or sends data to the server .See figure (4.7).

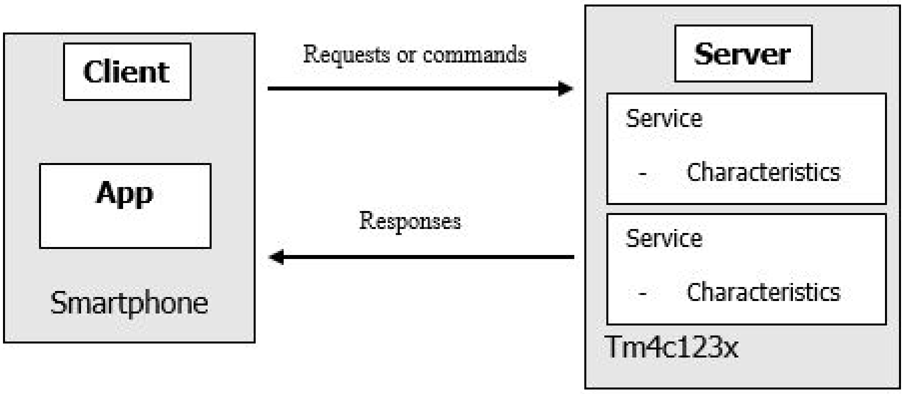


Fig. (4.7) Server-Client illustration

In this demonstration, the user mobile application is the client that sends data about the selected garage slot from the user to the microcontroller which is the server and requests to initiate the parking process. Graphically demonstration in figure (4.8).

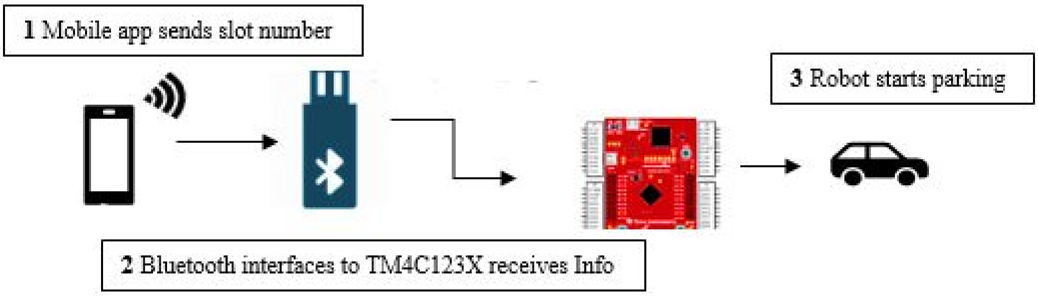

1 Mobile app sends slot number
3 Robot starts parking
2 Bluetooth interfaces to TM4C123X receives Info

Fig. (4.8) Graphical demo of app-Bluetooth interaction

Bluetooth module is a serial data transmitter (TX) and receiver (RX) works as master-slave device that makes the UART serial protocol is the best choice to be interfaced with, see figure (4.9) Bluetooth interface.

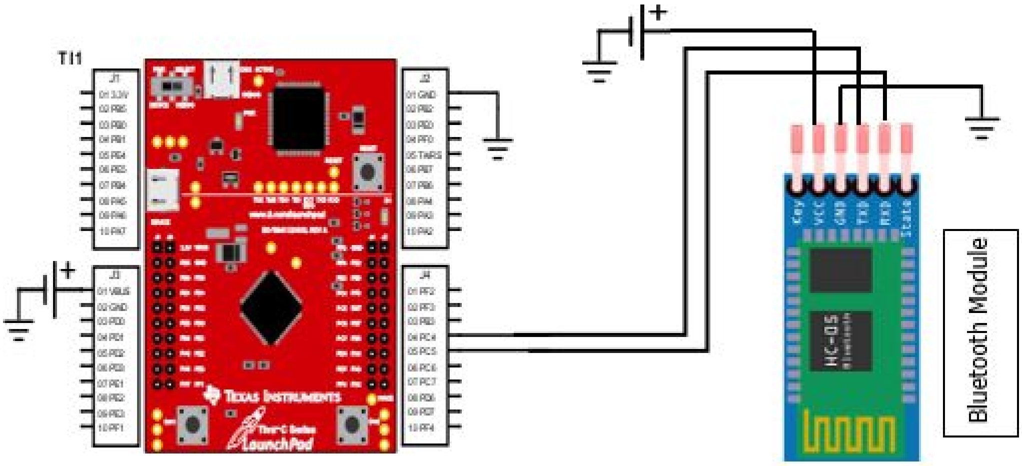


Fig. (4.9) Bluetooth interfacing

A UART1 interrupt triggers the reception in the TM4C123X, receiving data from the user mobile application, these data to be buffered in a UART1 hardware buffer and a flag register (FR) is set indicating a full buffer, see figure (4.10).

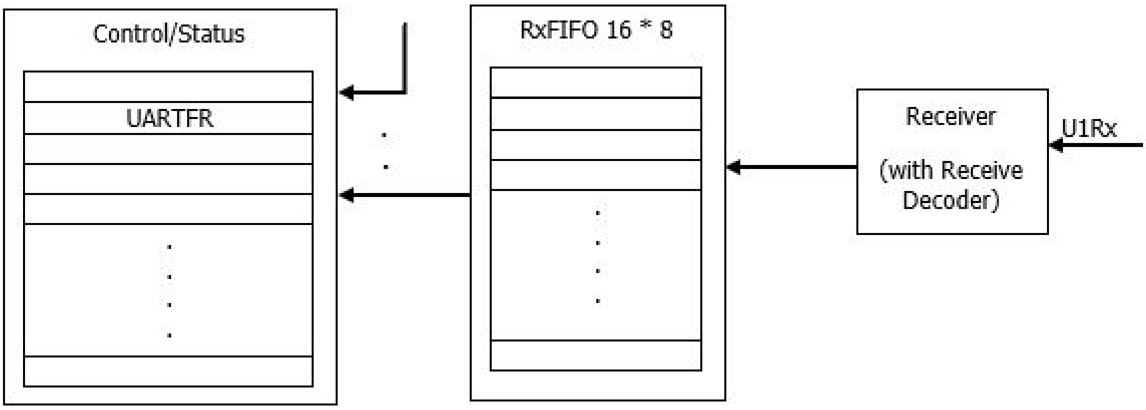


Fig. (4.10) UART1 receiver buffer and registers

Software routine called Interrupt Service Routine (ISR) or UART interrupt handler, in case of using a TM4C123X microcontroller, extracts these buffered data, selected slot number by user, and passes it to another software routine to compare it with the returned slot number from the IR routine, revise section 4.3.4, then resets the reception flag back again. Another role that the interrupt serves is triggering the software-implemented algorithm that drives the robot from the garage entrance until it settles in selected slot. UART protocol is to be selected by enabling the alternative functions of PORTC, pin 4 (PC4) as reception (RX) and pin 5 (PC5) as transmission (TX), disabling and enabling the analog function and the digital function respectively, setting the baud rate of data transfer by specifying the integer part and load it to Integer Baud Rate register (IBRD) and the floating part loading it to the Float Baud Rate register (FBRD) according to these two equation:

$$IBRD = int\left(\frac{System\ Clock\ in\ Hz}{16 * Baud\ Rate} \right) \tag{4.4}$$

$$FBRD = round(IBRD\, float\, part * 64) \tag{4.5}$$

Then enable First In First Out (FIFO) and the word length in register LCRH, masking the interrupt using Interrupt Mask register (IM), Enable UART1 interrupt from the Nested Vector Interrupt Controller (NVIC) by setting its priority and its enable registers with the appropriate interrupt number mentioned in the manufacturer’s datasheet and finally enable the interrupts. In the interrupt handler, we check if the receiving flag or receive timeout flag is set, then acknowledging the reception by clearing the proper bits, RTIC or RXIC, in the Interrupt Clear register (ICR) and deliver the FIFO contents to a software variable. See Appendix C for codes and initializations.

4.4 System implementation

This section demonstrates a software-implemented algorithm gathering different inputs that delivered from hardware interfaced, discussed in previous sections, and manipulates them in a manner that serves the purpose of parking simulation. The UART1 interrupt flag register triggers an FSM that passes through specific states sequentially, once the states are served successfully, the robot goes in sleep mode to save power. Figure (4.11) shows the derived software system in graphical flowchart.

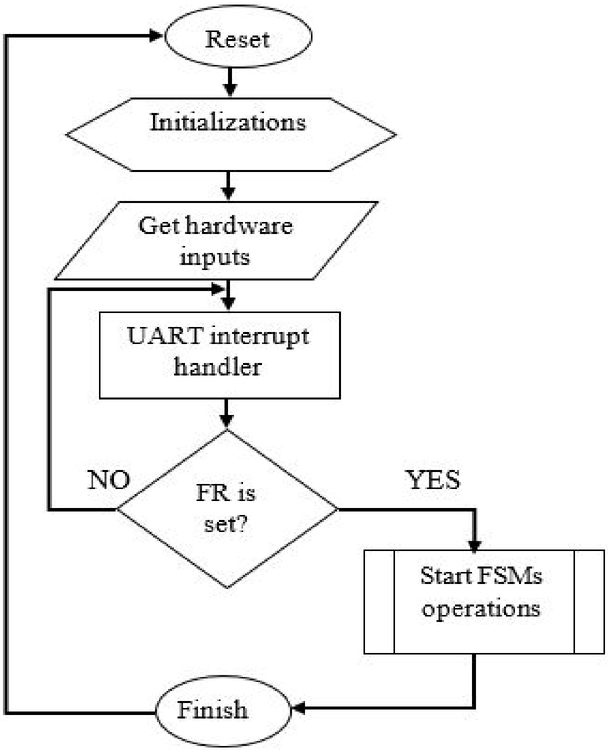


Fig. (4.11) System flowchart

From the hardware perspective, connecting components and other different modules alongside microcontroller is quite challenging in both reality and schematic drawings, which concerns us to view as most illustrative and complete schematic as possible to the reader to follow the discussed interfaces with microcontroller PORTS, we have connected only one ultrasonic sensor – four sensors to be used – since it is enough as an illustration, besides removing the power supply circuit and servo motor interface, otherwise it would not fit in a page, see figure (4.12) for simplified system schematic.

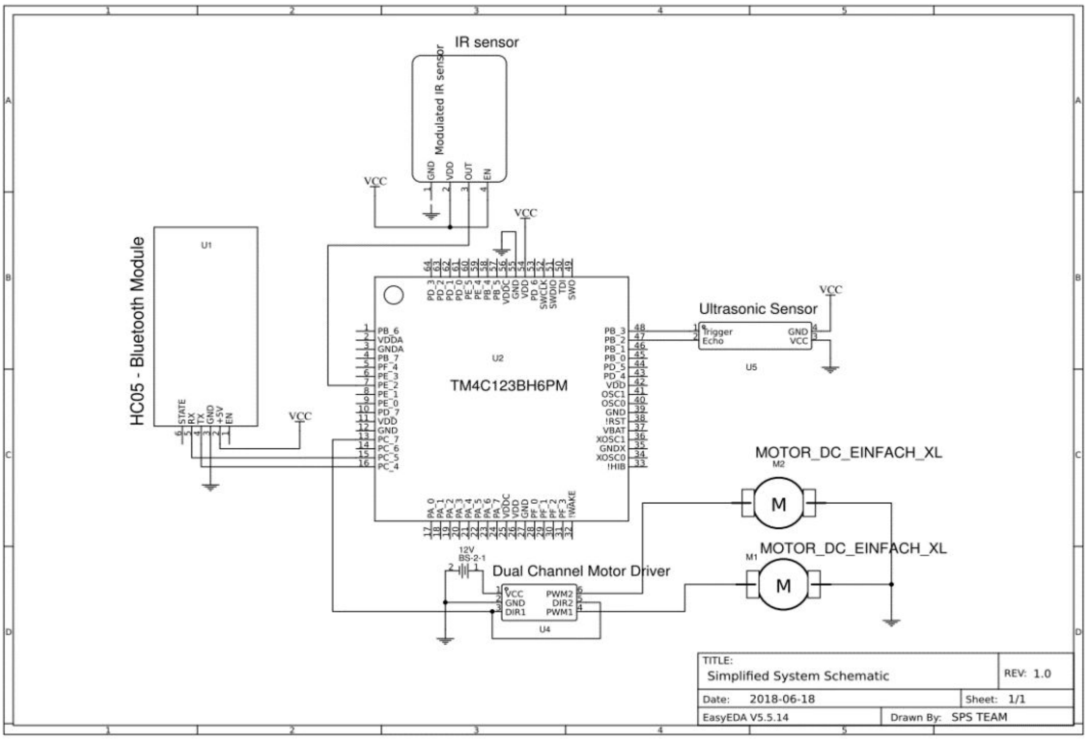


Fig. (4.12) Simplified system schematic

4.4.1 Available parking choices

Lastly the real parking interaction with algorithms and software, there are mainly two popular methods either in autonomous cars or in human-controlled cars:

- Parallel parking.
- Perpendicular parking.

Parallel parking is often the most difficult part of ordinary drivers, and one of the most feared tasks for some. Big cities specifically require great amount of parking skills as parking spaces are often limited. Removing the difficulty, stress and uncertainty of this task is very appealing. Imagine finding the parking spot of your choice and by simply pressing a button your car autonomously parks itself, hence we have built a low-cost prototype which would enable a car to find a suitable parking space and park itself without any assistance from the driver. They employ cheap ultrasonic sensors to sense obstacles and calculate the parking area, and this is what we have chosen for our project to be implemented.

Perpendicular parking is the most efficient and economical since it accommodates the most vehicles per linear meter, and is especially effective in long term parking areas. One of the difficulties in achieving automatic parking is the narrow operating place for collision free motion of the vehicle during the parking maneuver, and planning of optimal trajectories is often used in the applications, where an optimal stopping algorithm was designed for parking using an approach combining an occupancy grid with planning optimal trajectories for collision avoidance. Implementation of any of the two methods requires a software application for FSM that organizes and achieves the most likely states of the car to be in, which is being interpreted in the following sections.

4.4.2 Software and state machine

System structure based on two FSMs, one for parking slot detection and the other one for accurately performing the parking algorithm. Both of FSMs work at different times, that is, the parking algorithm is executed only when a parking space has been detected, after which the first FSM is no longer scheduled to be executed. In the written software view, system is consisting of two main software blocks, first block serves the interfaced hardware sensors and modules, saving its inputs in memory variables to be used as state machine inputs. The other software block consists of the two FSMs. Modular programming is a software Engineering concept that we used in breaking the whole software into independent and separated tasks with its own routines/functions. Declaration of each routine in a module is saved in a c standard header file, while the definition of that routine is saved in a well-defined name c file that represents the module tasks, that makes editing, understanding and redefining any module is much easier than writing software in one file, see figure (4.13) for more illustration.

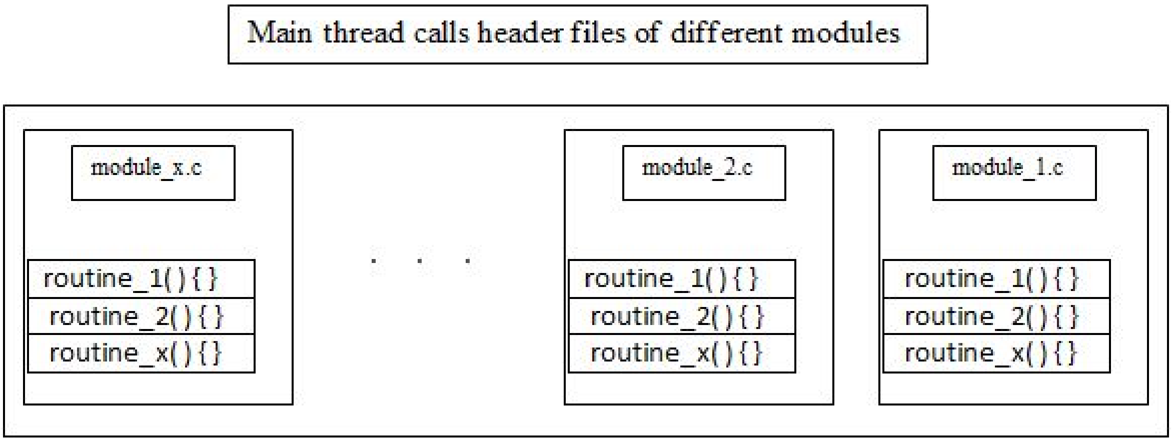


Fig. (4.13) Modular programming demo

Main system thread contains the parking slot detection and parallel parking algorithms. Initializing the movement flags of the car, it sets the car on a default trajectory and then calls detect parking slot state machine. Once a parking slot has been detected, the parallel parking algorithm is called. After the car has successfully settled, it goes idle until it is reset.

4.4.2.1 Parking slot detection FSM

Parking slot detection state machine starts by moving car forward with the IR sensor counts the slot numbers passing by until it matches the one that user has sent by the mobile application through Bluetooth module then calls the parallel parking state machine. Cases of slot detection state machine are:

I. START:

This is the state which the car is immediately in when it is turned on. The system will remain in this state for as long as there is not a match yet between IR counter and user selected number, the car will keep going forward at a relatively constant distance from the boundaries of the parking spot, once a matching happens, the system enters in a new state called DETECT. This indicates that the car has reached the start of a possible parking space.

II. DETECT:

In this state, a slot has been detected already. A backside ultrasonic sensor
returning the distance reading to be compared with a predetermined

boundary that indicates an end of the selected slot to park in then enters a new state called READY.

III. READY:

In this mode the car moves forward and positions itself to correctly reverse back into the parking slot. Once the READY state is entered, the algorithm raises a flag that yields control of the car to the Parking Algorithm FSM. Simultaneously, the parking slot detection algorithm is executed and no longer scheduled.

Figure (4.14) shows the parking slot detection FSM. Once the parking slot detection yield by a done flag, the Parallel Parking FSM acts and starts executing.

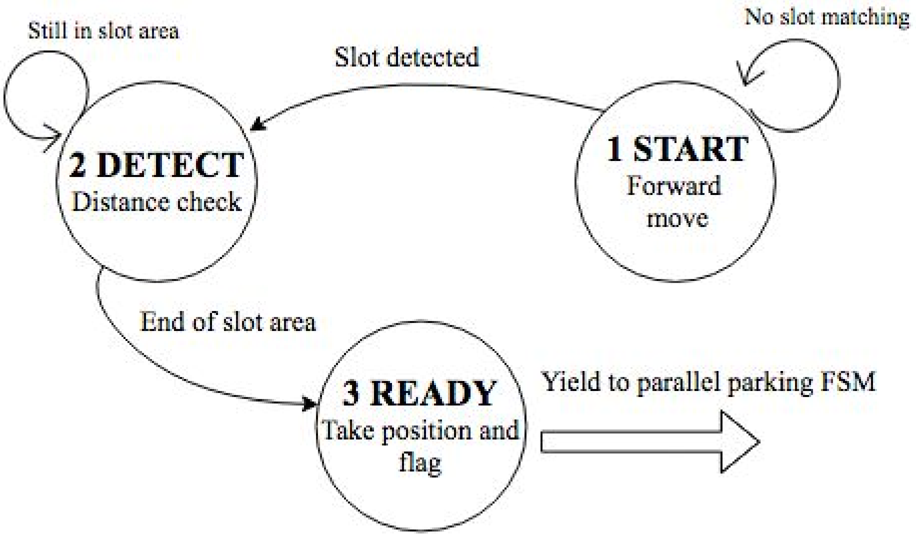


Fig. (4.14) Parking slot detection FSM

4.4.2.2 Perpendicular parking FSM

Cases that the car is most likely to go through in parallel parking state machine are:

I. RIGHT-BACKWARD:

It is the initiative state after parking slot detection FSM has accomplished its actions properly. The car turns the front wheels to the right and reverses into the parking space. RIGHT-BACKWARD state will continue to be executed for as long as the car is within the limits set by the corresponding condition defined by Flags. The purpose of RIGHT-BACKWARD state is to force the car-simulated robot to separate from the boundary into the parking slot. The state will be executed until a reading from the back-sensor or the left side sensor indicates a matching with a predefined distance and a flag is raised. If it is the back-sensor flag, then it means the robot has settled in the parking slot, then goes through SETTLE the last state. Case it is the left side sensor flag, then goes through LEFT-FORWARD state.

II. LEFT-FORWARD:

Means that the robot faces the left wall of the parking slot and it needs a sharper entrance angle to be done in two states, one to drive the robot out of the parking slot in left direction LEFT-FORWARD, then goes through RIGHT-BACKWARD state again or STRAIGHT-BACKWARD according to the situation and tests.

III. STRAIGHT-BACKWARD:

A state to act if the two side sensors returning an equal distance reading after the robot goes through the LEFT-FORWARD state and it means that the robot needs only to go back and settles by SETTLE state.

IV. SETTLE:

Positioning the robot steering mechanism in forward, make any position corrections according to the difference between the two sides sensors reading and starts sleep mode.

Previous states are meant to deal with the case of right available slot, while if the case is a left one, we need only to reverse the directions of right to left or left to right if it happens in any of the states. See figure (4.15) for parallel parking FSM. Implementing and connecting both FSMs is quite challenging and requires a lot of testing and may result in some modifications in some states or all states, as well as modifications in the software-implemented that may reflect on necessary editions on states sequence of operation, that is, according to the test results with hardware and the garage-simulated fabric, revise chapter 2 and chapter 3 to picture things as one package.

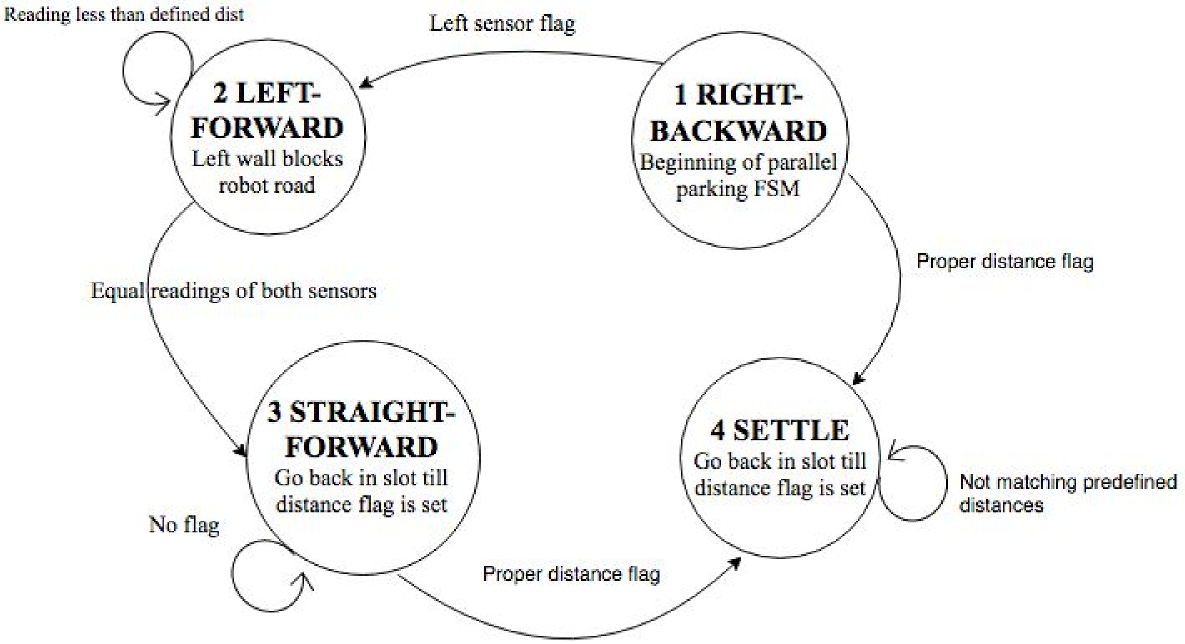

Reading less than defined dist
Left sensor flag
2 LEFT-FORWARD
Left wall blocks robot road
1 RIGHT-BACKWARD
Beginning of parallel parking FSM
Proper distance flag
Equal readings of both sensors
3 STRAIGHT-FORWARD
Go back in slot till distance flag is set
4 SETTLE
Go back in slot till distance flag is set
Not matching predefined distances
No flag
Proper distance flag

Fig. (4.15) Perpendicular parking FSM

CH.5

CHALLENGES

5.1 Garage controller using Raspberry Pi

1. Starting to learn about Raspberry Pi and its interfacing with different kinds of devices using a friendly reference as (Simply Raspberry Pi) which has fundamental information about Raspberry Pi and helped a lot in walking the first steps in dealing with the Raspberry Pi.

2. Searching for the convenient sensors needed for the project to be interfaced with the Raspberry Pi. This stage was a slightly difficult due to the complexity of the wire connections and the code needed amount of time to be build completely and to be tested in order to make sure that it works in different situations.

3. Since most of the available server platforms with dedicated databases are costly,we found a cloud server which provides the cloud services with the databases freely for the subscribers but with limited features.It is called (000Webhost).

4. In the scope of exploiting the platform provided services ,which receive the data from the garage, and using its database where the data is stored, A complete understanding of main web development languages as SQL, PHP, HTML and CSS was required.Doing this task was with the help of different kinds of resources such as videos,articles or some language learning websites like (SoloLearn) website.

5. After doing the hardware tasks as interfacing the different sensors with Raspberry Pi and finishing the software programming parts needed for the Raspberry code and the server code,the testing stage began in in order to check the working of the whole garage system including the Raspberry Pi with all of its hardware connections and the Server with its database. In this stage this two parts were perfectly tested to make sure of their cooperation to do the required tasks and to detect their behaviour in different situations.

6. After finishing the IoT garage tasks, it was necessary to check the interoperability between the server and the client application to see if the server is correctly accessed by the application to fetch the required data of the empty slots availability and if it can cope with the continuous changes of the parking slots in the real time.

5.2 DC motor using TivaC microcontroller

In this project, It is required to control a DC Motor with TivaC microcontroller. The maximum current that can be sourced or sunk from a TivaC microcontroller is 12 mA at 5.5v. But a DC Motor need currents very much more than that and it needs voltages 6v, 12v, 24v etc, depending upon the type of motor used. Another problem is that the back emf produced by the motor may affect the proper functioning of the microcontroller. Due to these reasons, we can’t connect a DC Motor directly to a microcontroller.

To overcome these problems you may use an H-Bridge using transistors. Freewheeling diodes or Clamp diodes should be used to avoid problems due to back emf. Thus it requires transistors, diodes and resistors, which may make our circuit bulky and difficult to assembly.

Again to overcome this problem the CYTRON circuit driver IC is used. It is a Fully NMOS H-Bridge for better efficiency, where no heat sink is required and it solves the problem completely. You needn’t connect any transistors, resistors or diodes. We can easily control the switching of CYTRON circuit driver using a microcontroller. All inputs of these ICs are TTL compatible and clamp diodes are provided with all outputs. They are used with inductive loads such as relays solenoids, motors etc.

The challenge in dealing with DC motors is to deal with the isolation and the protection between the microcontroller, the motor circuit and the supply.
Also, It is required to settle a suitable speed for the robot to have efficient time for sensors to detect targets which need trial and error procedures on PWM loading value.

5.2.1 Motor control using PWM

Bi-directional motor control can be done using an H-bridge circuit with pulse-width modulation (PWM) from a microcontroller to vary the speed. Several design challenges include preventing shoot-through, implementing a snubber circuit, as well as open and closed loop (such as PID) control mechanisms.

1. PWM Control of an H-Bridge
   An H-bridge circuit consists of four transistors (usually two PMOS's and two NMOS's). To maximize efficiency, the transistors are driven at a higher voltage than the microcontroller. A typical H-bridge circuit with logic scaling circuitry is shown above.

   Each side of the H-bridge has two transistors with the gates tied together resulting in complementary operation Q3 is always off when Q1 is on and vice versa. The same is true for Q2 and Q4. The circuit works by setting PWMB to logic zero (Q2 on; Q4 off) and then setting PWMA to logic high (Q1 off; Q3 on). The motor direction can be reversed by toggling PWMA and PWMB.

PWM is a simple way to vary the voltage applied to the motor. Most microcontrollers have dedicated PWM hardware, but an output compare timer can also generate a PWM signal. PWM works by rapidly turning the motor on and off. For example, if the motor supply is 12V, the motor can be driven at 6V by applying a 50% duty cycle where half the time 12V is applied, and half the time 0V is applied as shown in the plot below.

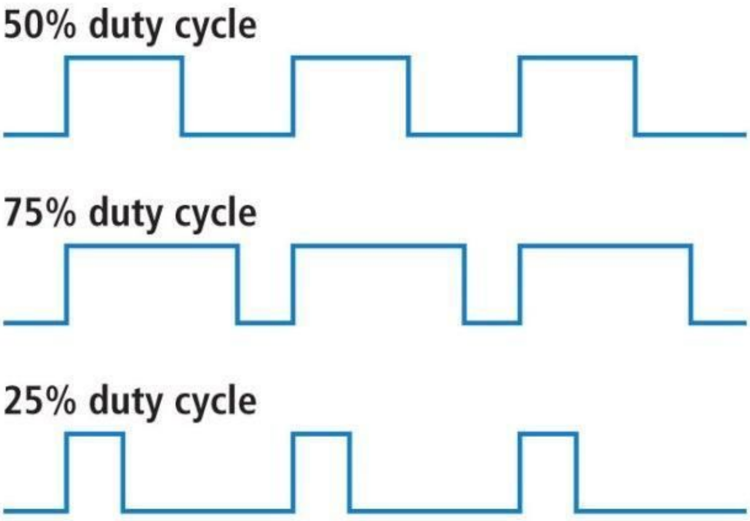


Fig. (5.1) Varying the duty cycle

While using PWM is simple, it introduces a problem called shoot-through which occurs when current flows directly from the power supply to ground when the transistors are being switched. For example, when the PWMA input signal switches from high to low, Q1 turns on and Q3 turns off. For a brief period of time, both Q1 and Q3 are partially on allowing current to flow from the supply to ground. This causes efficiency to plummet and introduces heating problems in the transistors. To overcome this problem additional circuitry must be added to ensure Q3 turns completely off before Q1 starts to turn on.

2. Preventing Shoot-through

   The easiest way to implement a shoot-through prevention circuit is to use an integrated circuit (IC) that has shoot-through protection built-in. The Si9986 IC from Vishay is an H-bridge motor driver circuit with built-in shoot-through protection as well as logic translation circuitry (Q5A and Q5B in the diagram above). It is a great solution for controlling a small DC motor using a PWM signal from a microcontroller. When using the Si9986, as with using any motor driver, a snubber circuit is required to reduce electromagnetic noise in the system.

3. A Snubber Circuit to Decrease Noise

   A snubber circuit is used to suppress the voltage transients caused by PWM switching (as well as by the inherent switching in brushed motors). A DC motor is an inductive load; the voltage across it is proportional to the change in current, given by:

$$V= L\, di/dt \tag{5.1}$$

When the PWM signal switches the motor from on to off, there is a rapid change in current (ie di/dt is large) which causes a voltage spike. Without a snubber circuit, the energy from the voltage spike can result in arcing, damage to the body diode in the H-bridge transistors, or cause electromagnetic interference in nearby circuitry. The snubber circuit safely dissipates the energy in passive elements. A simple, effective snubber circuit consists of a resistor and capacitor in series across the terminals of the motor.

CH.6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

A complete IoT system prototype was made to represent the working of different main IoT parts (sensors-gateway-server and database-IoT application) and to show the cooperation between each others.
In the scope of the IoT garage, a Raspberry Pi was implemented in it as its controller and the system gateway to which the sensors are connected to be the central data gathering point from them and to send this data to the server which do its work to receive this data,store it in the database and make some analytic operations before it is represented in a webpage for the garage coordinator in an appropriate shape.

Along the time when the team was doing its work to implement this prototype, some conclusions were noticed and needed to be mentioned such as:

1. In the beginning of the project, there was a hope to have enough time to make use of the Raspberry Pi camera.Using this camera and by learning some main concepts about computer vision and object detection, we could have implemented a software to this camera to make it smart enough to detect the car existence in different parking slots in the garage and its location.According to figuring out the specific location of the car in the parking area,the camera can detect in which parking slot the car has parked.
Therefore, the camera will trigger the Raspberry Pi about this new information so the Raspberry Pi will send it to the server telling it that there is car has already parked in a specific parking slot.
The team was about to begin working on this task but the time was very short before the project deadline.

2. For the server, there was a desire in the beginning to use a free and famous cloud platform which is (Thingspeak) but it was figured that it will not be sufficient for the server tasks needed to be performed as Thingspeak is used mainly to receive data from a gateway but it represents this data in a graphical interface which is continuously changed according to the data received from the gateway connected to some kind of sensors in the physical world.

But this was not the best choice for the project requirements as it was not supposed for the server to store and represent analog values with different ranges to be shown on a graph while it was supposed to show different strict data which is not carrying a numerical value by nature such as the parking slot state (empty/filled), which slot the car has parked in, the time spent by parking car in the slot and the parking cost.

3. IR sensor is used to detect the parking state if a car has already parked in the slot or not.According to this task it worked efficiently in the project but it was known that in the practical real life it may not measure the parking space state precisely in some special cases like if there is another object instead of a car was in the IR sensor range.In this case the sensor will figure that there is a car had parked in the slot while this did not happen.

An auto-parking car-simulated robot software implementation provides automated means to a private car owner who finds difficulties in the parking process, these difficulties may be the lack of experience in parking, time waste in settling the car in a proper space as well as the fuel consumption due to the unnecessary moves of the car according to the previously mentioned reasons, which specifically has an awful impact on car spends.

Connecting the car to other system parts, IoT garage and the mobile application, via a Bluetooth module to meet the best performance and make full use of IoT concepts by keeping the human interaction with the system operation as low as possible, up to 5% of human attention to the operation or system, this Bluetooth receives data represents selected available slot, which accordingly the car simulation system starts parking with a software implemented in ARM based microcontroller, TM4C123x, based on specific states, which are the most likely to be experienced with the robot through the parking process. These cases may be changed partially with tests results to achieve the best performance and meet the requirements.

Higher accuracy, performance and achievements can be obtained in case of replacing the ordinary sensors, which lacks the accuracy, with Cameras, hence the software mechanism will be changed accordingly referring to machine learning algorithms in the automation industry and computer vision technology that guarantees achieving the requirements trading of the cost that definitely will be much higher than the case of sensors usage and regular processors.

6.2 Future work

6.2.1 Improve the security

Our future work is focusing on some improvements in the auto-parking system including, improvement in the security, scalability and availability.

IoT can connect many objects together such as smartphones, sensors and nodes in the case of improving the security issue these nodes may be Closed Circuit Television (CCTV) cameras. The development in IoT domain may be included in many fields like health care, transportation and computer science so this is so important to increase the level of security in it.

6.2.2 Visual security system

This project can be implemented based on computer vision. The suitable approach to design an interactive technology between the SPS, car owner and the operator.
This section cover different aspects:

1- Presents a related work of different parking systems.
2- Design, implement and evaluate a smart parking system utilizing CCTV cameras.
3- In the field of IoT, the cameras representing some nodes on the server which could be connected together on one board which can be accessed by the operator.

6.2.3 Related work

To provide an overview of the different technologies used by others, the following is a summary of the systems we have reviewed.

Benson et al. proposed that an RF transceiver and microcontroller system could monitor the availability of car-parking spaces and send this information to drivers and car park administrators and it is so similar to what has been done in this project [24].

Funck et al. proposed a system using CCTV cameras fitted in car parks to automatically detect car parking spaces. However, all these methods have the problem that sometimes they are not accurate [25].

Panayappan et al. described a parking system using VANET, which located available parking spaces. The system depended on roadside units to relay parking messages and Global Positioning System (GPS) coordinates to locate parking positions [26].

6.2.4 Hardware and software requirements

- Micro-Controller as in our case, TivaC.
- CCTV cameras.
- Parking server computer (PS) as in our case, Raspberry Pi 3.
- Suitable software like Keil.

Fig. (6.1) CCTV camera

6.2.5 Design goals

Firstly, the system must provide real-time navigation support to the driver like Google Maps.

Secondly, it has to provide friendly parking information service to both car owner and the operator.The system flow chart is shown in figure (6.2).

6.2.6 Proposed system

There are three main considerations:

1- An application installed on the user smartphone connected to the parking server computer.
2- IoT cameras which are connected to the micro-controllers as one network.
3- Databases for the parking system.

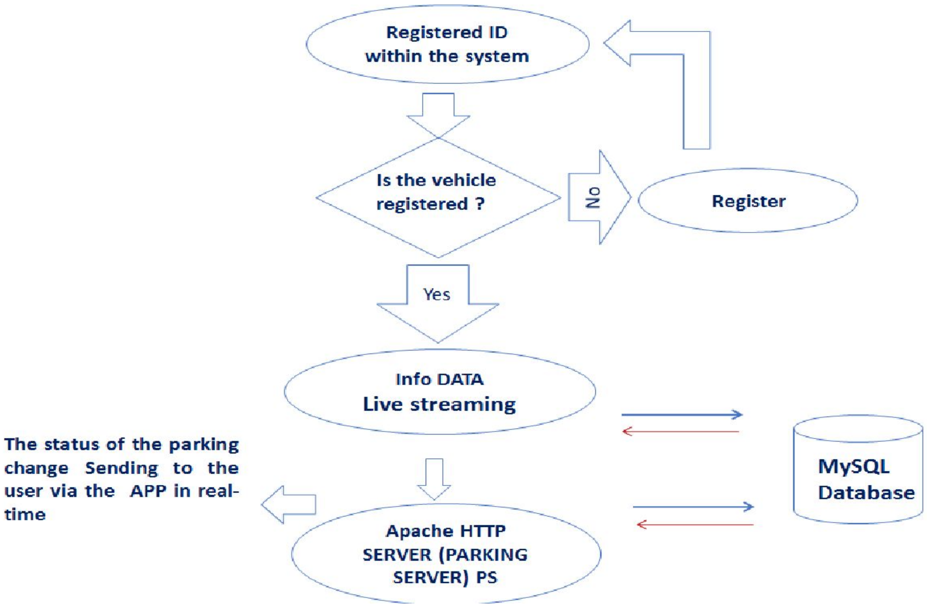

Registered ID within the system
Is the vehicle registered ?
No
Register
Yes
Info DATA Live streaming
The status of the parking change Sending to the user via the APP in real-time
Apache HTTP SERVER (PARKING SERVER) PS
MySQL Database

Fig. (6.2) The operations triggered by the car-in and car-out events.

6.3 Back-up database

Another database can be implemented on the network. Database mirroring involves two copies of the same data base built in two different hosts, one of them will be available to the user to use it and made changes on it and the other will be changed by any action applied on the original data base which called as the principal database. Mirroring involves applying the transaction log from every insertion, update, or deletion made on the principal database onto the mirror database. As shown in figure (6.3).

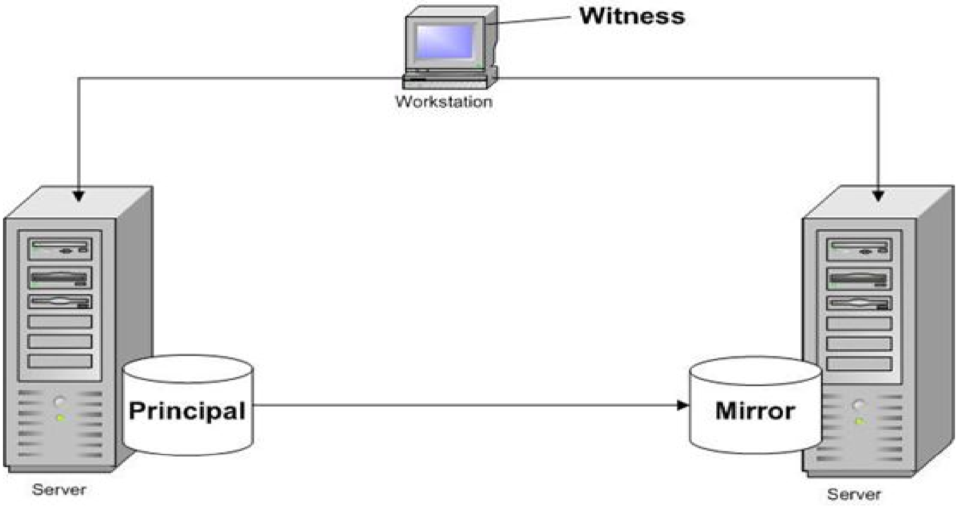


Fig. (6.3) Database mirroring

6.4 Improve the availability

Using satellite navigation device instead of wireless device to insure that drivers will always aware of whether there are free spots or not is a sufficient way to improve the availability.

6.4.1 Satellite navigation system

Satellite navigation is a new technology that can tell you where you are to the nearest few meters or better, it is technique that depends on two basic elements which are the transmitter and the receiver, the transmitter almost be navigational satellites and the most common used one in our days called GPS, The receiver technology is small enough to be incorporated into the electronics of a car or mobile phone and even in our cars.

To determine your location accurately, the receiver need to receive four different signals from four different satellites at least and by measuring the time taken by the signal to travel from the satellite to the receiver antenna. Some receivers have channels that can receive signals from up to 15 satellites.

6.4.2 Navigational satellites

The navigational satellites are positioned in highly stable Medium Earth Orbits (MEOs) at an altitude of about 22,000 Kilometers. MEOs are the orbits of choice for a number of reasons: their stability enables exact orbit predictions; the satellites travel relatively slowly and so can be observed over several hours, and the satellites can be arranged in a constellation so that at least four are visible from any point on the Earth’s surface at any time. As shown in figure (6.4).

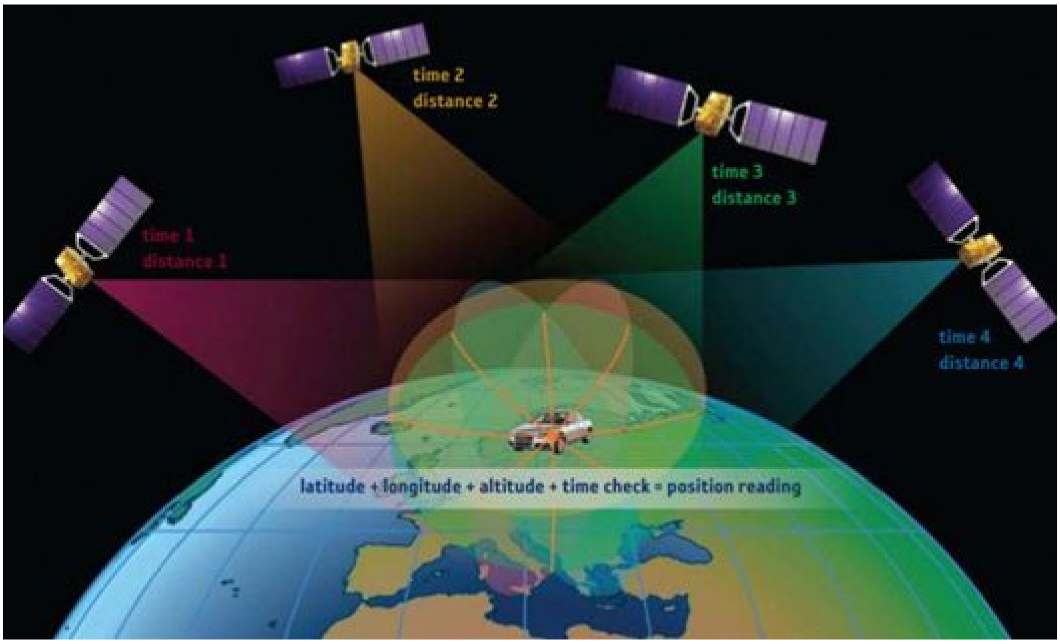


Fig. (6.4) Navigational satellites

6.5 Improve the scalability [29]

The scalability of this project can be improved by many ways like:

1- Making the algorithms effective to cover more parking spots.

2- Gathering many separate parking facilities in one cloud.

6.5.1 System overview

This system is derived from the idea of IoT, the system consists of RFID technology to monitor car parks and authenticate the user identity. The RFID is used because it could be implemented in large-scale systems and its low cost, and there is a Cloud-Based Server which is a Web entity that stores information provided by local units located at each car park, there is also a controller which could be Arduino or an Arm-Based Micro-Controller like TivaC and eventually a software application which running on Android operating system, the users will install it on their smartphones. As in figure (6.5).

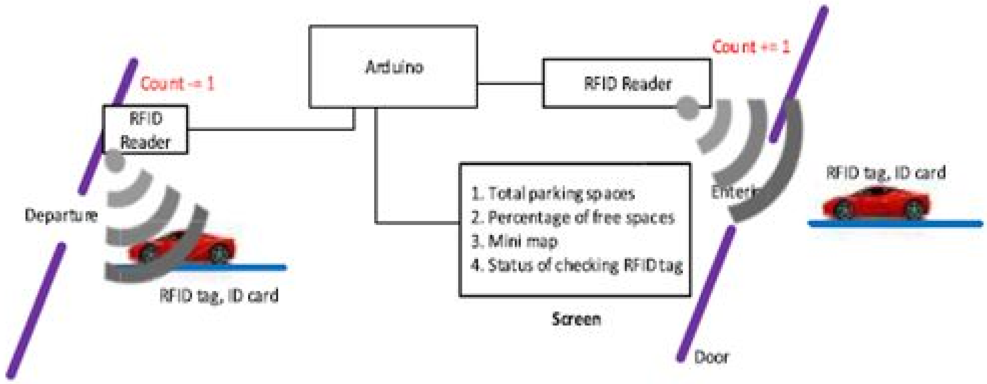


Fig. (6.5) System architecture

6.5.2 Network overview

1) Parking network

A network will be built as a backbone/infrastructure. A router will be in each car park as a node which has an access on the Internet by gateway functionality. This approach, also referred to as infrastructure meshing, provides the backbone for conventional clients.

It will assumed that each car park is a node.

The deployment network in a real environment is shown in figure (5.6) where each car park is labeled:

- P1 is car park number 1, N1 is the total parking spaces in P1.
- P2 is car park number 2, N2 is the total parking spaces in P2.
- Pn is car park number n, Nn is the total parking spaces in Pn.

The total capacity of the system is N = N1 + N2 + N3 + ------- Nn (spaces).

D is the real distance between two nodes in the network. Dij is the distance between nodes Pi and Pj. Figure (6.6) shows the network.

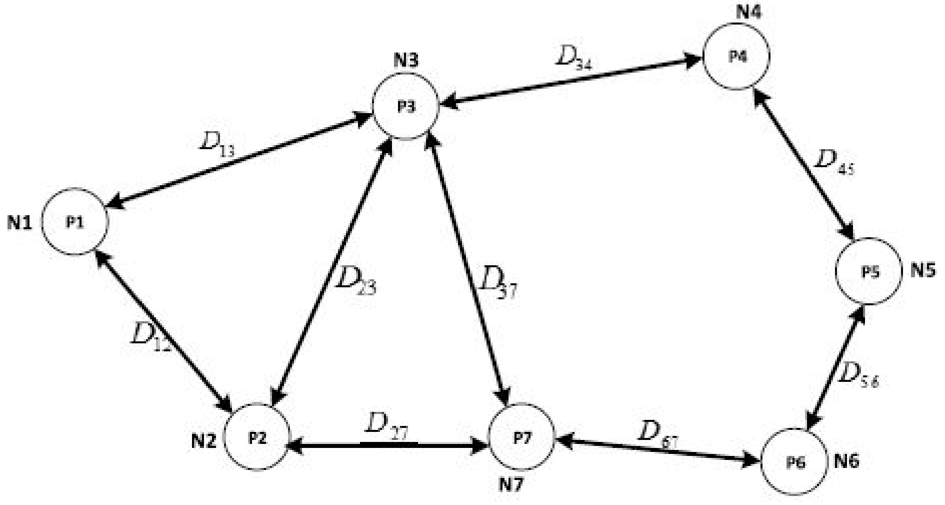
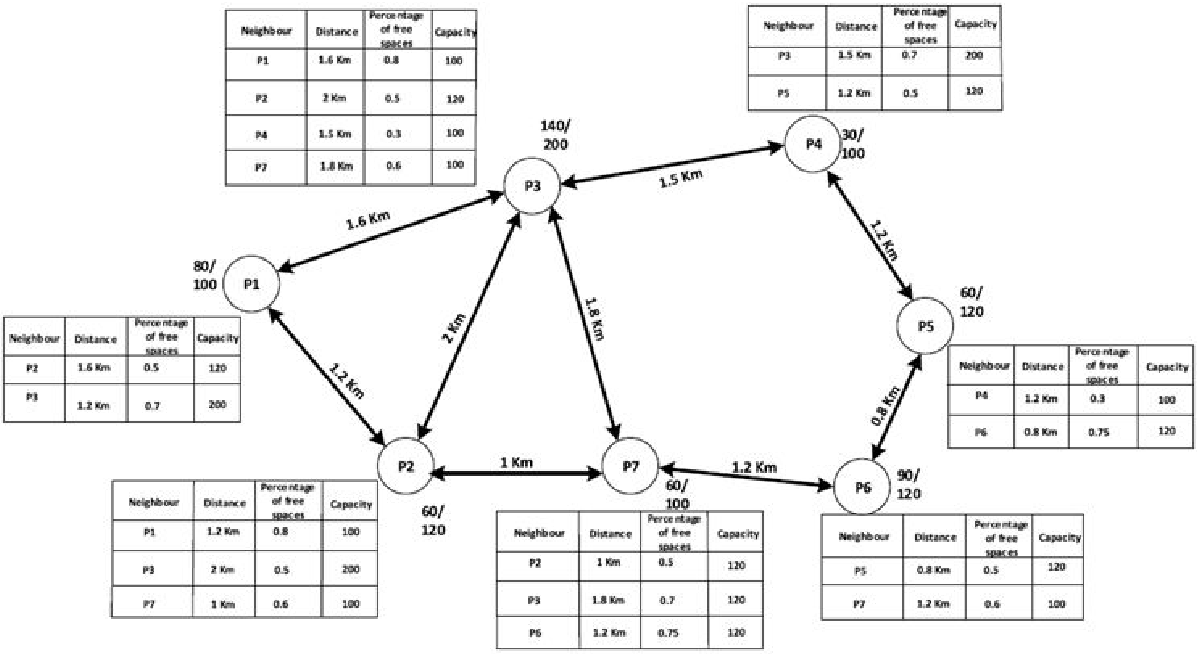


Fig. (6.6) The network architecture

Each node has a neighbor table to maintain information on the current status of the network. The neighbor table for each node contains information on the neighboring nodes directly linked to it. On the other hand, the queue is used to control the number of vehicles forwarded to the node, which aims to prevent overloading in the number of vehicles beyond the capacity of the node. In the proposed system, each node will broadcast a message to its neighboring nodes after a new node joins or leaves it. This message includes information on its total free resources. The neighboring node that receives this message will update its neighbor tables. In this network,
N1 = 100 spaces, N2 = 120 spaces, N3 = 200 spaces, N4 = 100 spaces,
N5 = 120 spaces, N6 = 120 spaces, N7 = 100 spaces; D12 = 1.2 km,
D13 = 1.6 km, D23 = 2.0 km, D27 = 1 km, D34 = 1.5 km, D37 = 1.8 km,
D45 = 1.2 km, D56 = 0.8 km and D67 = 1.2 km.
These parameters are shown in figure (6.7) using simple neighbor tables.
In figure (6.7), the total free spaces in N1 = 20, in N2 = 60, in N3 = 60, in N4 = 70, in N5 = 60, N6 = 30 and in N7 = 60. To increase the performance of finding a free parking resource, the neighbor table in each node contains information on the current number of free parking resources in the neighboring nodes. The idea is to use the number of total free parking resources in each node to calculate the cost for choosing a car park.



Neighbour	Distance	Percentage of free spaces	Capacity
P1	1.6 Km	0.8	100
P2	2 Km	0.5	120
P4	1.5 Km	0.3	100
P7	1.8 Km	0.6	100

Neighbour	Distance	Percentage of free spaces	Capacity
P3	1.5 Km	0.7	200
P5	1.2 Km	0.5	120

Neighbour	Distance	Percentage of free spaces	Capacity
P2	1.6 Km	0.5	120
P3	1.2 Km	0.7	200

Neighbour	Distance	Percentage of free spaces	Capacity
P4	1.2 Km	0.3	100
P6	0.8 Km	0.75	120

Neighbour	Distance	Percentage of free spaces	Capacity
P1	1.2 Km	0.8	100
P3	2 Km	0.5	200
P7	1 Km	0.6	100

Neighbour	Distance	Percentage of free spaces	Capacity
P2	1 Km	0.5	120
P3	1.8 Km	0.7	120
P6	1.2 Km	0.75	120

Neighbour	Distance	Percentage of free spaces	Capacity
P5	0.8 Km	0.5	120
P7	1.2 Km	0.6	100

Fig. (6.7) The neighbors table

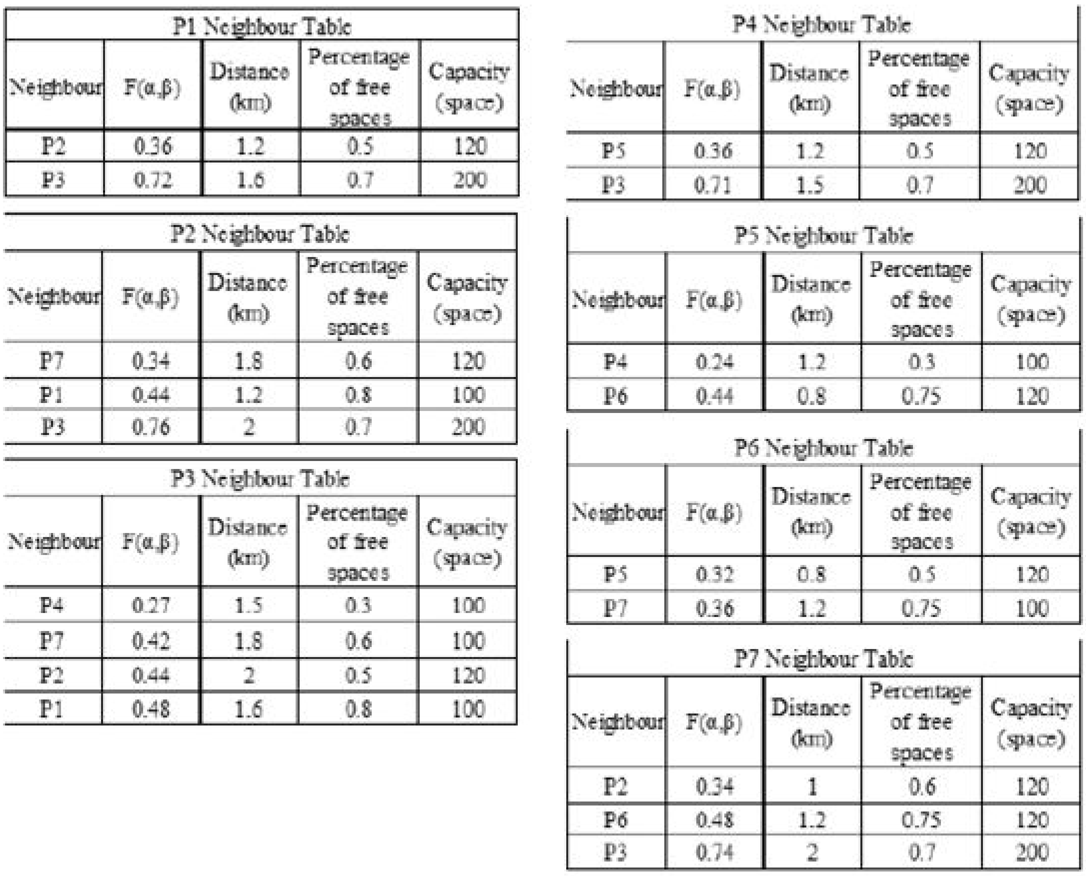
2) Constructing the neighbor table of nodes

A function named $F(\alpha;\beta)$ to calculate the cost between the nodes in the network. $F(\alpha;\beta)$ is a function that depends on the distance between two nodes and the number of free parking spaces in the destination node. $F(\alpha;\beta)$ is considered to be a weighted link between two nodes in the parking network. If two nodes are not directly linked, then $F(\alpha;\beta)$ D 1. If the vehicle comes into a node and that node is full, the vehicle will be forwarded to the next node, which is a neighbor of this node with the smallest value of $F(\alpha;\beta)$ in the neighbor table. By calculating the cost function $F(\alpha;\beta)$ from node Pi to node Pj, i.e.

$$F_{ij} = F(\alpha;\beta) = \alpha X \frac{d_{ij}}{D_{up}} + \beta X \frac{t_j}{T_{up}} \tag{6.1}$$

Where α is a coefficient that depends on the length of the path between two nodes and β is a coefficient that depends on the number of free slots in the destination node. $F(\alpha;\beta)$ is inversely proportional to the distance between two nodes and directly proportional to the total free slots in the destination node. Depending on which parameter it is considered to be the more important of the two parameters, i.e., the distance or the free slots, we can adjust α and β to achieve better network performance. α and β are parameters derived from the experiment, and their value is [0, 1]. If $\alpha = 0$, we only consider the number of free spaces to calculate the cost to the user. If $\beta = 0$, we only consider the distance between two nodes to calculate the cost to the user. In equation (1), we calculate the cost function based on the distance between two nodes and the percentage of free parking spaces at each node. We use the upper bound of the distance between two nodes and the upper bound of the capacity for parking in each car park. In equation (1), dij is the distance between nodes Pi and Pj,Dup is the upper bound of the distance and is a global parameter, tj is the number of spaces that are occupied at node Pj, and Tup is the upper bound of the capacity of the overall parking network and is a global parameter. We assume a network with seven nodes as in figure (5.7) and calculate the value of function F with $\alpha = 0.2$, $\beta = 0.8$, D = 2 km, T = 200 spaces, F12 = 0.36, F13 = 0.72, F21 = 0.44, F23 = 0.76, F27 = 0.34, F31 = 0.48, F32 = 0.44, F34 = 0.27, F37 = 0.42, F43 = 0.71, F45 = 0.36; F54 = 0.24, F56 = 0.44, F65 = 0.32, F67 = 0.36; F72 = 0.34, F73 = 0.74, F76 = 0.48.

The neighbor table of each node with the $F(\alpha;\beta)$ function is shown in figure (6.8). Figure (6.8) shows that the new neighbor table for each node follows equation (6.1). This routing table will be used in choosing the next node where to forward the user when a car park is full.



P1 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P2	0.36	1.2	0.5	120
P3	0.72	1.6	0.7	200

P2 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P7	0.34	1.8	0.6	120
P1	0.44	1.2	0.8	100
P3	0.76	2	0.7	200

P3 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P4	0.27	1.5	0.3	100
P7	0.42	1.8	0.6	100
P2	0.44	2	0.5	120
P1	0.48	1.6	0.8	100

P4 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P5	0.36	1.2	0.5	120
P3	0.71	1.5	0.7	200

P5 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P4	0.24	1.2	0.3	100
P6	0.44	0.8	0.75	120

P6 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P5	0.32	0.8	0.5	120
P7	0.36	1.2	0.75	100

P7 Neighbour Table

Neighbour	F(α,β)	Distance (km)	Percentage of free spaces	Capacity (space)
P2	0.34	1	0.6	120
P6	0.48	1.2	0.75	120
P3	0.74	2	0.7	200

Fig. (6.8) Neighbor table

6.5.3 System operation

When a user wants to find a parking slot, he must login to the system. After successful login, a request message is sent to search for a free parking slot. Then, the system will send back a response message containing the information, including the car park address and the directions to reach it. The choice of the car park is based on the function $F(\alpha;\beta)$, which is calculated based on the current location of the vehicle and the location of the car park. The system will forward the vehicle to a car park with a minimum $F(\alpha;\beta)$ value if the current car park is full. When the user arrives at the car park, he must be authorized to enter. This authorization is achieved via the RFID technology or by scanning the user card. This mechanism is simple but economical. If the information is correct, the user is allowed to park. If the current car park is full, the system will send a suggestion message that includes information on a new car park, including the address and new directions, with a minimum cost. The new car park will be selected based on the neighbor table of the current car park (the first node in the neighbor table), as shown in figure (6.9).

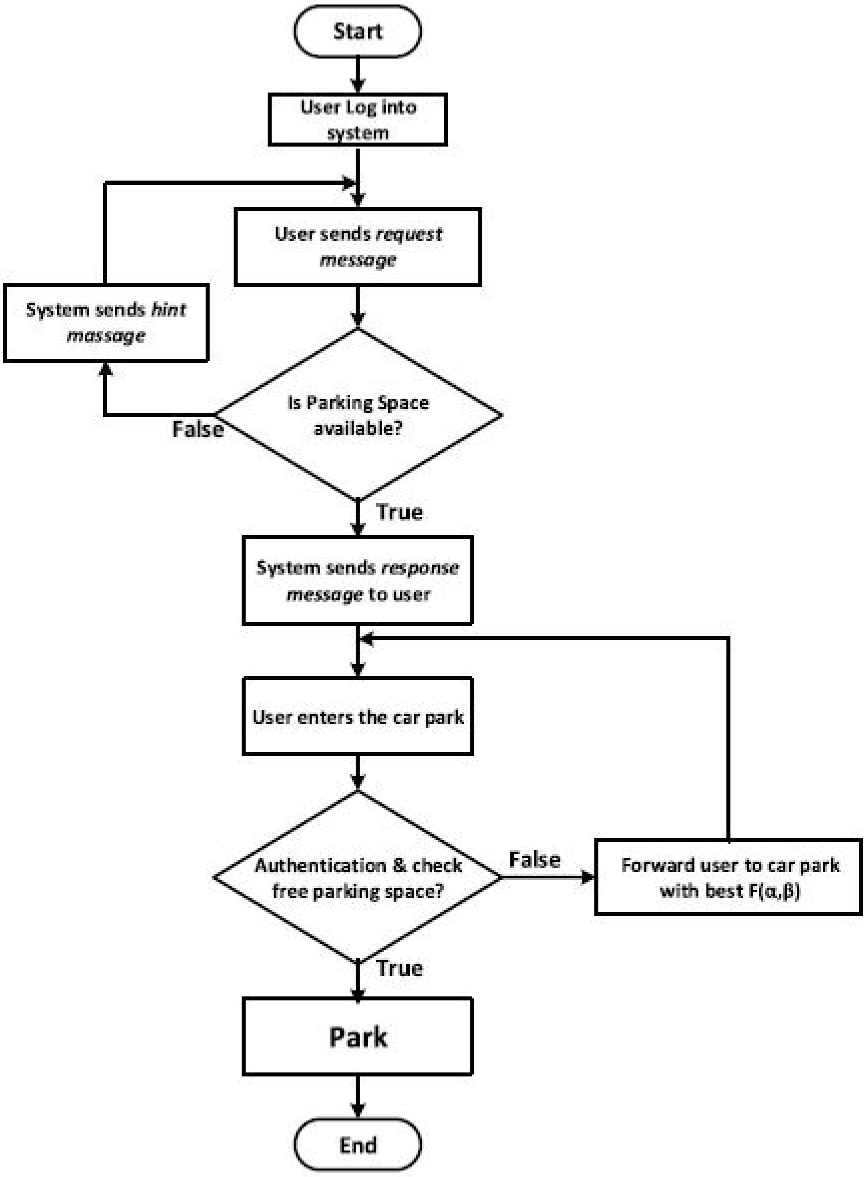

Start
User Log into system
User sends *request message*
System sends *hint massage*
Is Parking Space available?
False
True
System sends *response message* to user
User enters the car park
Authentication & check free parking space?
False
Forward user to car park with best F(α,β)
True
Park
End

Fig. (6.9) System operation algorithm

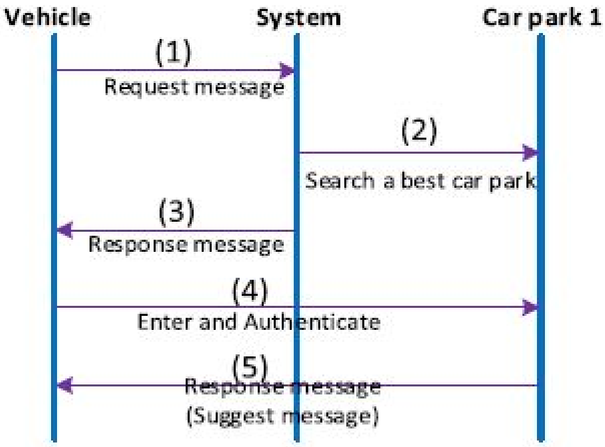


Fig. (6.10)Vehicle process

This proposed system involves two processes: reservation and entering.

- Reservation process: Starting from (1) to (3) shown in figure (6.10), if the user is looking for a free parking space, he will send a request message to the system (1), which is done using a smartphone. When the system receives this request, it will find car park P1 with the least cost [minimum value of $F(\alpha;\beta)$] and forward this message to the user. In this case, the least cost is the minimum value of function $F(\alpha;\beta)$. The value of $F(\alpha;\beta)$ is calculated as the distance (between the vehicle and car parks) and the number of free spaces in each car park. If this car park has free parking slots, it will send a response message to the user (3). The response message includes the address of car park P1 and its directions. Because the percentage of total free spaces in suggesting a new car park, a high probability of success exists in finding a free parking space.

Entering process: Starting from (4) going to (5), if a user enters car park P1; he must be authorized using an ID or an RFID card (4). If authorized, the door is opened, and the count will increase by one. The system will send a response message to the user to notify successful parking (5). If the car park is currently full, it will send a response message suggesting an alternative car park, including relevant information on new car park P2, with the least cost

6.5.4 Calculating the total free parking spaces and updating the neighbor table

In this proposed system, there is RFID technology to calculate the percentage of total free parking spaces in each car park. In each car park, an RFID reader is installed at the entrance. A variable named "Count" to calculate the total number of vehicles in the car park. Count = Count + 1 when a vehicle enters, and Count = Count - 1 when a vehicle leaves. When Count = Ni, car park i is full. The process of updating the neighbor table is described as follows: when a change in the value in the Counter occurs, which changes the percentage of the total free parking spaces at this node, this node will send a message containing updated information to the cloud-based server. The Cloud-Based Server will update the neighboring tables of its neighboring nodes, as shown in figure (6.11).

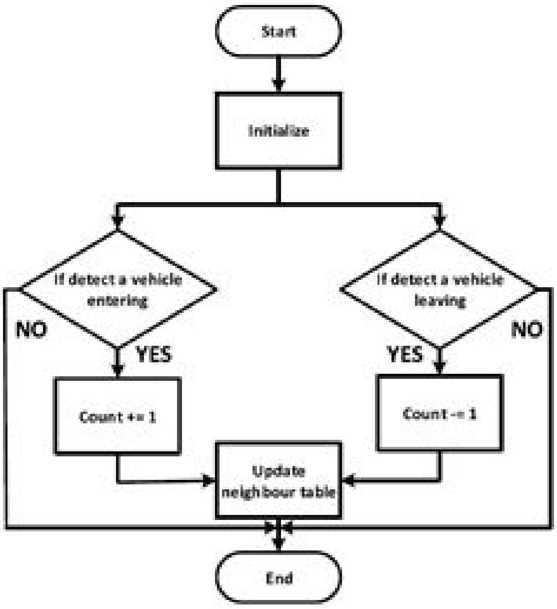


Fig. (6.11) Update the status of car park

6.6 Future plan for Android application

A plan is made in order to make SPS. application more beneficial and helpful for our system users -it may be applied in the future- that covers many points that will add more activities and features to the application, some of them as follows:

1) Reservation through the application:
   In order to providing user an easy way of booking a parking slot through the application to avoid the problem of traffic conjunction that unnecessarily consumes time, an easy reservation feature for parking may be added. through this feature the user can view various parking slots and check for their availability. Whenever a user books a particular slot it will be marked as reserved and all the available slots will remain empty. Booking can be done through credit card/net banking. An additional feature of canceling the booked slot within a specific time from the time of booking may be added . If the user fails to reach the destination on time then the reservation will be cancelled and the payment is refunded. On successful payment a parking number is sent to user's email or to his mobile number for further enquiry. Hence this can reduce the user's effort and time of searching the parking slot and also avoids conjunction of traffic.

2) Representing more that one garage:
   At present SPS system represents one garage but in the future it may be expanded to include more than one garage, maybe a whole city garages! Hence a list and a map of all the system garages may be added to SPS application so the user can view different locations where parking slots are available and select the nearest one to his destination.

3) AI enhancement :
   Adding some AI to the dataBase and some Algorithms will enhance the System and attract more users , for example using computing how many times the user reserve this slot weekly or monthly , or if the user is frequent to some place there will be a recommendation or a notification for the user asking him to reserve some slot .

REFERENCES

References

[1] http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[2] https://www.iotforall.com/what-is-a-gateway/ By Calum McClelland November 23, 2016

[3] https://www.networkworld.com/article/3224893/internet-of-things/what-is-edge-computing-and-how-it-is-changing-the-network.html By Brandon Butler SEP 21, 2017

[4] http://leanbi.ch/en/blog/iot-and-predictive-analytics-fog-and-edge-computing-for-industries-versus-cloud-19-1-2018/

[5] https://www.iotforall.com/openfog-consortium-reference-architecture-executive-summary/ By Yitaek hwang February 28, 2017

[6] https://www.iotforall.com/cloud-fog-computing-iot/by Brian Ray June 14, 2017

[7] https://www.iotforall.com/what-is-an-iot-platform/ By Calum McClelland February 9, 2017

[8] https://medium.com/iotforall/iot-connectivity-101-3f6fcee49a17 By Calum McClelland February 13, 2017

[9] https://learn.sparkfun.com/tutorials/connectivity-of-the-internet-of-things 27 july 2016

[10] Jonathan Valvano, Embedded Microcomputer Systems: Real Time Interfacing, 3rd Edition, (2016).

[11] Jonathan Valvano, Real Time Interfacing Operating Systems for ARM Cortex-M Microcontrollers Volume Three, 4th Edition, (2017).

[12] Richard Reese, Understanding and Using C Pointers, (2013).

[13] https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s

2009/asn28_asp36/FinalProjectReport/index.html[Accessed 6 2018].

[14] https://www.edx.org/course/embedded-systems-shape-the-world-microcontrollerinputoutput.

[15] http://users.ece.utexas.edu/~valvano/arm/

[16] https://www.embeddedrelated.com/

[17] http://www.instructables.com/id/Smart-Parking-System/

[18] http://web.stanford.edu/class/cs193a/lectures.shtml , Spring 2016.

[19] https://github.com/koush/ion ion , Android Library.

[20] https://www.stechies.com/android-activity-lifecycle-overview/

[21] https://github.com/mitchtabian/Sending-and-Receiving-Data-with-Bluetooth, Dec 2016.

[22] https://developer.android.com/guide/topics/ui/ , GUI programming.

[23] https://www.tutorialspoint.com/android/android_ui_design.htm,GUI programming

[24] J. P. Benson, T. O'Donovan, P. O'Sullivan, U. Roedig, C. Sreenan, J. Barton, et al., "Car-park management using wireless sensor networks," in Local Computer Networks, Proceedings 2006 31st IEEE Conference on, 2006, pp. 588-595.

[25] S. Funck, N. Mohler, and W. Oertel, "Determining car-park occupancy from single images," in Intelligent Vehicles Symposium, 2004 IEEE, 2004, pp. 325-328.

[26] R. Panayappan, J. M. Trivedi, A. Studer, and A. Perrig, "VANET-based approach for parking space availability," in Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks, 2007, pp. 75-76.

[27] https://docs.microsoft.com/en-us/sql/database-engine/database-mirroring/the-database-mirroring-endpoint-sql-server?view=sql-server-2017

[28] http://www.esa.int/our_activities/navigation

[29] Nam Pham, Ming-Fong Tasi, Duc Binh Nguyen, Chyi-Ren Dow, and Der-Jiunn Deng, "A Cloud-Based Smart-Parking System Based on Internet-of-Things Technologies," in Proc. Int. Conf, Green Computing and Internet of Things (ICGCIoT), 2015, pp. 352-354.

APPENDICES

Appendix (A)

Python code for the Raspberry Pi:

```
import time
import sys
import RPi.GPIO as GPIO
import requests
import spi
import SimpleMFRC522
reader = SimpleMFRC522.SimpleMFRC522()

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11,GPIO.IN)   #IR pin at pin11 for Sensor1
GPIO.setup(13,GPIO.IN)   #IR pin at pin13 for Sensor2
GPIO.setup(15,GPIO.IN)   #IR pin at pin15 for Sensor3
GPIO.setup(29,GPIO.IN)   #IR pin at pin29 for Sensor4

GPIO.setup(32,GPIO.OUT)  #LED for Sensor1
GPIO.setup(36,GPIO.OUT)  #LED for Sensor2
GPIO.setup(38,GPIO.OUT)  #LED for Sensor3
GPIO.setup(40,GPIO.OUT)  #LED for Sensor4

GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP) #Rfid Push button
GPIO.setup(37, GPIO.IN, pull_up_down=GPIO.PUD_UP) #Input pin of the smoke detector
GPIO.setup(18,GPIO.OUT)  #LED for rfid entry
GPIO.setup(16,GPIO.OUT)  #LED for fire detector

def Check_rfid():
    reader = SimpleMFRC522.SimpleMFRC522()
    id,text=reader.read()
    return text

def LED1_ON():
        GPIO.output(32,1)
        return
def LED2_ON():
        GPIO.output(36,1)
        return
def LED3_ON():
        GPIO.output(38,1)
        return
def LED4_ON():
        GPIO.output(40,1)
        return

def LED1_OFF():
        GPIO.output(32,0)
        return
```

```
def LED2_OFF():
        GPIO.output(36,0)
        return
def LED3_OFF():
        GPIO.output(38,0)
        return
def LED4_OFF():
        GPIO.output(40,0)
        return

led_on=[LED1_ON,LED2_ON,LED3_ON,LED4_ON]
led_off=[LED1_OFF,LED2_OFF,LED3_OFF,LED4_OFF]

try:
    flag=[0,0,0,0]
    fire_flag=0

r=requests.get('https://mahmoudelwalili.000webhostapp.com/?state1=empty&state2=empty&state3=empty&state4=empty')
    print(r.url)
    time.sleep(1.5)
    while True:
        button_state = GPIO.input(12)
        if (button_state == False):
              GPIO.output(18, True)
              print('Button Pressed..’)
              info=Check_rfid()
              print(info)
        else:
              GPIO.output(18, False)

        slot1 = GPIO.input(11)
        slot2 = GPIO.input(13)
        slot3 = GPIO.input(15)
        slot4 = GPIO.input(29)
        parkingslots = [slot1,slot2,slot3,slot4]

        for x in range(0, 4):
            if (parkingslots[x]==1):     #to check if there is an empty parking slot for each parking id
                led_off[x]()
                if(flag[x]==1):
                    parameters="?id="+repr(x+1)+"&state=empty"
                    URL='https://mahmoudelwalili.000webhostapp.com/'+parameters
                    r=requests.get(URL)
                    print(r.url)
                    flag[x]=0
            else:                   #if filled parking slot
                led_on[x]()
                if(flag[x]==0):
                    parameters="?id="+repr(x+1)+"&state=filled&info="+info
                    URL='https://mahmoudelwalili.000webhostapp.com/'+parameters
                    r=requests.get(URL)
                    print(r.url)
                    flag[x]=1

        if (GPIO.input(37)==0):         #if there is a gas detected
            if (fire_flag==1):
```

```
        GPIO.output(16,True)
        print("Gas detected!!”)
        r=requests.get('https://mahmoudelwalili.000webhostapp.com/?fire_alarm=1’)
        print(r.url)
        fire_flag=0
    else:
      if (fire_flag==0):
        GPIO.output(16,False)
        r=requests.get('https://mahmoudelwalili.000webhostapp.com/?fire_alarm=0’)
        print(r.url)
        fire_flag=1

    time.sleep(0.5)
except KeyboardInterrupt:
  GPIO.cleanup()
  sys.exit()
```

Appendix B

```
<?php
  $MyHostname = "localhost";
  $MyUsername = "id3794679_root";
  $MyPassword = "IoT_parking_system";
  $MyDatabase ="parking_database";

  #connect to the server
  $conn = mysqli_connect($MyHostname,$MyUsername,$MyPassword,$MyDatabase);
  if (!$conn)  //if connection is failed then print an indication
  {
    echo "Connection failed <br>";
  }
   include ('add_data.php');
?>
```

Appendix C

```
<?php
  #------------------------------functions------------------------------------------------
  #Function to insert the initial "empty" state to table in the first time
  function add_data($state1,$state2,$state3,$state4,$conn) {
    $states=array($state1,$state2,$state3,$state4);
    for ($i=0;$i<4;$i++)
     {
     $SQL = "INSERT INTO parkingtable (state) VALUES ('$states[$i]')";
     #Execute SQL statement
     if (!mysqli_query($conn, $SQL)) {echo "false record";}
     }
    $SQL="UPDATE parkingtable SET time= NOW()";      //adding the intitial time
    if (!mysqli_query($conn, $SQL)) {echo "false insert the initial time";}
  }
  #Function to update the data in a certain row
  function update_data($idd,$state,$info,$conn)
  {
     $SQL="UPDATE parkingtable SET state = '$state' , Carinfo = '$info' WHERE id = '$idd'";
     if (!mysqli_query($conn, $SQL)) {echo "false update";}

     $SQL="UPDATE parkingtable SET time= NOW() WHERE id = '$idd'";     //adding the start time of the
change
     if (!mysqli_query($conn, $SQL)) {echo "false make the initial time for this slot";}
  }

  #Function to subtract the current time with the past one in second and adding the result in time diff column
  function time_diff($conn)
  {
     for ($i=1;$i<5;$i++){
        $SQL="SELECT TIMESTAMPDIFF(SECOND,(SELECT time FROM parkingtable where id='$i'),
NOW())";
        $timediffer = mysqli_query($conn,$SQL);
        if (!$timediffer) {echo "failed to get the time difference";}
        $row = mysqli_fetch_array($timediffer);
        $timediff=$row["TIMESTAMPDIFF(SECOND,(SELECT time FROM parkingtable where id='$i'),
NOW())"];

        $SQL="UPDATE parkingtable SET timediff = '$timediff' WHERE id = '$i'";
        if (!mysqli_query($conn, $SQL)) {echo "false adding time diff";}
     }
  }

  function Count_empty($conn)
  {
     $SQL="SELECT COUNT(*) FROM parkingtable WHERE state='empty'";
     $result=mysqli_query($conn,$SQL);
     if (!$result) {echo"false count";}
     $values=mysqli_fetch_array($result);
     $count=$values['COUNT(*)'];
     return $count;
  }

  function add_alarm($conn,$fire_alarm)
  {
     $SQL="UPDATE parkingtable SET alarm = '$fire_alarm' WHERE id = 1";
     if (!mysqli_query($conn, $SQL)) {echo "false adding alarm value";}
  }

  function check_fire_alarm($conn)
  {
     $SQL="SELECT COUNT(*) FROM parkingtable WHERE alarm=1";
     $result=mysqli_query($conn,$SQL);
     if (!$result) {echo"false count";}
     $values=mysqli_fetch_array($result);
     $alarm=$values['COUNT(*)'];
     return $alarm;
  }
  #--------------------------Preparation of the initial variables------------------------------------------
  $state1=$_GET["state1"];
  $state2=$_GET["state2"];
```

```
    $state3=$_GET["state3"];
    $state4=$_GET["state4"];
    $state=$_GET["state"];
    $fire_alarm=$_GET["fire_alarm"];
    $info=$_GET["info"];
    $idd=$_GET["id"];

    $data = [
     'id' => $idd,
     'state' => $state
    ];
    $data_json=json_encode($data);
    //echo $data;
    $file = fopen("save.html","w");
    fwrite($file, $data_json);
    fclose($file);

    $url = 'http://' . $_SERVER['SERVER_NAME'] . $_SERVER['REQUEST_URI'];
    #$echo $url;echo "<br>";

    if (strpos($url, "state1") !== false) #to check if the url string has word "state1"
     {
        $SQL = "TRUNCATE TABLE parkingtable";
        #Execute SQL statement
        if (!mysqli_query($conn, $SQL)) {echo "false truncate<br>";}
        add_data($state1,$state2,$state3,$state4,$conn);
     }elseif (strpos($url, "state") !== false)
     {
        #echo "enter update"; echo"<br>";
        update_data($idd,$state,$info,$conn);
     }
     elseif (strpos($url, "fire_alarm") !== false)
     {
         add_alarm($conn,$fire_alarm);
     }
     else
     {
         time_diff($conn);
         $count_empty=Count_empty($conn);
         $alarm=check_fire_alarm($conn);
     }
?>
```

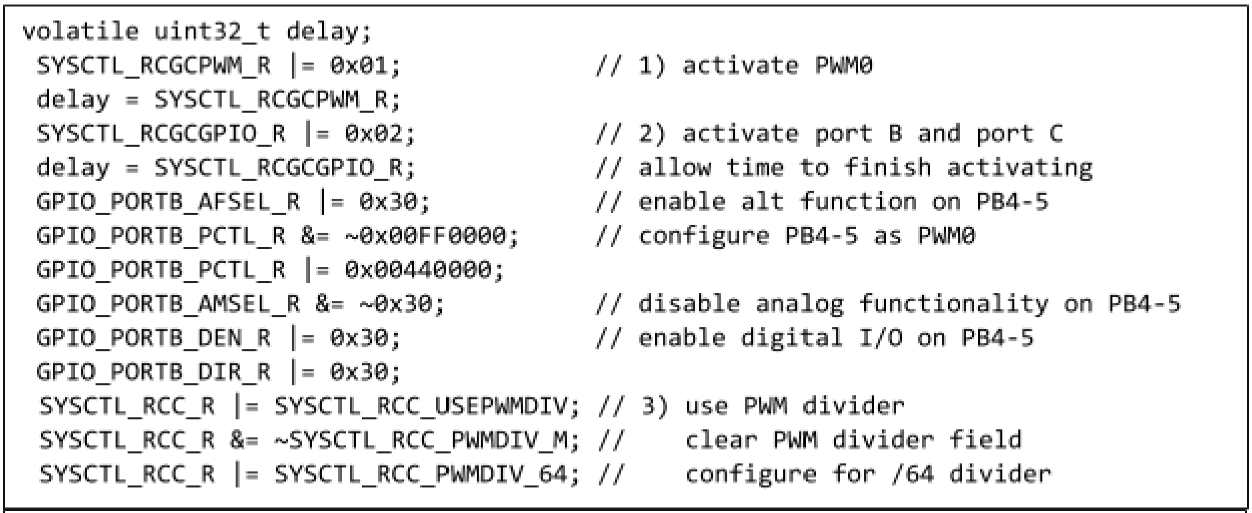
Appendix D

```
<?php
    // Start MySQL Connection
    include('connect.php');
    include ('add_date.php');
?>

<html>
<head>
    <meta http-equiv="refresh" content="4">
   <title>IOT Parking System</title>
```

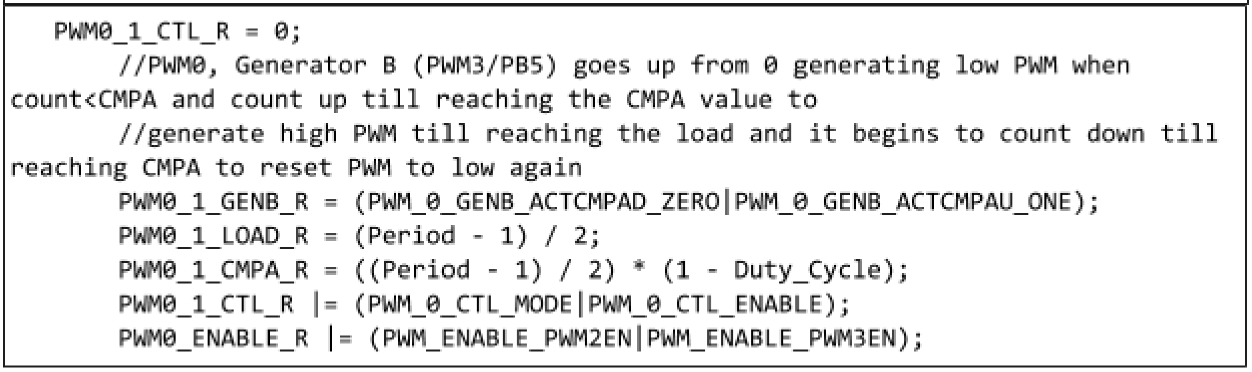
```
    <style type="text/css">
        .table_titles, .table_cells_odd, .table_cells_even {
                padding-right: 30px;
                padding-left: 30px;
                color: #000;
        }
        .table_titles {
            color: #FFF;
            background-color: #666;
        }
        .table_cells_odd {
            background-color: #CCC;
        }
        .table_cells_even {
            background-color: #FAFAFA;
        }
        table {
            border: 3px solid #333;
        }
        body { font-family: "Trebuchet MS", Arial; }
        div3 {
                 width: 320px;
                 padding: 10px;
                 border: 5px solid pink;
                 margin: 0;
            }
    </style>
</head>

    <body align="center">
        <br>
         <center><h1>Parking Table </h1></center>
    <table border="0" cellspacing="-3" cellpadding="4" align="center">
      <tr>
            <td class="table_titles">Parking slots id</td>
            <td class="table_titles">state</td>
            <td class="table_titles">Car Info. (color-plate number)</td>
            <td class="table_titles">Time since last change</td>
            <td class="table_titles">Cost for parking</td>
          </tr>
<?php
    define('Month', 2592000);
    define('Week', 604800);
    define('Day', 86400);
    define('Hour', 3600);
    define('Minute', 60);

//Function to represent the time spent by the parking car in the slot in a form of (X hours Y minutes Z seconds
ago)
    function SecondsToTime($seconds, $num_units=3) {
        $time_descr = array(
        "Months" => floor($seconds / Month),
        "Weeks" => floor(($seconds % Month) / Week),
        "Days" => floor(($seconds % Week) / Day),
        "Hours" => floor(($seconds % Day) / Hour),
        "Minutes" => floor(($seconds % Hour) / Minute),
        "Secs" => floor($seconds % Minute),
        );
        $res = "";
        $counter = 0;
        foreach ($time_descr as $k => $v) {
        if ($v) {
        $res.=$v." ".$k;
        $counter++;
        if($counter>=$num_units)
        break;
        elseif($counter)
        $res.="  ";
            }
        }
        return $res;
    }

//Function to represent the cost of the time spent by the parking car in the slot
    function count_cost($string)
    {
        if ($string[2] == 'S' || $string[3] =='S')
            $s=0;
        if ($string[2] == 'M' || $string[3] =='M')
            {
                $s=1.5*strstr($string,'Minutes',true);
            }
        if ($string[2] == 'H' || $string[3] =='H')
            {
                $h_in_min=60*1.5*strstr($string,$string[1],true);
                $ss=strstr($string,'Minutes',true);
                $min=1.5*substr($ss, strpos($ss, "Hours") +5);
                $s=$min+$h_in_min;
            }
        return $s;
    }
    // Retrieve all records and display them
    $query = mysqli_query($conn,"SELECT * FROM parkingtable");
    if ($query)
    {
        // process every record
        while( $row = mysqli_fetch_array($query) and $k<4)
         {
             $cost=0;
             if ($row["state"]=="filled")
                $cost=count_cost(SecondsToTime($row["timediff"]));
        if ($oddrow)
            {
                $css_class=' class="table_cells_odd"';
            }
            else
            {
                $css_class=' class="table_cells_even"';
            }
            $oddrow = !$oddrow;

            echo "<br>";
            echo '<tr>';
            echo '    <td'.$css_class.'>'.$row["id"].'</td>';
            echo '   <td'.$css_class.'>'.$row["state"].'</td>';
            echo '   <td'.$css_class.'>'.$row["Carinfo"].'</td>';
            echo '   <td'.$css_class.'>'.SecondsToTime($row["timediff"]).'ago</td>';
            echo '   <td'.$css_class.'>'.$cost.' LE</td>';
            #echo '   <td'.$css_class.'>'.$row["timediff"].'</td>';
            echo '</tr>';
        }
    } else {
        die ('cannot select table' . mysql_error($dbh));
    }

    // Used for row color toggle <b><font color="red">
    $oddrow = true;
?>
    </table>
    <center><h3>
        <?php
        echo "<br>";
        echo "Number of empty slots : $count_empty <br>";
        echo "<br>";
        echo "<br>";
        if ($alarm==1) echo '<div3> <b><font color="red"> ALARM: Smoke gas is detected
!!</font></b></div3>';
        ?></h3></center>
    </body>
</html>
```

Appendix E

DC Motor Initializations

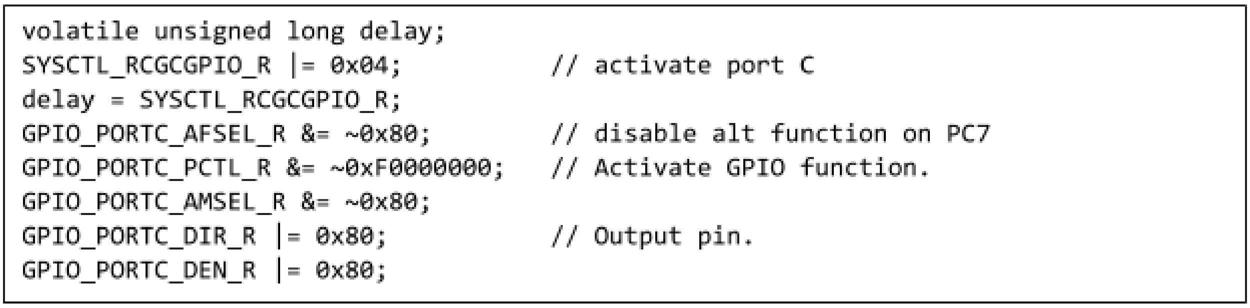


```
volatile uint32_t delay;
 SYSCTL_RCGCPWM_R |= 0x01;             // 1) activate PWM0
 delay = SYSCTL_RCGCPWM_R;
 SYSCTL_RCGCGPIO_R |= 0x02;            // 2) activate port B and port C
 delay = SYSCTL_RCGCGPIO_R;            // allow time to finish activating
 GPIO_PORTB_AFSEL_R |= 0x30;           // enable alt function on PB4-5
 GPIO_PORTB_PCTL_R &= ~0x00FF0000;     // configure PB4-5 as PWM0
 GPIO_PORTB_PCTL_R |= 0x00440000;
 GPIO_PORTB_AMSEL_R &= ~0x30;          // disable analog functionality on PB4-5
 GPIO_PORTB_DEN_R |= 0x30;             // enable digital I/O on PB4-5
 GPIO_PORTB_DIR_R |= 0x30;
 SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV; // 3) use PWM divider
 SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M; //    clear PWM divider field
 SYSCTL_RCC_R |= SYSCTL_RCC_PWMDIV_64; //    configure for /64 divider
```



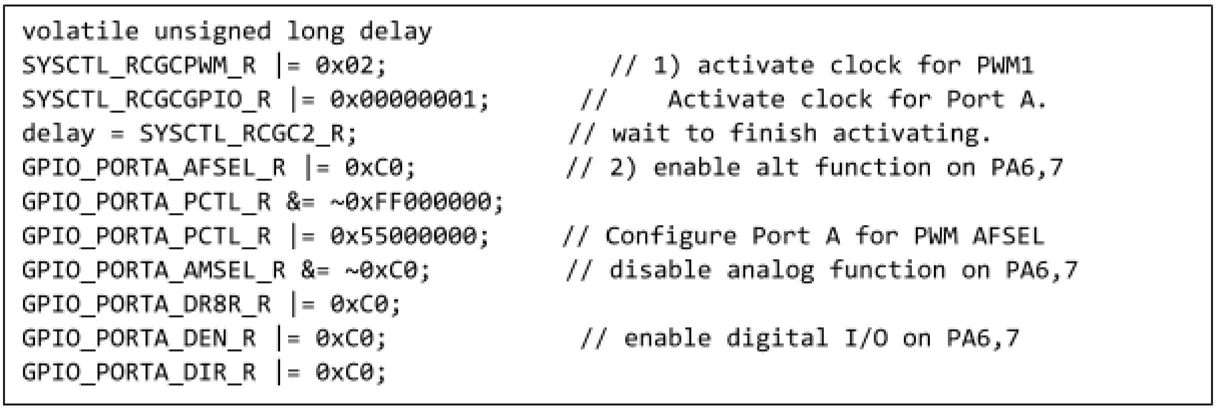
```
  PWM0_1_CTL_R = 0;
      //PWM0, Generator B (PWM3/PB5) goes up from 0 generating low PWM when
count<CMPA and count up till reaching the CMPA value to
      //generate high PWM till reaching the load and it begins to count down till
reaching CMPA to reset PWM to low again
      PWM0_1_GENB_R = (PWM_0_GENB_ACTCMPAD_ZERO|PWM_0_GENB_ACTCMPAU_ONE);
      PWM0_1_LOAD_R = (Period - 1) / 2;
      PWM0_1_CMPA_R = ((Period - 1) / 2) * (1 - Duty_Cycle);
      PWM0_1_CTL_R |= (PWM_0_CTL_MODE|PWM_0_CTL_ENABLE);
      PWM0_ENABLE_R |= (PWM_ENABLE_PWM2EN|PWM_ENABLE_PWM3EN);
```

DC Motor Direction Routine

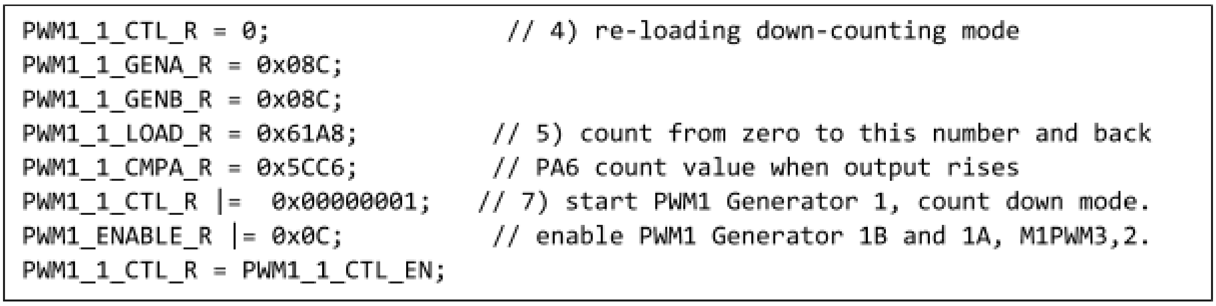


```
volatile unsigned long delay;
SYSCTL_RCGCGPIO_R |= 0x04;          // activate port C
delay = SYSCTL_RCGCGPIO_R;
GPIO_PORTC_AFSEL_R &= ~0x80;        // disable alt function on PC7
GPIO_PORTC_PCTL_R &= ~0xF0000000;   // Activate GPIO function.
GPIO_PORTC_AMSEL_R &= ~0x80;
GPIO_PORTC_DIR_R |= 0x80;           // Output pin.
GPIO_PORTC_DEN_R |= 0x80;
```

Servo Motor Initializations

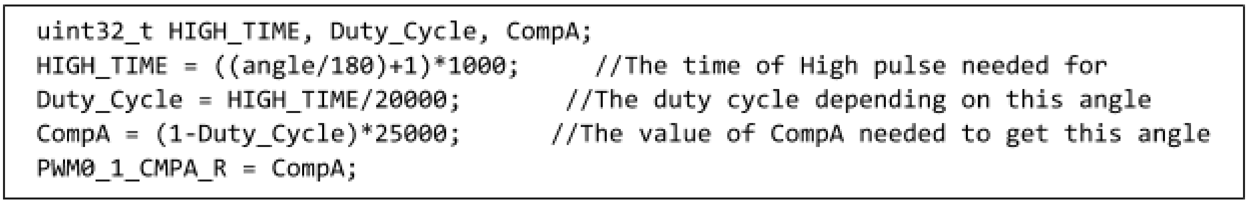


```
volatile unsigned long delay
SYSCTL_RCGCPWM_R |= 0x02;               // 1) activate clock for PWM1
SYSCTL_RCGCGPIO_R |= 0x00000001;      //    Activate clock for Port A.
delay = SYSCTL_RCGC2_R;              // wait to finish activating.
GPIO_PORTA_AFSEL_R |= 0xC0;          // 2) enable alt function on PA6,7
GPIO_PORTA_PCTL_R &= ~0xFF000000;
GPIO_PORTA_PCTL_R |= 0x55000000;     // Configure Port A for PWM AFSEL
GPIO_PORTA_AMSEL_R &= ~0xC0;         // disable analog function on PA6,7
GPIO_PORTA_DR8R_R |= 0xC0;
GPIO_PORTA_DEN_R |= 0xC0;              // enable digital I/O on PA6,7
GPIO_PORTA_DIR_R |= 0xC0;
```



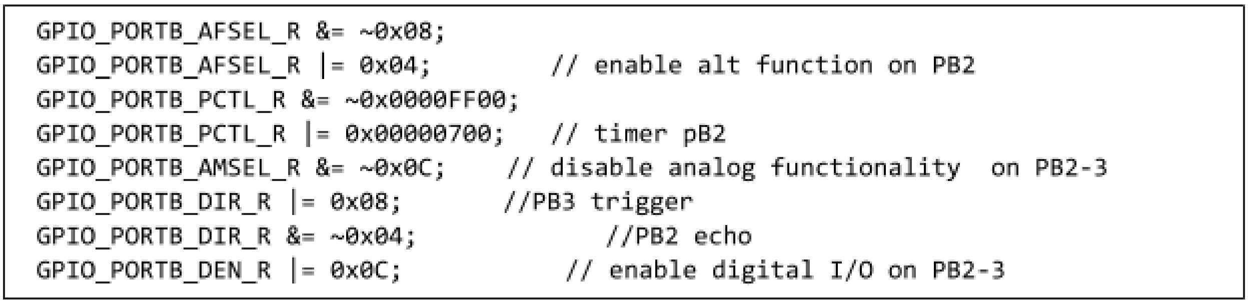
```
PWM1_1_CTL_R = 0;                 // 4) re-loading down-counting mode
PWM1_1_GENA_R = 0x08C;
PWM1_1_GENB_R = 0x08C;
PWM1_1_LOAD_R = 0x61A8;         // 5) count from zero to this number and back
PWM1_1_CMPA_R = 0x5CC6;         // PA6 count value when output rises
PWM1_1_CTL_R |=  0x00000001;   // 7) start PWM1 Generator 1, count down mode.
PWM1_ENABLE_R |= 0x0C;          // enable PWM1 Generator 1B and 1A, M1PWM3,2.
PWM1_1_CTL_R = PWM1_1_CTL_EN;
```

Servo Motor Angle Driving

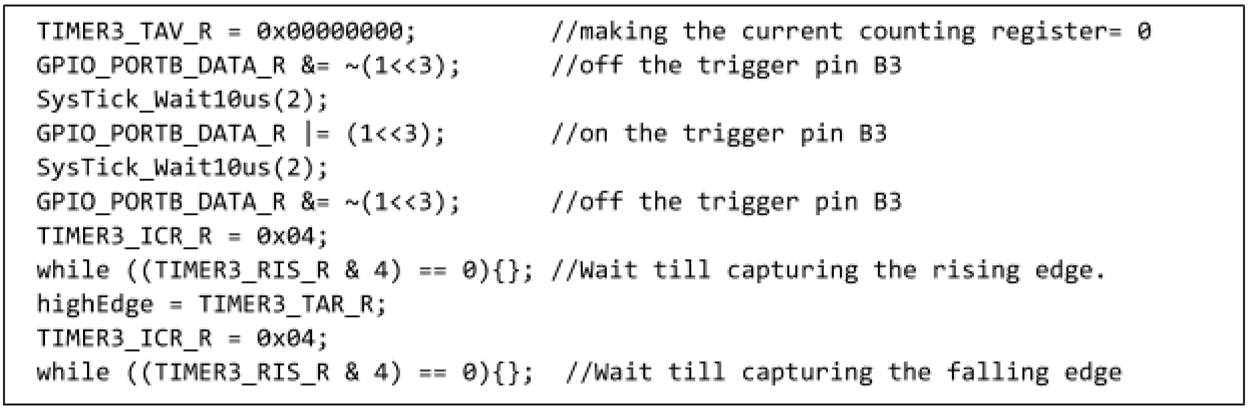


```
uint32_t HIGH_TIME, Duty_Cycle, CompA;
HIGH_TIME = ((angle/180)+1)*1000;     //The time of High pulse needed for
Duty_Cycle = HIGH_TIME/20000;       //The duty cycle depending on this angle
CompA = (1-Duty_Cycle)*25000;      //The value of CompA needed to get this angle
PWM0_1_CMPA_R = CompA;
```

Timer Initializations

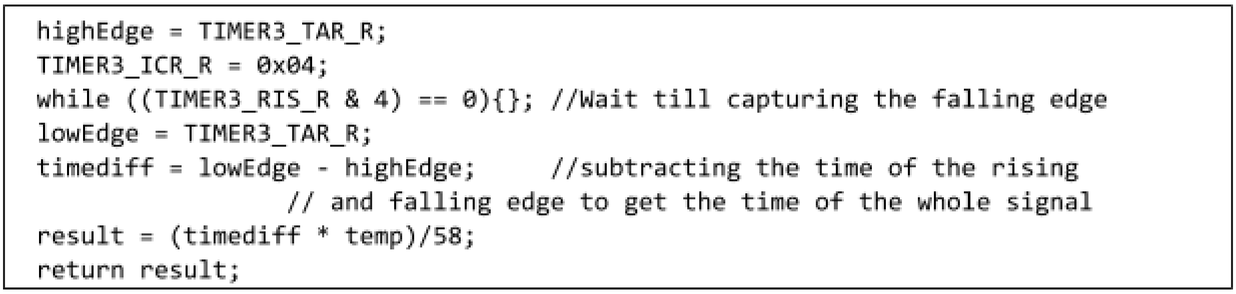


```
GPIO_PORTB_AFSEL_R &= ~0x08;
GPIO_PORTB_AFSEL_R |= 0x04;        // enable alt function on PB2
GPIO_PORTB_PCTL_R &= ~0x0000FF00;
GPIO_PORTB_PCTL_R |= 0x00000700;   // timer pB2
GPIO_PORTB_AMSEL_R &= ~0x0C;    // disable analog functionality  on PB2-3
GPIO_PORTB_DIR_R |= 0x08;       //PB3 trigger
GPIO_PORTB_DIR_R &= ~0x04;             //PB2 echo
GPIO_PORTB_DEN_R |= 0x0C;           // enable digital I/O on PB2-3
```



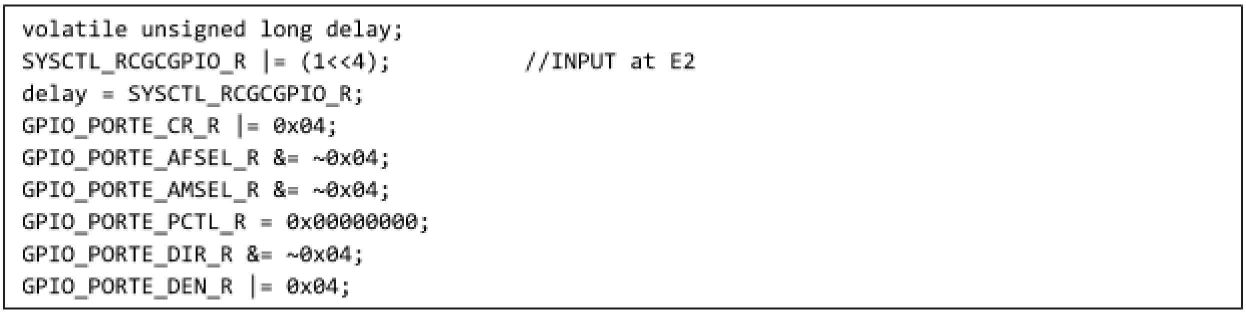
```
TIMER3_TAV_R = 0x00000000;         //making the current counting register= 0
GPIO_PORTB_DATA_R &= ~(1<<3);      //off the trigger pin B3
SysTick_Wait10us(2);
GPIO_PORTB_DATA_R |= (1<<3);       //on the trigger pin B3
SysTick_Wait10us(2);
GPIO_PORTB_DATA_R &= ~(1<<3);      //off the trigger pin B3
TIMER3_ICR_R = 0x04;
while ((TIMER3_RIS_R & 4) == 0){}; //Wait till capturing the rising edge.
highEdge = TIMER3_TAR_R;
TIMER3_ICR_R = 0x04;
while ((TIMER3_RIS_R & 4) == 0){};  //Wait till capturing the falling edge
```

Conversion of Received Pulses of Ultrasonic Sensors Routine



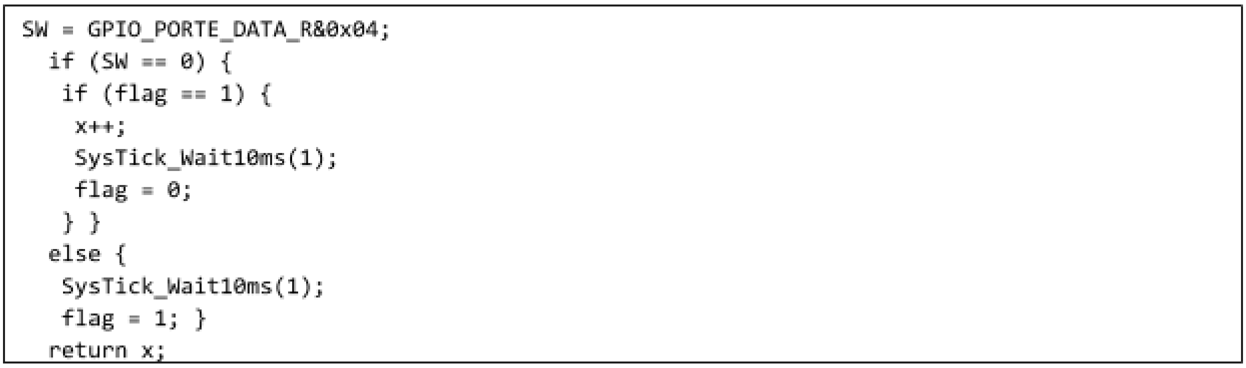
```
highEdge = TIMER3_TAR_R;
TIMER3_ICR_R = 0x04;
while ((TIMER3_RIS_R & 4) == 0){}; //Wait till capturing the falling edge
lowEdge = TIMER3_TAR_R;
timediff = lowEdge - highEdge;     //subtracting the time of the rising
                 // and falling edge to get the time of the whole signal
result = (timediff * temp)/58;
return result;
```

IR Sensor Initializations



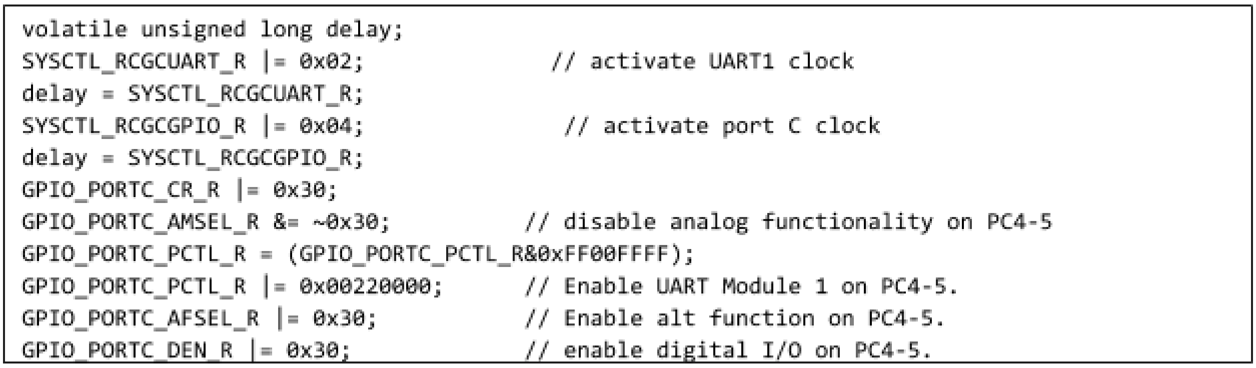
```
volatile unsigned long delay;
SYSCTL_RCGCGPIO_R |= (1<<4);          //INPUT at E2
delay = SYSCTL_RCGCGPIO_R;
GPIO_PORTE_CR_R |= 0x04;
GPIO_PORTE_AFSEL_R &= ~0x04;
GPIO_PORTE_AMSEL_R &= ~0x04;
GPIO_PORTE_PCTL_R = 0x00000000;
GPIO_PORTE_DIR_R &= ~0x04;
GPIO_PORTE_DEN_R |= 0x04;
```

Reading IR Sensor C Definition

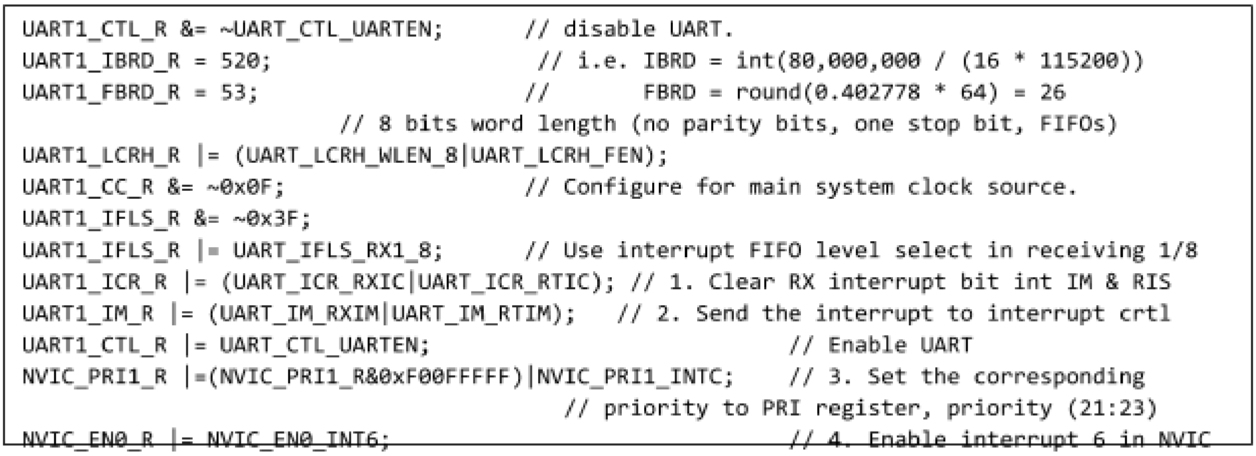


```
SW = GPIO_PORTE_DATA_R&0x04;
  if (SW == 0) {
   if (flag == 1) {
    x++;
    SysTick_Wait10ms(1);
    flag = 0;
   } }
  else {
   SysTick_Wait10ms(1);
   flag = 1; }
  return x;
```

UART1 Initializations

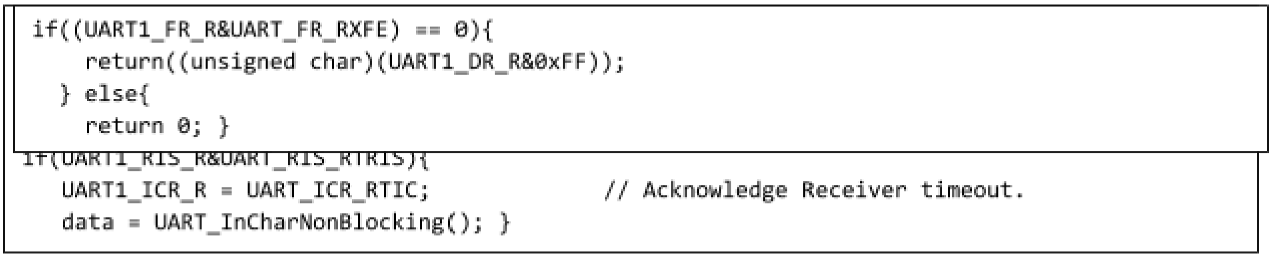


```
volatile unsigned long delay;
SYSCTL_RCGCUART_R |= 0x02;              // activate UART1 clock
delay = SYSCTL_RCGCUART_R;
SYSCTL_RCGCGPIO_R |= 0x04;               // activate port C clock
delay = SYSCTL_RCGCGPIO_R;
GPIO_PORTC_CR_R |= 0x30;
GPIO_PORTC_AMSEL_R &= ~0x30;          // disable analog functionality on PC4-5
GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R&0xFF00FFFF);
GPIO_PORTC_PCTL_R |= 0x00220000;      // Enable UART Module 1 on PC4-5.
GPIO_PORTC_AFSEL_R |= 0x30;           // Enable alt function on PC4-5.
GPIO_PORTC_DEN_R |= 0x30;             // enable digital I/O on PC4-5.
```



```
UART1_CTL_R &= ~UART_CTL_UARTEN;      // disable UART.
UART1_IBRD_R = 520;                    // i.e. IBRD = int(80,000,000 / (16 * 115200))
UART1_FBRD_R = 53;                    //       FBRD = round(0.402778 * 64) = 26
                        // 8 bits word length (no parity bits, one stop bit, FIFOs)
UART1_LCRH_R |= (UART_LCRH_WLEN_8|UART_LCRH_FEN);
UART1_CC_R &= ~0x0F;                  // Configure for main system clock source.
UART1_IFLS_R &= ~0x3F;
UART1_IFLS_R |= UART_IFLS_RX1_8;      // Use interrupt FIFO level select in receiving 1/8
UART1_ICR_R |= (UART_ICR_RXIC|UART_ICR_RTIC); // 1. Clear RX interrupt bit int IM & RIS
UART1_IM_R |= (UART_IM_RXIM|UART_IM_RTIM);   // 2. Send the interrupt to interrupt crtl
UART1_CTL_R |= UART_CTL_UARTEN;                          // Enable UART
NVIC_PRI1_R |=(NVIC_PRI1_R&0xF00FFFFF)|NVIC_PRI1_INTC;    // 3. Set the corresponding
                                          // priority to PRI register, priority (21:23)
NVIC_EN0_R |= NVIC_EN0_INT6;                             // 4. Enable interrupt 6 in NVIC
```

UART Receive *char* Routine

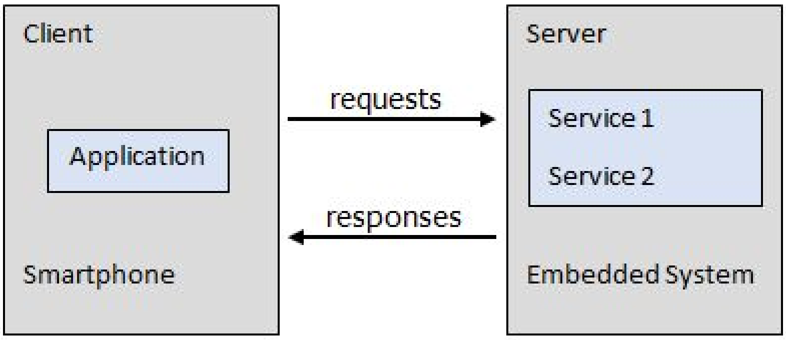


```
if((UART1_FR_R&UART_FR_RXFE) == 0){
    return((unsigned char)(UART1_DR_R&0xFF));
  } else{
    return 0; }
if(UART1_RIS_R&UART_RIS_RTRIS){
  UART1_ICR_R = UART_ICR_RTIC;           // Acknowledge Receiver timeout.
  data = UART_InCharNonBlocking(); }
```

Table of Used Ports in TM4C123X Board

Port	Pin	Function in use	Service
PORTB	4	Pulse width Modulation	DC Motors driving
PORTB	5	Pulse width Modulation	Servo Motor driving
PROTC	7	General Purpose I/O	DC Motor direction control
PORTC	4 and 5	UART1 RX and TX	Bluetooth module
PORTE	2	General Purpose I/O	IR sensor
PORTB	2,3,6 and 7	Timer	Ultrasonic sensors
PORTF	2 and 3	Timer	Ultrasonic sensors
PORTA	2 and 3	General Purpose I/O – optional	Steering control

DECLARATION OF FSM IN SOFTWARE


Client
Application
Smartphone
requests
responses
Server
Service 1
Service 2
Embedded System