THE UNIVERSITY OF DANANG
DANANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

CAPSTONE PROJECT

SIMULATE A SELF-PARKING CAR

LÊ VĂN VŨ AN
LÊ THÁI XƯƠNG
TRẦN KHANG NGUYÊN
PHẠM THANH TRÂM
PHẠM THANH VINH

Supervisor: **Mr. PHẠM MINH HẢI**
Co-Supervisors: **Dr. NGUYỄN LÊ HÒA**
Dr. TRƯƠNG THỊ BÍCH THANH

Submitted to the Center Of Excellence, Advanced Enginerring Program in (Embedded System OR Electronic Communication) Engineering in Partial Fulfillment of the Requirements for the Degree of Engineer

Danang, June 2016

THE UNIVERSITY OF DANANG
DANANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

CAPSTONE PROJECT

SIMULATE A SELF-PARKING CAR

LÊ VĂN VŨ AN
LÊ THÁI XƯƠNG
TRẦN KHANG NGUYÊN
PHẠM THANH TRÂM
PHẠM THANH VINH

Supervisor: **Mr. PHẠM MINH HẢI**
Co-Supervisors: **Dr. NGUYỄN LÊ HÒA**
Dr. TRƯƠNG THỊ BÍCH THANH

Danang, June 2016

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Số: ..96../ĐHBK-ĐT

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

Đà Nẵng, ngày 20 tháng 01 năm 2016

QUYẾT ĐỊNH

V/v giao đề tài và thành lập Hội đồng hướng dẫn Capstone project cho sinh viên Chương trình tiên tiến

HIỆU TRƯỞNG TRƯỜNG ĐẠI HỌC BÁCH KHOA

Căn cứ Nghị định số 32/CP ngày 4/4/1994 của Chính phủ về việc thành lập Đại học Đà Nẵng;

Căn cứ Thông tư số 08/2014/TT-BGDĐT ngày 20 tháng 03 năm 2014 của Bộ Trưởng Bộ Giáo dục và Đào tạo về việc ban hành Quy chế tổ chức và hoạt động của đại học vùng và các cơ sở giáo dục đại học thành viên;

Căn cứ Quyết định số 6950/QĐ-ĐHĐN ngày 01 tháng 12 năm 2014 của Giám đốc Đại học Đà Nẵng về việc ban hành Quy định nhiệm vụ, quyền hạn của Đại học Đà Nẵng, các cơ sở giáo dục đại học thành viên và các đơn vị trực thuộc;

Căn cứ Quyết định số 39/ĐHBK-ĐT ngày 12 tháng 01 năm 2016 của Hiệu trưởng trường Đại học Bách khoa về việc ban hành Quy định tổ chức và triển khai thực hiện Thực tập tốt nghiệp và làm Đồ án tốt nghiệp cho sinh viên Chương trình tiên tiến;

Theo đề nghị của ông Giám đốc Trung tâm xuất sắc và Trưởng phòng Đào tạo,

QUYẾT ĐỊNH:

Điều 1. Triển khai Capstone project cho sinh viên năm cuối của của Chương trình Tiên tiến ngành Điện tử - Viễn thông và ngành Hệ thống Nhúng. Tên đề tài, nhóm sinh viên thực hiện, danh sách Hội đồng hướng dẫn được đính kèm theo quyết định.

Điều 2. Hội đồng hướng dẫn tổ chức cho sinh viên thực hiện Capstone project theo quy định hiện hành. Sau khi hoàn thành nhiệm vụ, Hội đồng tự giải thể.

Điều 3. Các Ông (Bà) Trưởng phòng Tổ chức - Hành chính, Trưởng phòng Đào tạo, Trưởng phòng Kế hoạch - Tài chính, Trưởng phòng Công tác sinh viên, Phụ trách Chương trình tiên tiến và các cán bộ, sinh viên có tên trong danh sách ở Điều 1 chịu trách nhiệm thi hành Quyết định này./.

Nơi nhận:
- *Như Điều 3*
- *Phòng TC-HC, KH-TC, CTSV, CTTT*
- *Lưu Phòng ĐT*

KT **HIỆU TRƯỞNG**
PHÓ HIỆU TRƯỞNG

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC BÁCH KHOA
ĐẠI HỌC ĐÀ NẴNG

PGS.TS LÊ CUNG

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

DANH SÁCH THÀNH VIÊN HỘI ĐỒNG HƯỚNG DẪN
CAPSTONE PROJECT
Chương trình tiên tiến

(Ban hành kèm theo Quyết định số 96/ĐHBK-ĐT, ngày 20 tháng 01năm 2016)

Hội đồng 16

1. Tên đề tài: Simulate a self-parking car

2. Nhóm sinh viên thực hiện:

Phạm Thanh	Trâm	Lớp 11ECE
Phạm Thanh	Vinh	Lớp 11ECE
Lê Thái	Xương	Lớp 11ECE

3. Hội đồng hướng dẫn:

Stt	Họ và tên	Chức vụ/Đơn vị	Nhiệm vụ
1	TS. Nguyễn Lê Hòa	Phụ trách CTTT ES, Trường ĐHBK	Chủ tịch – Hướng dẫn
2	Ông Phạm Minh Hải	Kỹ sư, Công ty FPT Software Đà Nẵng	Ủy viên – Hướng dẫn
3	TS. Lê Quốc Huy	Giảng viên, Khoa Điện, Trường ĐHBK	Ủy viên – Phản biện

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

DANH SÁCH THÀNH VIÊN HỘI ĐỒNG HƯỚNG DẪN CAPSTONE PROJECT

Chương trình tiên tiến

(*Ban hành kèm theo Quyết định số 96/ĐHBK-ĐT, ngày 20 tháng 01năm 2016*)

Hội đồng 17

1. Tên đề tài: Simulate a self-parking car

2. Nhóm sinh viên thực hiện:

Lê Văn Vũ An		Lớp 11ES
Trần Khang Nguyên		Lớp 11ES

3. Hội đồng hướng dẫn:

Stt	Họ và tên	Chức vụ/Đơn vị	Nhiệm vụ
1	TS. Trương Thị Bích Thanh	Trưởng Bộ môn TĐH, Khoa Điện, Trường ĐHBK	Chủ tịch – Hướng dẫn
2	Ông Phạm Minh Hải	Kỹ sư, Công ty FPT Software Đà Nẵng	Ủy viên – Hướng dẫn
3	TS. Nguyễn Văn Minh Trí	Giảng viên, Khoa Điện, Trường ĐHBK	Ủy viên – Phản biện

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Số: 716./ĐHBK-ĐT

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

Đà Nẵng, ngày 24 tháng 5 năm 2016

QUYẾT ĐỊNH

V/v thành lập Hội đồng bảo vệ tốt nghiệp Capstone project cho sinh viên Chương trình tiên tiến

HIỆU TRƯỞNG TRƯỜNG ĐẠI HỌC BÁCH KHOA

Căn cứ Nghị định số 32/CP ngày 4/4/1994 của Chính phủ về việc thành lập Đại học Đà Nẵng;

Căn cứ Thông tư số 08/2014/TT-BGDĐT ngày 20 tháng 03 năm 2014 của Bộ Trưởng Bộ Giáo dục và Đào tạo về việc ban hành Quy chế tổ chức và hoạt động của đại học vùng và các cơ sở giáo dục đại học thành viên;

Căn cứ Quyết định số 6950/QĐ-ĐHĐN ngày 01 tháng 12 năm 2014 của Giám đốc Đại học Đà Nẵng về việc ban hành Quy định nhiệm vụ, quyền hạn của Đại học Đà Nẵng, các cơ sở giáo dục đại học thành viên và các đơn vị trực thuộc;

Căn cứ Quyết định số 39/ĐHBK-ĐT ngày 12 tháng 01 năm 2016 của Hiệu trưởng trường Đại học Bách khoa về việc ban hành Quy định tổ chức và triển khai thực hiện Thực tập tốt nghiệp và làm Đồ án tốt nghiệp cho sinh viên Chương trình tiên tiến;

Căn cứ Quyết định số 96/ĐHBK-ĐT ngày 20 tháng 01 năm 2016 của Hiệu trưởng trường Đại học Bách khoa về việc giao đề tài và thành lập Hội đồng BẢO VỆ TỐT NGHIỆP Capstone project cho sinh viên Chương trình Tiên tiến;

Theo đề nghị của ông Giám đốc Trung tâm xuất sắc và Trưởng phòng Đào tạo,

QUYẾT ĐỊNH:

Điều 1. Thành lập các Hội đồng bảo vệ tốt nghiệp Capstone project cho các sinh viên khóa 2010 và 2011 chính quy Chương trình Tiên tiến ngành Điện tử - Viễn thông và ngành Hệ thống Nhúng. Tên đề tài, nhóm sinh viên thực hiện, danh sách Hội đồng bảo vệ tốt nghiệp được đính kèm theo quyết định.

Điều 2. Các Hội đồng có trách nhiệm hoàn thành tốt nhiệm vụ theo đúng quy chế hiện hành của Bộ Giáo dục và Đào tạo và quy định của Trường. Sau khi hoàn thành nhiệm vụ, Hội đồng tự giải thể.

Điều 3. Các Ông (Bà) Trưởng phòng Tổ chức - Hành chính, Trưởng phòng Đào tạo, Trưởng phòng Kế hoạch - Tài chính, Phụ trách Chương trình tiên tiến và các cán bộ, sinh viên có tên trong danh sách ở Điều 1 chịu trách nhiệm thi hành Quyết định này./.

Nơi nhận:
- *Như Điều 3*
- *Phòng TC-HC, KH-TC, CTSV, CTTT*
- *Lưu Phòng ĐT*

HIỆU TRƯỞNG
PHÓ HIỆU TRƯỞNG

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC BÁCH KHOA
ĐẠI HỌC ĐÀ NẴNG

PGS.TS LÊ CUNG

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

DANH SÁCH THÀNH VIÊN HỘI ĐỒNG BẢO VỆ TỐT NGHIỆP
CAPSTONE PROJECT
Chương trình tiên tiến

(*Ban hành kèm theo Quyết định số 716/ĐHBK-ĐT, ngày 24 tháng 5 năm 2016*)

Hội đồng 16

1. Tên đề tài: Simulate a self-parking car

2. Nhóm sinh viên thực hiện:

Phạm Thanh	Trâm	Lớp 11ECE
Phạm Thanh	Vinh	Lớp 11ECE
Lê Thái	Xương	Lớp 11ECE

3. Hội đồng bảo vệ tốt nghiệp (M4):

Stt	Họ và tên	Chức vụ/Đơn vị	Nhiệm vụ
1	TS. Nguyễn Hoàng Mai	Giảng viên, Khoa Điện, Trường ĐHBK	Chủ tịch
2	TS. Giáp Quang Huy	Giảng viên, Khoa Điện, Trường ĐHBK	Thư ký
3	TS. Nguyễn Lê Hòa	Phụ trách CTTT ES, Trường ĐHBK	Ủy viên – Hướng dẫn
4	Ông Phạm Minh Hải	Kỹ sư, Công ty FPT Software Đà Nẵng	Ủy viên – Hướng dẫn
5	TS. Lê Quốc Huy	Giảng viên, Khoa Điện, Trường ĐHBK	Ủy viên – Phản biện

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

DANH SÁCH THÀNH VIÊN HỘI ĐỒNG BẢO VỆ TỐT NGHIỆP
CAPSTONE PROJECT
Chương trình tiên tiến

(Ban hành kèm theo Quyết định số 716/ĐHBK-ĐT, ngày 24 tháng 5 năm 2016)

Hội đồng 17

1. Tên đề tài: Simulate a self-parking car

2. Nhóm sinh viên thực hiện:

Lê Văn Vũ	An	Lớp 11ES
Trần Khang	Nguyên	Lớp 11ES

3. Hội đồng bảo vệ tốt nghiệp (M4):

Stt	Họ và tên	Chức vụ/Đơn vị	Nhiệm vụ
1	TS. Nguyễn Hoàng Mai	Giảng viên, Khoa Điện, Trường ĐHBK	Chủ tịch
2	TS. Giáp Quang Huy	Giảng viên, Khoa Điện, Trường ĐHBK	Thư ký
3	TS. Trương Thị Bích Thanh	Trưởng Bộ môn TĐH, Khoa Điện, Trường ĐHBK	Ủy viên – Hướng dẫn
4	Ông Phạm Minh Hải	Kỹ sư, Công ty FPT Software Đà Nẵng	Ủy viên – Hướng dẫn
5	TS. Nguyễn Văn Minh Trí	Giảng viên, Khoa Điện, Trường ĐHBK	Ủy viên – Phản biện

ABSTRACT

Everyone loves driving cars, there is no doubt about that. However, this does not mean that they also like to do the parking. In particular, parallel parking is an ordeal for many drivers. With parking space limited in big cities, squeezing their cars into tiny spaces is not simple. It is seldom an easy task, and it can lead to traffic tie-ups, frazzled nerves and bent fenders. Beside parallel parking, perpendicular parking (garage parking) is apparently drivers' nightmare also. Many of them have encountered problematic situations, which are taking them hours to fit their cars into the garages or even causing property damage such as crashed sidelights, bent wing mirrors. Fortunately, technology has an answer – cars that can park themselves. Imagine fiding the perfect parking spot, but instead of struggling to maneuver the car back and forth, the user simply press a button, sit back and relax as the car automatically does the parking process. Inspired by this idea, our thesis proposes a design of a self-parking system applied to the car so that it can handle both parallel and garage parkings. To obtain these functionalities, we apply two parking algorithms (parallel and garage) in addition to a combination of multiple ultrasonic sensors, microcontrollers, servo motor, driver and hall effect module. About the obtained result so far, this design works properly on an $1/10^{th}$ RC (Radio controlled) car.

ACKNOWLEDGMENT

This diploma thesis was written in the year 2016 under supervision of Mr. Pham Minh Hai, FPT Software, Embedded Systems department in DaNang, Vietnam.

Filled with immense gratitude, we would like to thank our supervisor Mr. Pham Minh Hai for his patience and eager help and supervision in every aspect of the thesis. We would also like to express our deep gratitude to Mr. Phan Tuan Hiep and Mr. Ho Viet Nhu Dao for the invaluable support and guidance ever since we step foot in FPT.

We would like to thank FPT Software, Embedded Systems department in particular for providing us the friendly working environment and extraordinary equipment, as well as sharing their experience to help us overcome problematic situations efficiently.

We would also like to thank Center of Excellence for the enthusiasm of helping us to complete this thesis.

Danang, June 2016

Le Van Vu An

Le Thai Xuong

Tran Khang Nguyen

Pham Thanh Tram

Pham Thanh Vinh

Contents

List of Figures

List of Tables

Abbreviations

MCU	Microcontroller unit
PWM	Pulse Width Modulation
CAN	Controller Area Network
RTR	Remote Tranmission Request
DLC	Data Length Code
GUI	Graphical User Interface
LiPo	Lithium-ion polymer battery

INTRODUCTION

1. Motivation

These days, self-parking maneuver is not a new concept in automation field. In fact, it exists for almost a decade and has been developing constantly. However, most of the current self-parking systems require two indispensable factors: the car-like robot and parking-aid garage (parking slot). In this way, both two factors must operate simultaneously to ensure proper self-parking process. The thing is if the parking-aid garage is under maintenace or currently out of order, the car-like robot will be useless. Therefore, we proposes a a new concept- Self parking car, which is a car that can automatically park to garage (or parking spot) without human effort or parking-aid from parking lot.

Self-parking car, as its name implies, is capable of handling parking maneuver on its own with no human effort and parking guidance from parking lot. In order to achieve this, we must take full advantage of multiple range detecting sensors (ultrasonic sensors) for distance readings from the car to wall of garage; apply hall effect to record precise distances the car travels; and apply self-parking algorithm for two possible situation: parallel parking and garage parking.

2. Contribution of the thesis

In order to solve the mentioned problems, we chose the topic of “Simulate a self-parking car” as our thesis topic. Our implementation ensures that the car is capable of handling the parking maneuver on its own, without any human effort or external guidance-system installed on a parking lot, for example. To begin with, we focus on detailed theory of specific concepts and equipment for software and hardware design purposes, which is very essential for our system. Then we propose two algorithms for parallel and garage parking processes. Last but not least, the experimental results and evaluation will be indicated at the end of our thesis.

3. Organization of the thesis

Our thesis has two main parts, which are divided into four chapters.

- Part 1- Theory: Chapter 1 and chapter 2
- Part 2- Design: Chapter 3 and chapter 4

Chapter 1- Detailed theory (Review of Literature and Software design purpose)

- Review of Literature
- Ackermann steering
- Car Kinematic
- Fuzzy logic
- Pulse Width Modulation (PWM)

Chapter 2- Detailed theory (Hardware design purpose)

- Ultrasonic sensor
- CAN BUS
- Hall Effect
- Stepper motor
- Bluetooth

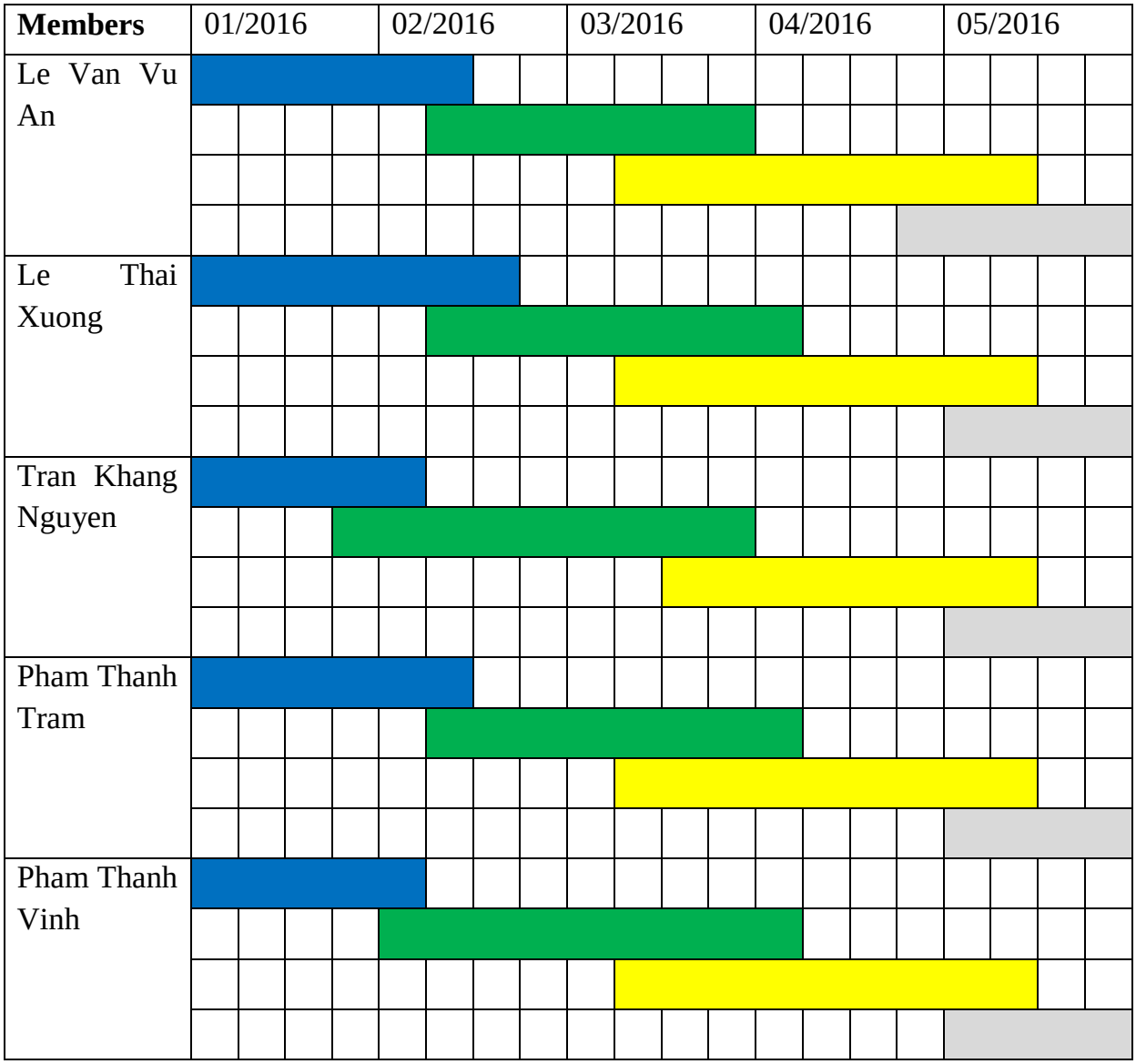
Chapter 3- Proposed algorithms/systems

- System Overview
- Wall alignment fuzzy logic system
- Finite State Machine for Self-Parking Car: Parallel Parking and Garage Parking

Chapter 4- Experimental Results and Evaluation

- Hardware design
- Software design
- Simulation and Experimental Result
- Evaluation

4. Milestone of the Project



	Study theory, equipment functionality
	Program/simulate and test
	Evaluate obtained results
	Thesis writing

5. Work distribution

1. Le Van Vu An:

- Study and apply CAN BUS to system
- Design algorithm for garage parking
- Design an Android application GUI

2. Le Thai Xuong:

- Study kinematics on wheeled systems
- Design task scheduler
- Design algorithm for parallel parking

3. Pham Thanh Tram

- Study and apply PWM, Hall effect, stepper motor to system
- Implement hardware assembly

4. Tran Khang Nguyen and Pham Thanh Vinh:

- Study ultrasonic sensor
- Research garage and parking lot standards
- Design car box and sensor stabilizer

PART 1 - THEORY

CHAPTER 1- DETAILED THEORY (REVIEW OF LITERATURE AND SOFTWARE DESIGN PURPOSE)

1.0. Review of literature

Self-parking car, as its name implies, is capable of handling parking maneuver on its own with no human effort and parking guidance from parking lot. In order to achieve this, we must take full advantage of multiple range detecting sensors (ultrasonic sensors) for distance readings from the car to wall of garage; apply hall effect to record precise distances the car travels; and apply self-parking algorithm for two possible situation: parallel parking and garage parking.

Related Works

Automatic car parking is available in many vehicles till now, but all are not completely autonomous. Most of them are semi autonomous parking system which assist driver for car parking. Brake and accelerator is to be controlled by driver.

Following are some related work about car parking.

- In 2003, Toyota began to sell their Japanese Prius hybrid vehicle with an automatic parallel parking capability offered as an option named Intelligent Parking Assist.
- In 2006, Lexus added a self-parking system to the redesigned Lexus LS sedan. It parallel parks as well as angle parks.
- In 2010, BMW introduced a system called "parking assistant" on the redesigned 5-series. It does parallel parking.
- Up to 2012, automatic parking systems were being developed by several automobile manufacturers. Ford and Lincoln offered active park assist on Ford Focus, Fusion, Escape, Explorer and Flex and Lincoln MKS and MKT. Toyota and Lexus had advanced parking assistant on Toyota Prius V Five and Lexus LS460 and LS460 L. BMW all-new sixth-generation 3 Series used a system called parking assistant. Audi A6, Mercedes-Benz also offered parktronic on their C-Class, CLSClass Coupe, M-Class SUV, E-Class, S-Class, GL350, GL450 SUV (standard on GL550) and R-Class in different prices. Jeep introduced automatic parallel and perpendicular parking on its 2014 Cherokee model. Chrysler introduced an all new

2015 200 sedan, offering automatic parking as part of a Safetytec package. The system steers the car into parallel and perpendicular parking spaces. But all above system are having parking assist feature which take care of steering but brake and accelerator has to be handled by driver. Which make them semi-autonomous parking or parking assist systems. This paper proposes method of fully automatic car parking in which driver can get out of the car and control parking through Smartphone or wireless communication.

For the next sections, only the detailed theory of technical solutions will be mentioned here. The reasons why we use these solutions will be explained in chapter 3.

1.1. Ackermann steering

To begin with, we consider Ackermann steering, which is very crucial for car kinematics. Ackermann steering geometry is a geometric arrangement of linkages in the steering of a car or other vehicle designed to solve the problem of wheels on the inside and outside of a turn needing to trace out circles of different radii.

The intention of Ackermann geometry is to avoid the need for tyres to slip sideways when following the path around a curve. The geometrical solution to this is for all wheels to have their axles arranged as radii of circles with a common centre point. As the rear wheels are fixed, this centre point must be on a line extended from the rear axle. Intersecting the axes of the front wheels on this line as well requires that the inside front wheel is turned, when steering, through a greater angle than the outside wheel.

Rather than the preceding "turntable" steering, where both front wheels turned around a common pivot, each wheel gained its own pivot, close to its own hub. While more complex, this arrangement enhances controllability by avoiding large inputs from road surface variations being applied to the end of a long lever arm, as well as greatly reducing the fore-and-aft travel of the steered wheels. A linkage between these hubs pivots the two wheels together, and by careful arrangement of the linkage dimensions the Ackermann geometry could be approximated. This was achieved by making the linkage not a simple parallelogram, but by making the length of the track rod (the moving link between the hubs) shorter than that of the axle, so that the steering arms of the hubs appeared to "toe out". As the steering moved, the wheels turned according

to Ackermann, with the inner wheel turning further. If the track rod is placed ahead of the axle, it should instead be longer in comparison, thus preserving this same "toe out".

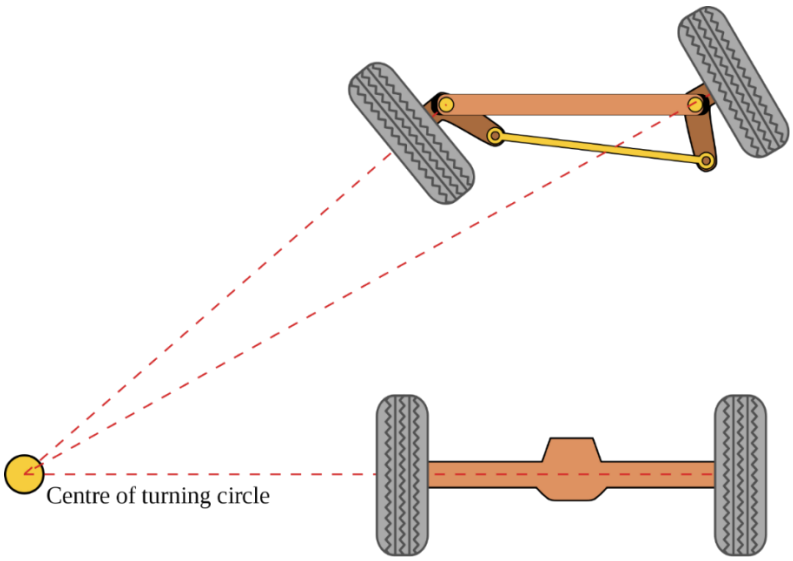


Figure 1.1a. Ackermann geometry illustration

1.2. Car Kinematics

In this project, we consider a Car-Like Robot (RC Car) as a model of a real car. The cars are known to be non-holonomic because there are differential constraints that cannot be completely integrated. To acknowledge the moving behaviors of a car, we have to consider the kinematic of a wheeled system[1].

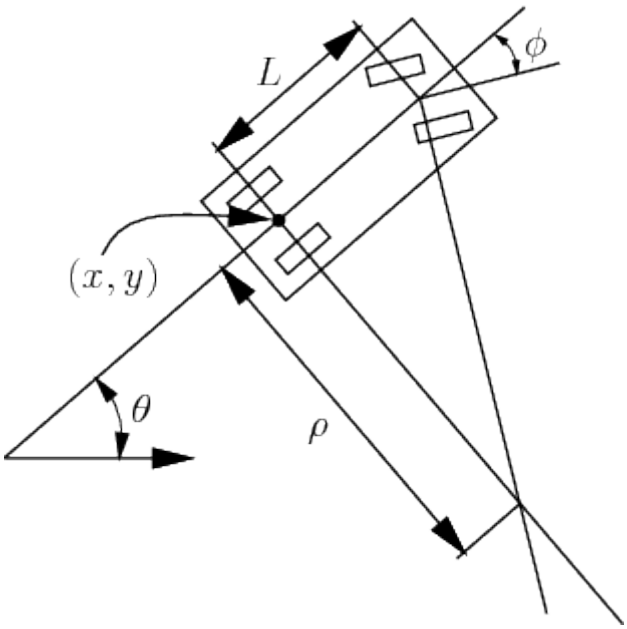


Figure 1.2a. The simple car has three degrees of freedom, but the velocity space at any configuration is only two-dimensional.

We can assume that the car as a rigid body moving in a plane. Therefore, its C-space is $\mathrm{C} = R^2 \times S^1$. The car configuration is: $q = (x, y, \theta) \in R \times S^1$, (x, y) is the middle point of the rear wheel. The body frame of the car places the origin at the center of rear axle, and the x-axis points along the main axis of the car. Let s denote the (signed) speed of the car and let $\emptyset$ as the steering angle (negative value when turning right and positive value when turning left). The distance between the front and rear axles is represented as L. If the steering angle is fixed at $\emptyset$, the car travels in a circular motion, in which the radius of the circle is ρ. Note that ρ can be determined from the intersection of the two axes shown in Figure 1.

Using the current notation, the task is to represent the motion of the car as a set of equations of the form:

$$\dot{x} = f_1(x, y, \theta, s, \emptyset)$$
$$\dot{y} = f_2(x, y, \theta, s, \emptyset)$$
$$\dot{\theta} = f_3(x, y, \theta, s, \emptyset)$$

In a small time interval Δt, the car must move approximately in the direction that the rear wheels are pointing. In the limit as Δt tends to zero, this implies that $\frac{dy}{dx} = tan\theta$. Since $\frac{dy}{dx} = \frac{\dot{y}}{\dot{x}}$ and $tan\theta = \frac{sin\theta}{cos\theta}$ this condition can be written as a Pfaffian constraint :

$$-\dot{x}sin\theta + \dot{y}cos\theta = 0$$

The constraint is satisfied if $\dot{x} = cos\theta$ and $\dot{y} = sin\theta$. Furthermore, any scalar multiple of this solution is also a solution; the scaling factor corresponds directly to the speed s of the car. Thus, the first two scalar components of the configuration transition equation are $\dot{x} = s \times cos\theta$ and $\dot{y} = s \times sin\theta$.

The next task is to derive the equation for $\dot{\theta}$. Let ω denote the distance traveled by the car (the integral of speed). As shown in Figure 1, ρ epresents the radius of a circle that is traversed by the center of the rear axle, if the steering angle is fixed. Note that $d\omega = \rho d\theta$. From trigonometry, $= \frac{L}{tan\emptyset}$, which implies

$$d\theta = \frac{tan\theta}{L} d\omega$$

Dividing both sides by dt and using the fact that $\dot{\omega} = s$ yields

$$\dot{\theta} = \frac{s}{L} tan\emptyset$$

So far, the motion of the car has been modeled, but no action variables have been specified. Suppose that the speed s and steering angle $\emptyset$ are directly specified by the action variables u_s and $u_\emptyset$, respectively. The convention of using a u variable with the old variable name appearing as a subscript will be followed. This makes it easy to identify the actions in a configuration transition equation. A two-dimensional action vector, $= (u_s,\ u_\emptyset)$, is obtained. The configuration transition equation for the simple car is:

$$\dot{x} = u_s cos\theta$$
$$\dot{y} = u_s sin\theta$$
$$\dot{\theta} = \frac{u_s}{L} tanu_\emptyset$$

Now we have successfully investigate movement of a car-like robot, now we will simulate that movement on Simulink based on above equation.

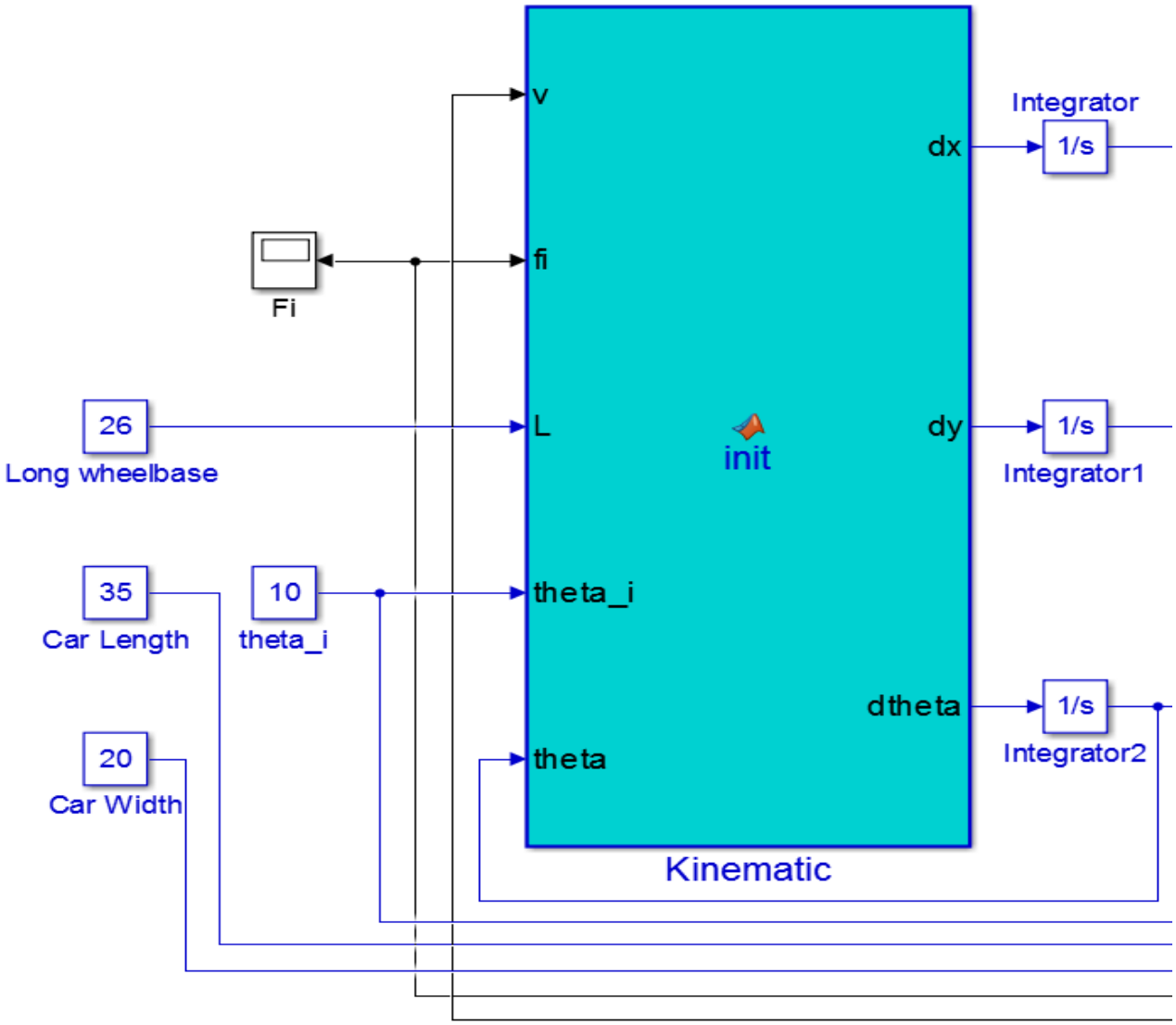


Figure 1.2b. Car Kinematic Block on Simulink

Our Car Kinematic Block gets car dimensions and velocity and initial heading angle of a car. The output of this block is the rate of change of x, y, θ then simulink will intergrate them by time to get the value of x, y, θ.

We run a simple simulation that a car go straight forward with 20 degree in steering angle and 20 cm/s in speed. We have a figure below.

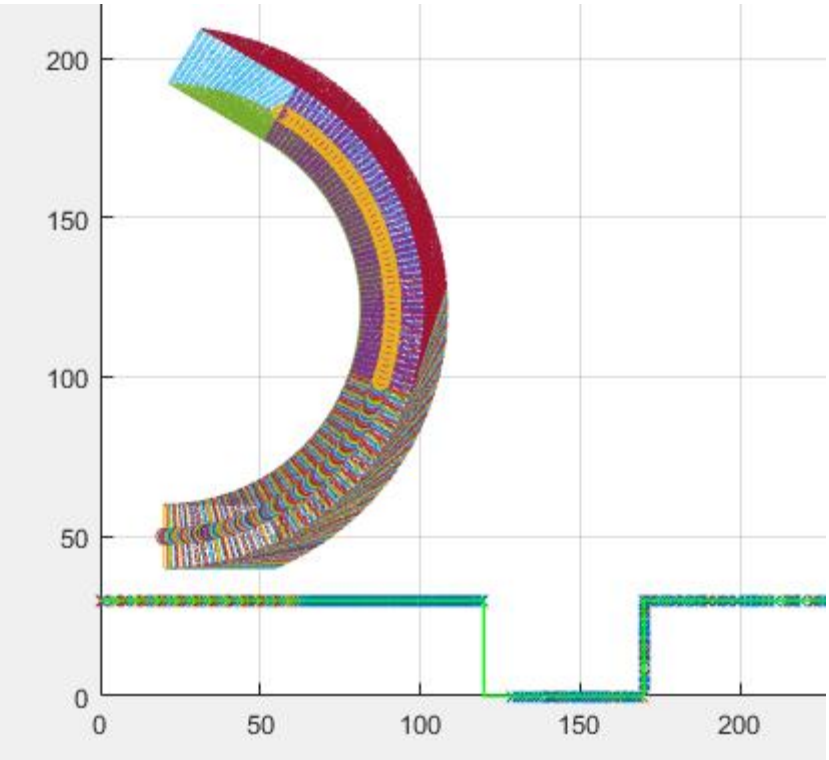


Figure 1.2c. Car Movement Simulation

Based on the car's trajectory on the above figure which resembles the real life car's trajectory. We can use that model to simulate, testing and apply our algorithm.

1.3. Fuzzy Logic

Fuzzy logic[2] is against Aristotelian logic. According to Aristotelian logic, for a given proposition or state we only have two logical values: true-false, black-white, 1-0. However, in real life, things are not necessary be either black or white. In fact, most of the times are grey. Thus, in many practical situations, it is convenient to consider intermediate logical values.

Let us consider the statement "you are smart" in terms of examination results. Is the statement true if you score 90 marks? Is it false if you score 60? Everyone is smart to a certain degree and "not smart" to some degree. Let us use "s" to represent smart and "n" to represent not smart. If you are totally smart, then of course s = 1 and n = 0. Usually, everyone has a mixture of both and $s<1$, but $s + n = 1$.

In the other extreme condition, n = 1 and s = 0, so that you score 0 marks (you handed in empty script). In the case where you score 90 marks, we may write s = 0.9 and n = 0.1.

Fuzzy logic emerged as a consequence of the 1965 proposal of fuzzy set theory by Lotfi Zadeh. Though fuzzy logic has been applied to many fields, from control theory to artificial intelligence, it still remains controversial among most statisticians, who prefer Bayesian logic, and some control engineers, who prefer traditional two-valued logic.

Fuzzy Sets

Fuzzy logic starts with the concept of a fuzzy set. Fuzzy sets are sets without crisp and clearly defined boundary. It contains elements with only a partial degree of membership.

Fuzzy set is an extension of the classical set. In classical set theory, the membership of elements in a set is assessed in binary terms according to a bivalent condition — it wholly includes or excludes any given element. By contrast, fuzzy set theory permits the gradual assessment of the membership of elements in a set; this is described with the aid of a membership function valued in the real unit interval [0, 1]. The fuzzy set theory can be used in a wide range of domains in which information is incomplete or imprecise, such as bioinformatics.

The following statement lays the foundation of fuzzy logic:

In the world of fuzzy logic, the truth of any statement becomes only a matter of degree.

Membership functions

A membership function (MF) is a curve that defines the degree of membership (lies between 0 and 1) that a value in the input universe of discourse (input space) is mapped to. The simplest membership functions are formed using straight lines. Of these, the most commonly used MFs include the triangular MF and the trapezoidal MF, which is really a truncated triangular MF. If X is the universe of discourse and its elements are denoted by X, then $\mu_A(x)$ is defined as the membership value or degree of membership of x in the MF of the linguistic variable A. The linguistic variables are usually labeled LN (Large Negative), MN (medium Negative), SN (Small Negative), ZE (Zero), SP (Small Positive), MP (Medium Positive) and LP (Large Positive).

Fuzzification

A fuzzy controller will receive crisp inputs on its input or communications port and initially fuzzify them. Each system input is divided into sets of membership functions which are overlapping. The predefined membership functions cover the whole range of values for an input and will then define a degree of truth for every point in the range of values.

The input is always a crisp numerical value limited to a range of values of the input variable and the output is a fuzzy degree of membership (always in the interval between 0 and 1) corresponding to one or more membership functions.

A diagram of an example of fuzzification is shown in figure 2.3a.

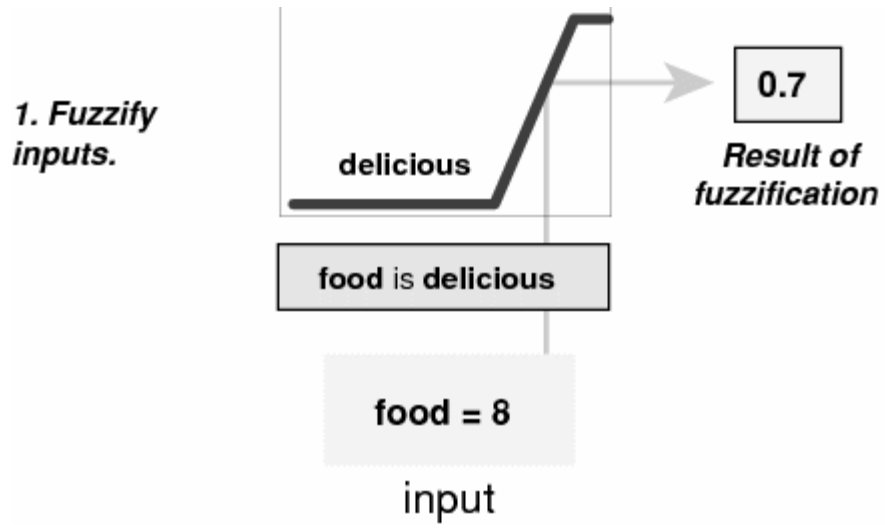


Figure 1.3. Example of fuzzification

Fuzzy Inferencing

Fuzzy rules combine 2 or more input fuzzy sets, called the anecedents sets, and associate an output with them, or consequent set. Due to the partial matching attribute of fuzzy control rules and the fact that the preconditions of the rules do overlap, usually more than one fuzzy rule will fire off at one time. The methodology in deciding what control action to take as a result of firing several rules simultaneously can be referred to as the process of fuzzy inferencing.

Defuzzification

The final output from the fuzzy controller has to be defuzzified in order to obtain a crisp value to be input back into the controlled system. This necessary operation produces a non-fuzzy (crisp) control signal that best represents the aggregate membership function obtained from the fuzzy inferencing process.

The 2 most common defuzzification methods are the Centre – of – Area method and the Centre-of-Sum method.

The COA method is the most commonly used technique and is very accurate. It can be expressed as

$$x^* = \frac{\int \mu_i(x)\, x\, dx}{\int \mu_i(x)\, dx}$$

Where x* is the defuzzified output, $\mu_i(x)$ is the aggregated membership function and x is the output variable. The only disadvantage of this method is that it is computationally difficult for complex membership functions.

The COS method is a simplified version of the former. Under the COS method, the defuzzified output U is given by:

$$U = \frac{\sum_{i=1}^{m} \mu_{z,i}\, \overline{z_i}}{\sum_{i=1}^{m} \mu_{z,i}}$$

Where:

m = number of overlapped rules that are fired simultaneously

$\mu_{z,I}$ = membership value of the output for the i^{th} fired rule

$\overline{z_i}$ = specific crisp (numerical) value assigned to each linguistic variable

Advantages

Fuzzy logic offers many advantages in systems design. The operators used in fuzzy logic are as simple as traditional Boolean logic operators. System operators can therefore use it to further extend their knowledge of operation into the domain of membership functions and fuzzy logic rules, which linguistically is modelled on our own language.

For traditional system developers with complex systems, the complexity can be simplified using fuzzy logic. Complex applications with multiple inputs and outputs can be modelled and implemented using fuzzy logic.

Another advantage is that fuzzy logic reduces the processing requirement of a system and thus reduces the cost of the embedded control hardware. In many cases, complex mathematical modelling can be replaced with membership functions and a set of fuzzy rules to control a system. Minimizing these mathematical constraints can reduce code size and hence allow the system to run faster.

1.4. Pulse Width Modulation (PWM)

An analog signal has a continuously varying value, with infinite resolution in both time and magnitude. A nine-volt battery is an example of an analog device, in that its output voltage is not precisely 9V, changes over time, and can take any real-numbered value. Similarly, the amount of current drawn from a battery is not limited to a finite set of possible values. Analog signals are distinguishable from digital signals because the latter always take values only from a finite set of predetermined possibilities, such as the set {0V, 5V}.

Analog voltages and currents can be used to control things directly, like the volume of a car radio. In a simple analog radio, a knob is connected to a variable resistor. As you turn the knob, the resistance goes up or down. As that happens, the current flowing through the resistor increases or decreases. This changes the amount of current driving the speakers, thus increasing or decreasing the volume. An analog circuit is one, like the radio, whose output is linearly proportional to its input.

As intuitive and simple as analog control may seem, it is not always economically attractive or otherwise practical. For one thing, analog circuits tend to drift over time and can, therefore, be very difficult to tune. Precision analog circuits, which solve that problem, can be very large, heavy (just think of older home stereo equipment), and expensive. Analog circuits can also get very hot; the power dissipated is proportional to the voltage across the active elements multiplied by the current through them. Analog circuitry can also be sensitive to noise. Because of its infinite resolution, any perturbation or noise on an analog signal necessarily changes the current value.

By controlling analog circuits digitally, system costs and power consumption can be drastically reduced. What's more, many microcontrollers and DSPs already include on-chip PWM controllers, making implementation easy.

In a nutshell, PWM is a way of digitally encoding analog signal levels. Through the use of high-resolution counters, the duty cycle of a square wave is modulated to encode a specific analog signal level. The PWM signal is still digital because, at any given instant of time, the full DC supply is either fully on or fully off. The voltage or current source is supplied to the analog load by means of a repeating series of on and off pulses. The on-time is the time during which the DC supply is applied to the load, and the off-time is the period during which that supply is switched off. Given a sufficient bandwidth, any analog value can be encoded with PWM.

Three different PWM signals show below. Figure 1.4a shows a PWM output at a 10% duty cycle. That is, the signal is on for 10% of the period and off the other 90%. Figures 1.4b and 1.4c show PWM outputs at 50% and 90% duty cycles, respectively. These three PWM outputs encode three different analog signal values, at 10%, 50%, and 90% of the full strength. If, for example, the supply is 9V and the duty cycle is 10%, a 0.9V analog signal results.

Figure 1.4a. PWM outputs at 10% duty cycles

Figure 1.4b. PWM outputs at 50% duty cycles

Figure 1.4c. PWM outputs at 90% duty cycles

Figure 1.4d shows a simple circuit that could be driven using PWM. In the figure, a 9V battery powers an incandescent lightbulb. If we closed the switch connecting the

battery and lamp for 50ms, the bulb would receive 9V during that interval. If we then opened the switch for the next 50ms, the bulb would receive 0V. If we repeat this cycle 10 times a second, the bulb will be lit as though it were connected to a 4.5V battery (50% of 9V). We say that the duty cycle is 50% and the modulating frequency is 10Hz.

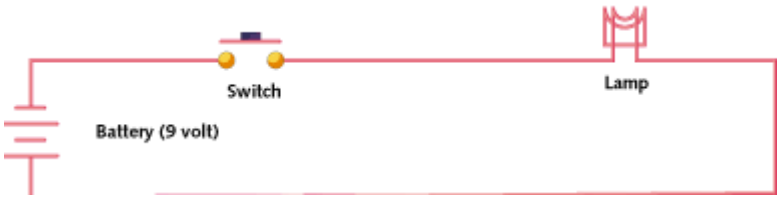


Figure 1.4d. Simple circuit driven by PWM

Most loads, inductive and capacitive alike, require a much higher modulating frequency than 10Hz. Imagine that our lamp was switched on for five seconds, then off for five seconds, then on again. The duty cycle would still be 50%, but the bulb would appear brightly lit for the first five seconds and off for the next. In order for the bulb to see a voltage of 4.5 volts, the cycle period must be short relative to the load's response time to a change in the switch state. To achieve the desired effect of a dimmer (but always lit) lamp, it is necessary to increase the modulating frequency. The same is true in other applications of PWM. Common modulating frequencies range from 1kHz to 200kHz.

Timer and PWM modes

Counter/timer hardware is a crucial component of most embedded systems. In some cases, a timer measures elapsed time (counting processor clock ticks). In others, we want to count or time external events. The names counter and timer can be used interchangeably when talking about the hardware. The difference in terminology has more to do with how the hardware is used in a given application.

The ATmega328P has three timers known as Timer 0, Timer 1, and Timer 2. Each timer has two output compare registers that control the PWM width for the timer's two outputs: when the timer reaches the compare register value, the corresponding output is toggled. The two outputs for each timer will normally have the same frequency, but can have different duty cycles (depending on the respective output compare register).

Each of the timers has a prescaler that generates the timer clock by dividing the system clock by a prescale factor such as 1, 8, 64, 256, or 1024. The Arduino has a system clock of 16MHz and the timer clock frequency will be the system clock frequency divided by the prescale factor. Note that Timer 2 has a different set of prescale values from the other timers.

The timers are complicated by several different modes. The main PWM modes are "Fast PWM" and "Phase-correct PWM", which will be described below. The timer can either run from 0 to 255, or from 0 to a fixed value. (The 16-bit Timer 1 has additional modes to supports timer values up to 16 bits.) Each output can also be inverted.

The timers can also generate interrupts on overflow and/or match against either output compare register, but that's beyond the scope of this article. Timer Registers Several registers are used to control each timer. The Timer/Counter Control Registers TCCRnA and TCCRnB hold the main control bits for the timer. (Note that TCCRnA and TCCRnB do not correspond to the outputs A and B.) These registers hold several groups of bits:

- Waveform Generation Mode bits (WGM): these control the overall mode of the timer.
  (These bits are split between TCCRnA and TCCRnB.)
- Clock Select bits (CS): these control the clock prescaler
- Compare Match Output A Mode bits (COMnA): these enable/disable/invert output A
- Compare Match Output B Mode bits (COMnB): these enable/disable/invert output B

The Output Compare Registers OCRnA and OCRnB set the levels at which outputs A and B will be affected. When the timer value matches the register value, the corresponding output will be modified as specified by the mode.

The bits are slightly different for each timer, so consult the datasheet for details. Timer 1 is a 16-bit timer and has additional modes. Timer 2 has different prescaler values.

Fast PWM

In the simplest PWM mode, the timer repeatedly counts from 0 to 255. The output turns on when the timer is at 0, and turns off when the timer matches the output compare register. The higher the value in the output compare register, the higher the duty cycle. The outputs for two particular values of OCRnA and OCRnB have the same frequency, matching the frequency of a complete timer cycle.

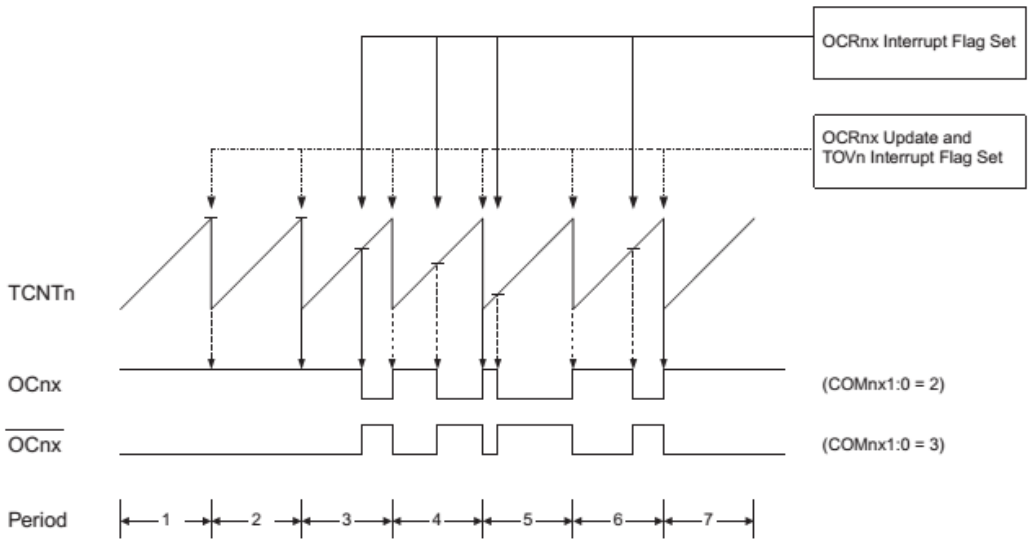


Figure 1.4e. Fast PWM illustration

Phase-Correct PWM

In this mode, the timer counts from 0 to 255 and then back down to 0. The output turns off as the timer hits the output compare register value on the way up, and turns back on as the timer hits the output compare register value on the way down. The result is a more symmetrical output. The output frequency will be approximately half of the value for fast PWM mode, because the timer runs both up and down.

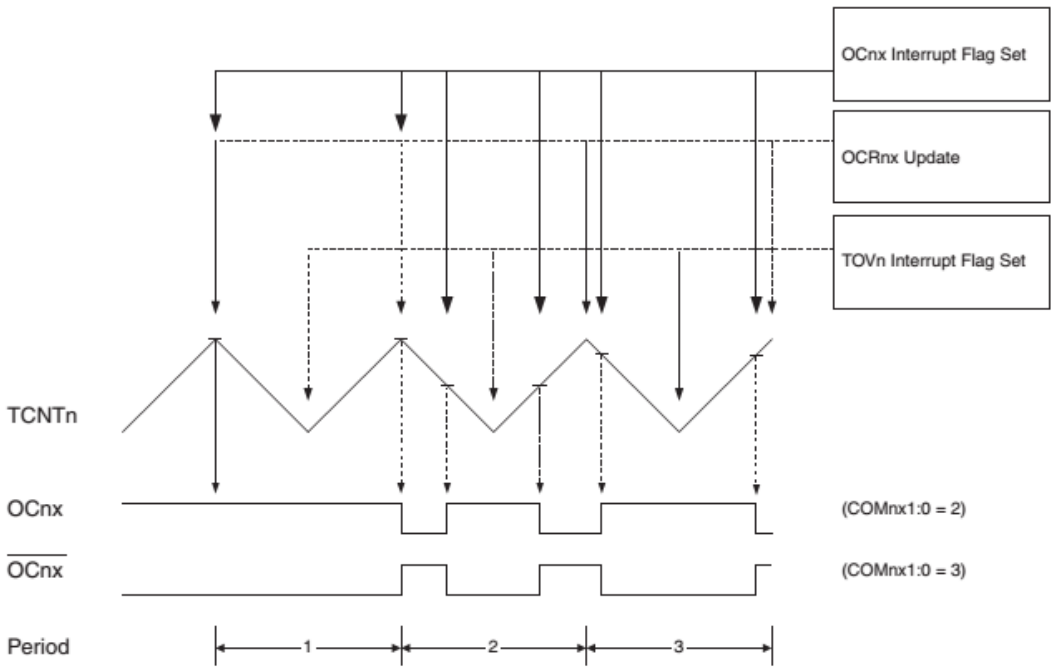


Figure 1.4f. Phase-correct PWM illustration

Varying the timer top limit

Fast PWM have an additional mode that gives control over the output frequency. In this mode, the timer counts from 0 to OCRA (the value of output compare register A), rather than from 0 to 255. This gives much more control over the output frequency than the previous modes. (For even more frequency control, use the 16-bit Timer 1.)

Note that in this mode, only output B can be used for PWM; OCRA cannot be used both as the top value and the PWM compare value. However, there is a special-case mode "Toggle OCnA on Compare Match" that will toggle output A at the end of each cycle, generating a fixed 50% duty cycle and half frequency in this case.

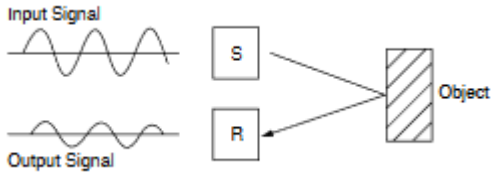
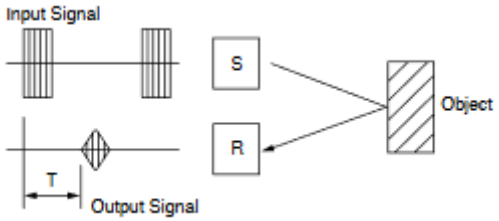
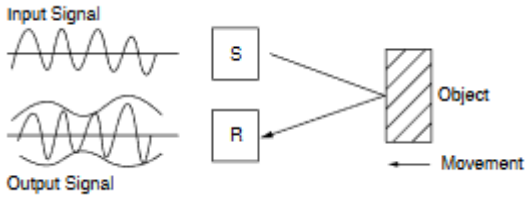
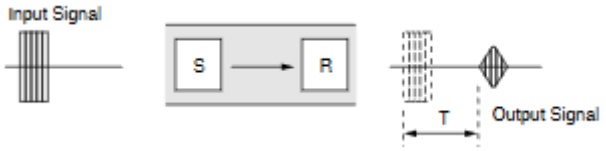
Varying the timer top limit: phase-correct PWM

Similarly, the timer can be configured in phase-correct PWM mode to reset when it reaches OCRnA.

CHAPTER 2- DETAILED THEORY (HARDWARE DESIGN PURPOSE)

2.1. Ultrasonic sensor

An ultrasonic sensor transmits ultrasonic waves into the air and detects reflected waves from an object. Ultrasonic sensors are utilized for many purposes depending on function method.

No.	Function Method	Performance Principle (S: transmitter, R: receiver)	Applications
1	**Detection of signal level of continuous wave**		Counting instruments Access switches
2	**Measurement of pulse reflection time**		Automatic doors Level gauges Automatic change-overs of traffic signals Back sonars of automobiles
3	**Utilization of Doppler effect**		Intruder alarm systems
4	**Measurement of direct propagation time**		Densitometers Flowmeters

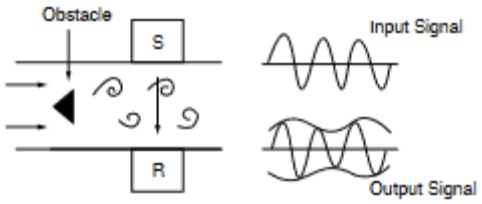
5	**Measurement of Karman vortex**		Flowmeters

Table 2.1. Ultrasonic sensor function methods and applications

The ultrasonic range sensor works by transmitting an ultrasonic pulse and measuring the time that it takes to "hear" the pulse echo. Output from the sensor is in the form of a variable-width pulse that corresponds to the distance to the target.

Ultrasonic sensors overcome many of the weaknesses of IR sensors - they provide distance measurement regardless of color and lighting of obstacles.

They also provide lower minimum distances and wider angles of detection to guarantee that obstacles are not missed because of a narrow sensor beam.

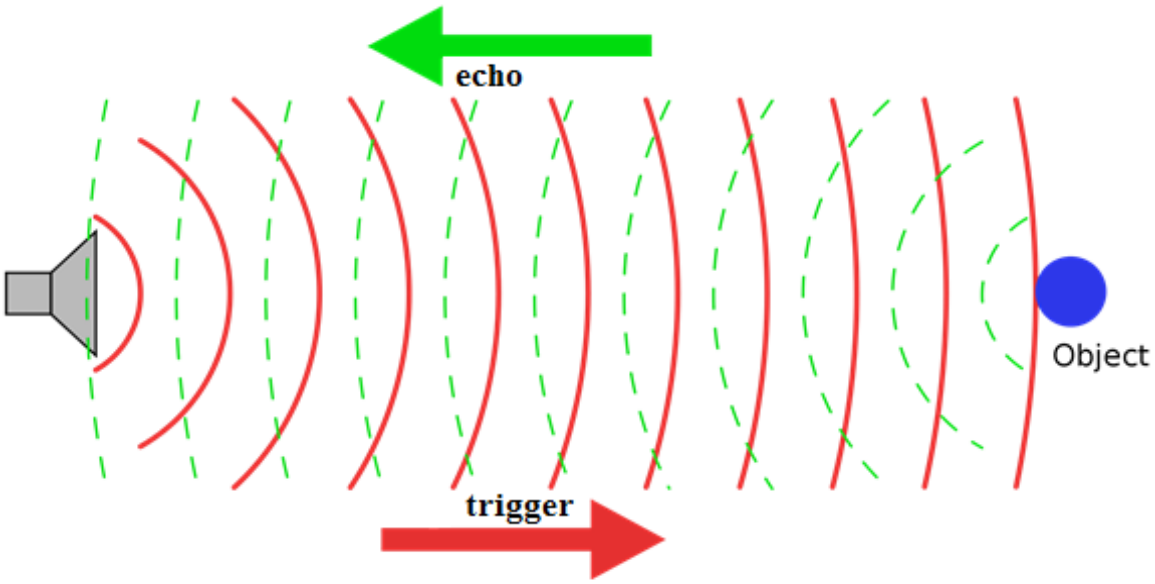


Figure 2.1. Ultrasonic sensor implementation

The ultrasonic range sensor detects objects in it's path and can be used to calculate the range to the object. It is sensitive enough to detect a 3cm diameter broom handle at a distance of over 3m.

Calculating the distance:

Every time the pulse of Trigger pin is sent out, it hit the wall and bounces back to the Echo pin. In addition, travelling time is duration of high pulse of echo pulse. As a result, the real distance from the sensor to the wall is half that of the travelled distance of Trigger pulse.

The speed of sound is 343m/s or it takes 2.9 s to travel a distance of one kilometer. Alternatively, for a distance of one centimeter, it will take 29 uS. Therefore, we can come up with a formula to calculate the distance from the sensor to the wall:

$$Distance = \frac{\text{Travelling Time}}{2 * 29} \ (in\ centemeter)$$

2.2. CAN BUS (Controller Area Network)

A **Controller Area Network** (**CAN bus**) [3] is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles, but is also used in many other contexts.

CAN is a multi-master serial bus standard for connecting Electronic Control Units (ECUs) also known as nodes. Two or more nodes are required on the CAN network to communicate. The complexity of the node can range from a simple I/O device up to an embedded computer with a CAN interface and sophisticated software. The node may also be a gateway allowing a standard computer to communicate over a USB or Ethernet port to the devices on a CAN network.

All nodes are connected to each other through a two wire bus. The wires are 120 Ω nominal twisted pair.

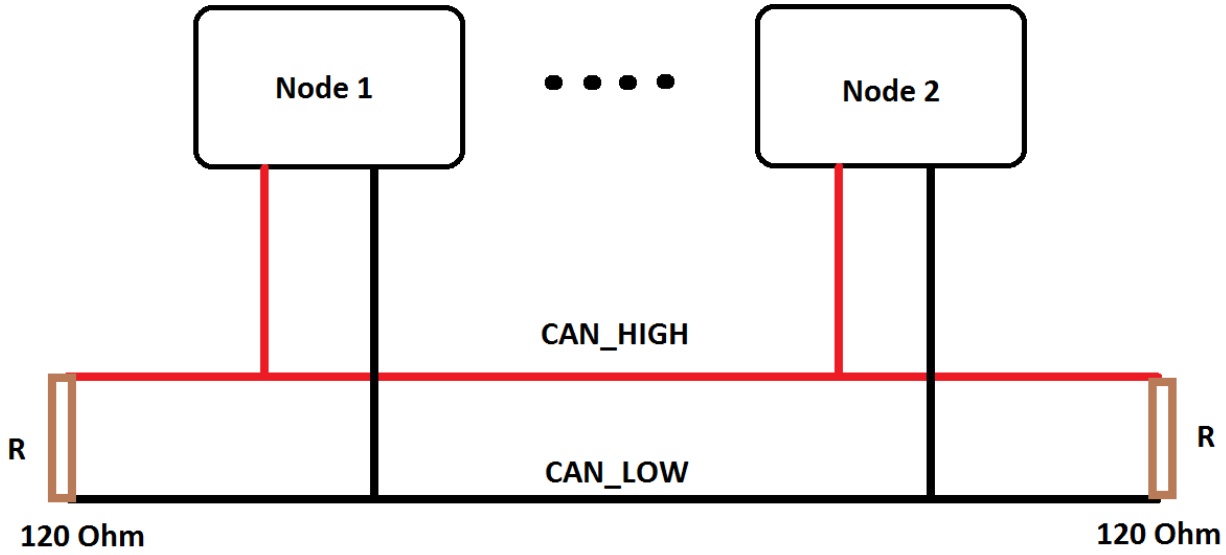


Figure 2.2a. CAN BUS Overview

To elaborate on the CAN BUS, we need to focus on three main factors which are: CAN node, message and how the data transmitted through CAN BUS.

CAN NODE:

Firstly, we will explain thoroughly the definition[4] of a CAN node. A CAN node includes two indispensable parts which are the central processing unit and the CAN module (CAN Controller + CAN Transceiver).

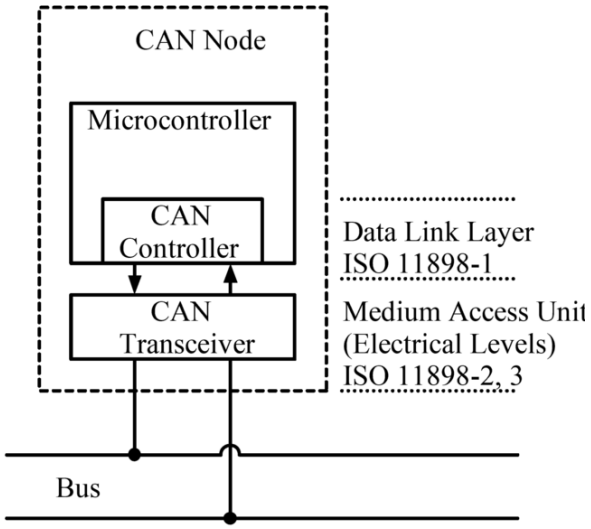


Figure 2.2b. CAN Node diagram

- Central processing unit (i.e. microcontroller): Decides what the received messages mean and what messages it wants to transmit.
- Inside the CAN module,
  - CAN Controller enables interfacing between the microcontroller and CAN Module via SPI interface. The interfacing tasks are as follows:
    - Receiving: the CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the microcontroller (usually by the CAN controller triggering an interrupt).
    - Sending: the microcontroller sends the transmit messages to a CAN controller, which transmits the bits serially onto the bus when the bus is free.
  - CAN Transceiver gives the microcontroller CAN-BUS capability.
    - Receiving: it converts the data stream from CAN bus levels to levels that the CAN controller uses. It usually has protective circuitry to protect the CAN controller.
    - Transmitting: it converts the data stream from the CAN controller to CAN bus levels.

CAN MESSAGE[5]:

Next, multiple nodes require two dedicated wires for communication. The wires are call CAN HIGH and CAN LOW terminated by two resistors of 120 Ohm each. CAN Nodes communicate with one another through messages. Each node is able to send and receive message, but not simultaneously.

A Message or Frame consists primarily of:

- ID (identifier) represents the priority of the message
- RTR which is Remote Transmission Request
- DLC which is Data Length Code
- Eight data bytes

The following figure represents the content of a CAN Message:

Message (10 bytes)							
11 bits Identifier			RTR		DLC		
Data byte 1	Data byte 2	Data byte 3	Data byte 4	Data byte 5	Data byte 6	Data byte 7	Data byte 8

The message is transmitted serially onto the bus using a non-return-to-zero (NRZ) format and may be received by all nodes.

Message transfer is manifested and controlled by four different frame types:

- A DATA FRAME carries data from a transmitter to the receivers.
- A REMOTE FRAME is transmitted by a bus unit to request the transmission of the DATA FRAME with the same IDENTIFIER.
- An ERROR FRAME is transmitted by any unit on detecting a bus error.
- An OVERLOAD FRAME is used to provide for an extra delay between the preceding and the succeeding DATA or REMOTE FRAMEs.

Data Transmission:

At this point, we have already known about CAN Node and Message. This part will give us an explaination of how data is transmitted through CAN BUS.

The CAN specifications use the terms "dominant" bits and "recessive" bits where dominant is a logical 0 and recessive is a logical 1. The idle state is represented by the recessive level (Logical 1). If one node transmits a dominant bit and another node transmits a recessive bit then there is a collision and the dominant bit "wins". This means there is no delay to the higher-priority message, and the node transmitting the lower priority message automatically attempts to re-transmit six bit clocks after the end of the dominant message. This makes CAN very suitable as a real time prioritized communications system.

The basic principle of CAN requires that each node listens to the data on the CAN network including the data that the transmitting node is transmitting. If a logical 1 is transmitted by all transmitting nodes at the same time, then a logical 1 is seen by all of the nodes, including both the transmitting nodes and receiving nodes. If a logical 0 is transmitted by all transmitting nodes at the same time, then a logical 0 is seen by all nodes. If a logical 0 is being transmitted by one or more nodes, and a logical 1 is being transmitted by one or more nodes, then a logical 0 is seen by all nodes including the node(s) transmitting the logical 1. When a node transmits a logical 1 but sees a logical 0, it realizes that there is a contention and it quits transmitting.

By using this process, any node that transmits a logical 1 when another node transmits a logical 0, the node transmitting a logical 1 will lose the arbitration. A node that loses arbitration re-queues its message for later transmission and the CAN frame bit-stream continues without error until only one node is left transmitting. This means that the node that transmits the first 1 loses arbitration. Since the 11 (or 29 for CAN 2.0B) bit identifier is transmitted by all nodes at the start of the CAN message, the node with the lowest identifier transmits more zeros at the start of the frame, and that is the node that wins the arbitration or has the highest priority.

In brief, the data transmission is under control of the following rules:

- CAN Arbitration: 0 (dorminant)

  1 (recessive)
- A Node transmitting logical 0 (dorminant) always win over a Node transmitting logical 1 (recessive).
- The ID (identifier) determines which node will take over the bus and start data transmission. The node with the lowest identifier wins the arbitration (or has the highest priority).

To understand more about this, let's consider this example.

	Start Bit	ID Bits										The Rest of the Frame
		10	9	8	7	6	5	4	3	2	1	0
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN Data	0	0	0	0	0	0	0	0	1	1	1	1	

Figure 2.2c. CAN transmission example

We are considering an 11-bit CAN network, with two nodes with IDs of 15 (binary representation, 00000001111) and 16 (00000010000). If these two nodes transmit at the same time, each will first transmit the start bit then transmit the first six zeros of their ID with no arbitration decision being made.

When the 8th bit is transmitted (the ones highlined with green and red), the node with the ID of 16 transmits a 1 (recessive) for its ID, and the node with the ID of 15

transmits a 0 (dominant) for its ID. According to CAN arbitration, Node 16 stops transmitting which allows the node with ID of 15 to continue its transmission without any loss of data. In conclusion, the node with the lowest ID will always win the arbitration, and therefore has the highest priority.

Extra information: About the data throughput on the CAN bus: CAN bus can use multiple baud rates up to 1Mbps. Also, The CAN bus communication enables bus-loads of up to 100% (data being transmitted all the time and all nodes can transmit, allowing full usage of the nominal bit rate.

2.3. Hall Effect sensor

A **Hall effect sensor** is a transducer that varies its output voltage in response to a magnetic field. Hall effect sensors are used for proximity switching, positioning, speed detection, and current sensing applications.

In its simplest form, the sensor operates as an analog transducer, directly returning a voltage. With a known magnetic field, its distance from the Hall plate can be determined. Using groups of sensors, the relative position of the magnet can be deduced.

Frequently, a Hall sensor is combined with circuitry that allows the device to act in a digital (on/off) mode, and may be called a switch in this configuration. Commonly seen in industrial applications such as the pictured pneumatic cylinder, they are also used in consumer equipment; for example some computer printers use them to detect missing paper and open covers. When high reliability is required, they are used in keyboards.

When a beam of charged particles passes through a magnetic field, forces act on the particles and the beam is deflected from a straight path. The flow of electrons through a conductor is known as a beam of charged carriers. When a conductor is placed in a magnetic field perpendicular to the direction of the electrons, they will be deflected from a straight path. As a consequence, one plane of the conductor will become

negatively charged and the opposite side will become positively charged. The voltage between these planes is called Hall voltage.

When the force on the charged particles from the electric field balances the force produced by magnetic field, the separation of them will stop. If the current is not changing, then the Hall voltage is a measure of the magnetic flux density. Basically, there are two kinds of Hall effect sensors. One is linear which means the output of voltage linearly depends on magnetic flux density; the other is called threshold which means there will be a sharp decrease of output voltage at each magnetic flux density.

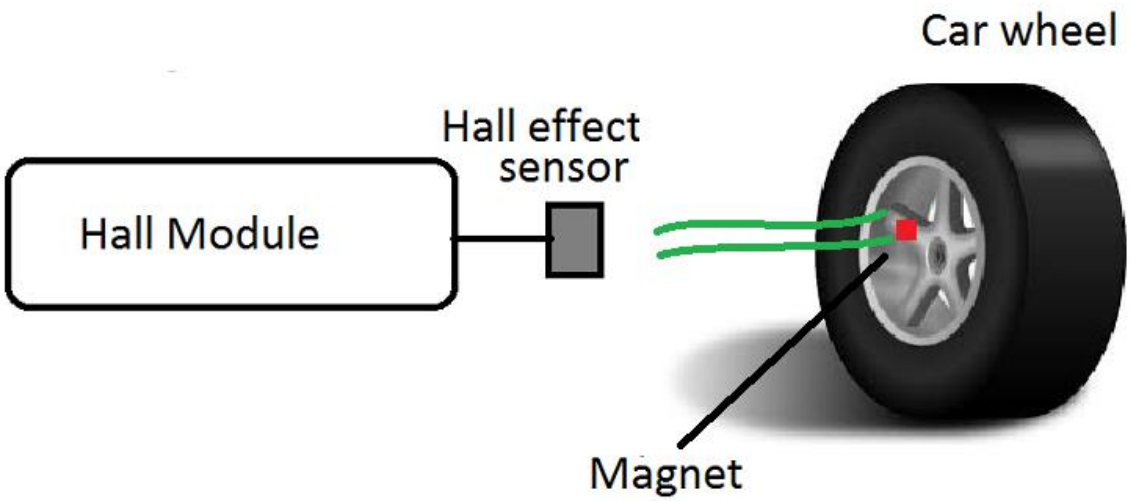


Figure 2.3. Hall effect implementation

We apply Hall Effect to calculate the velocity of the car as well as the distance it travels. Every time the magnet gets near the sensor, a pulse is created and causes the output pin to toggle. For consecutive times that the magnet passes the sensor, we can come up with a formula to calculate the velocity of the car (applying for one magnet) and the distance it travels.

The distance the car travels is:

$$Distance = \mathrm{n} * (\text{Perimeter of wheel}) \ (in\ centemeter) \quad \textbf{(1)}$$

Where:

- **n**: Numbers of times the magnet passes the sensor

2.4. Stepper motor

A stepper motor or step motor or stepping motor is a brushless DC electric motor that divides a full rotation into a number of equal steps. The motor's position can then be commanded to move and hold at one of these steps without any feedback sensor (an open-loop controller), as long as the motor is carefully sized to the application in respect to torque and speed.

Switched reluctance motors are very large stepping motors with a reduced pole count, and generally are closed-loop commutated.

There are several different ways of driving the stepper motor: We only focus on the one we are using to drive the stepper motor: Microstepping.

Microstepping

The most common method of controlling stepper motors nowadays is the Microstepping. In this mode we provide variable controlled current to the coils in form of sin wave. This will provide smooth motion of the rotor, decrease the stress of the parts and increase the accuracy of the stepper motor.

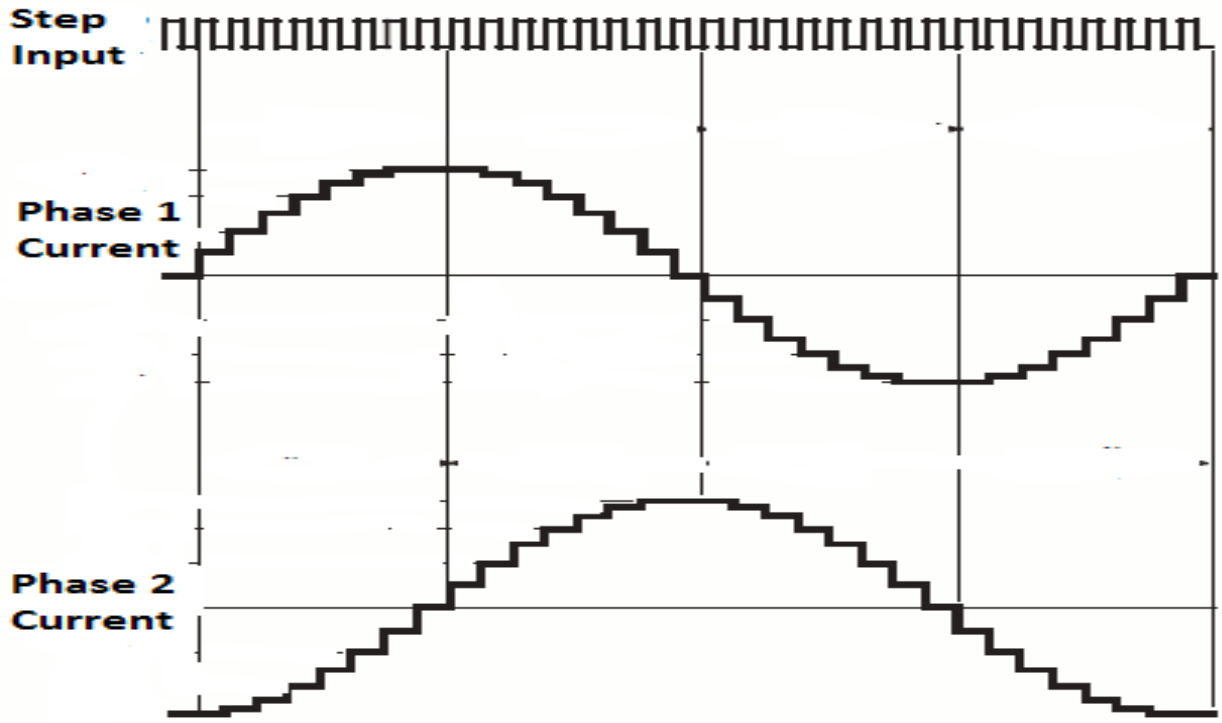


Figure 2.4. Microstepping

2.5. Bluetooth

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices, and building personal area networks (PANs).

In our world of embedded electronics hackery, Bluetooth serves as an excellent protocol for wirelessly transmitting relatively small amounts of data over a short range (<100m). It's perfectly suited as a wireless replacement for serial communication interfaces.

Bluetooth networks (commonly referred to as piconets) use a master/slave model to control when and where devices can send data. In this model, a single master device can be connected to up to seven different slave devices. Any slave device in the piconet can only be connected to a single master

The master coordinates communication throughout the piconet. It can send data to any of its slaves and request data from them as well. Slaves are only allowed to transmit to and receive from their master. They can't talk to other slaves in the piconet.

Every single Bluetooth device has a unique 48-bit address, commonly abbreviated BD_ADDR. This will usually be presented in the form of a 12-digit hexadecimal value. The most-significant half (24 bits) of the address is an organization unique identifier (OUI), which identifies the manufacturer. The lower 24-bits are the unique part of the address.

PART 2 - DESIGN

CHAPTER 3- PROPOSED ALGORITHMS/SYSTEMS

3.1. System Overview

Our system applies the technical solutions which are already mentioned in chapter 1 and chapter 2.

The objectives of our technical solutions, briefly:

Kinematic: This part is for the simulation on MATLAB. The simulation gives us a quick review of how the car parks into garage or parking spot.

Fuzzy logic: We apply fuzzy logic to the parking algorithm. See section **3.2**.

Ultrasonic sensor: Upon the signals feedback from these sensors, MCU1 can calculate the corresponding distances from the car to wall of garages.

CAN BUS: Enables connection between Node A and Node B.

Pulse Width Modulation (PWM): Enables microcontroller to control servo with corresponding steering angles.

Hall Effect: Feedback the distance the car travels.

Stepper motor: Determines the speed of the car (in our system, the speed is fixed), also it makes the car go forward, backward or stand still.

Bluetooth: Provides connection between the car and the Android Application GUI.

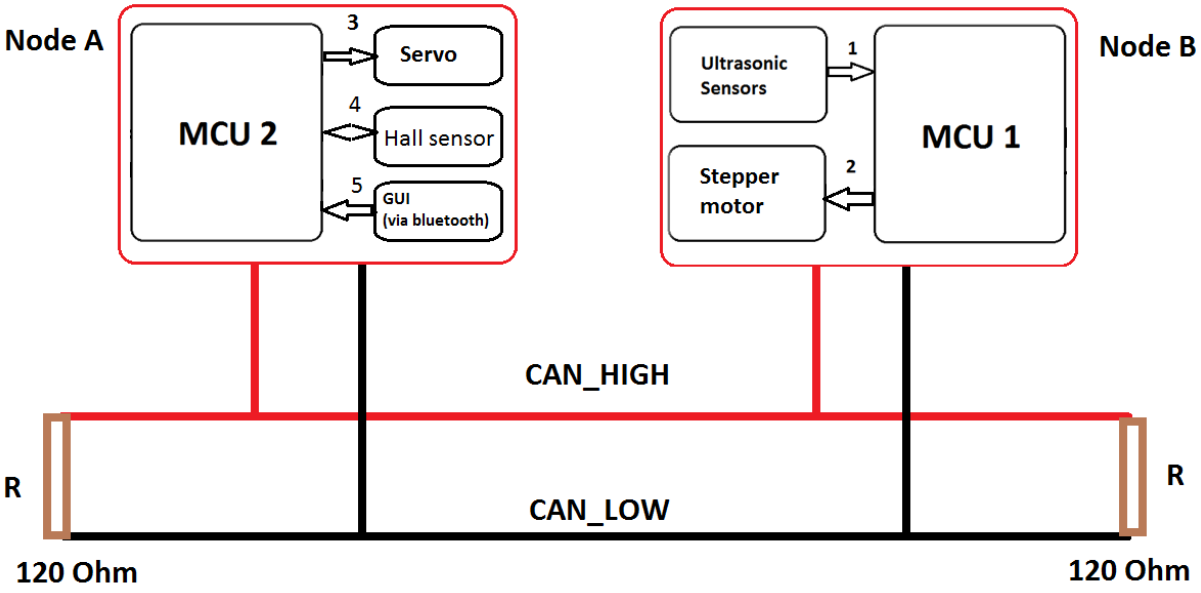


Figure 3.1a. System Overview

As the figure displays, there are two nodes communicating on a CAN BUS.

Node A consists of:

- A microcontroller (MCU2): Arduino UNO
- A CAN module: CAN BUS Shield
- Five ultrasonic sensors: SRF-05
- A stepper motor with an Easy driver

Node B consists of:

- A microcontroller (MCU1): Arduino UNO
- CAN module: CAN BUS Shield
- A bluetooth module: HC-05
- A Hall effect sensor

Besides, there are a total of five signals (denoted from 1 to 5) represented connections between the above equipement.

Signal 1: MCU1 gets the signals from ultrasonic sensors and accordingly, calculates distances from the car to the wall of garage.

Signal 2: MCU1 generates pulse, step and direction to the Easy driver, so that this driver can generate corresponding pulse to phase A and B of stepper motor, making the car move at certain speed.

Signal 3: MCU2 uses PWM to control the servo. Depends on the calculated distances, the steering angles may vary.

Signal 4: The Hall Effect module enables MCU2 to calculate the distance the car travels; thus, the car can detect whether the width of parking lot or garage is satisfied or not.

Signal 5: Interaction between user and car. While the user decides which mode the car will operate (parallel, garage parking or getting out), the car feedbacks its status. We will explain more about this in chapter 4.2.2: Android Application GUI.

Connection between Node A and Node B via CAN BUS:

To begin with, the IDs of Node A and Node B are 0x01 and 0x00, respectively. Therefore, Node B will win the arbitration and transmit its data first.

- Node B starts the data transmission by sending calculated distances to Node A. Accordingly, Node A can determine the steering angle.

- After Node B finishes the transmission, Node A starts to transmit data to Node B, which is the signal to control the stepper motor (go forward, backward or stand still).

Simply put, Node A controls the servo whist Node B takes over the stepper motor, in order to balance out the tasks that each Node has to handle.

Now we move to the algorithm. The algorithm has three parts:

- The first part: Wall alignment fuzzy logic system
- The second part: Parking slot scanning
- The last part: This part includes two finite state machine for self-parking car, which are Parallel parking and Garage Parking.

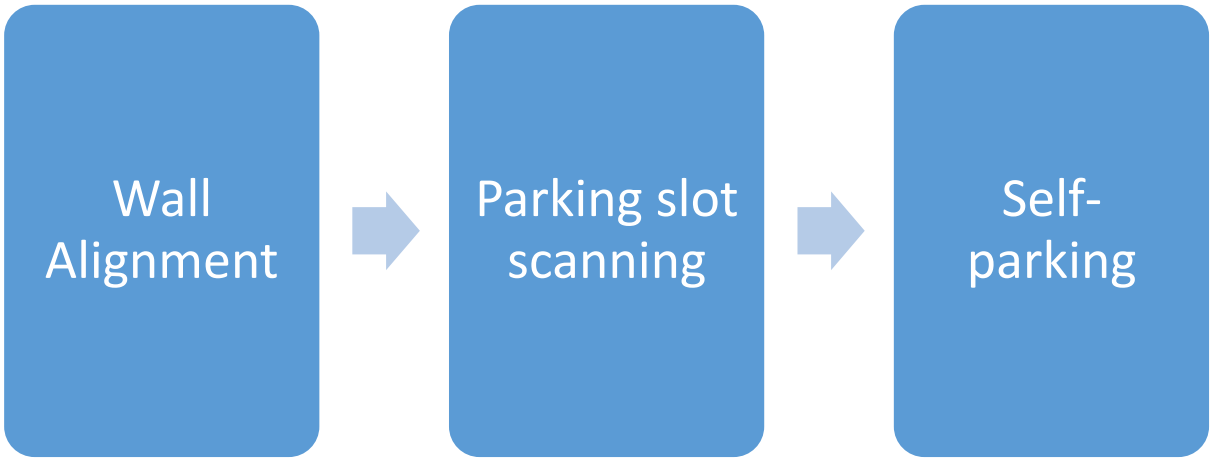


Figure 3.1b. Self-parking algorithm in steps

3.2. Wall alignment fuzzy logic system

The car-like robot begins searching the wall, following it and moving along the wall with a safe distance. In this project we determined this safe distance is about 10 to 12 centimeters. We call this step wall alignment state. The fuzzy logic is applied in this state to keep our car moving along the wall in safe distance. The detailed procedure of this state is as follows:

The inputs of this state are distance from Rear-Right and Front-Right of the car to the wall. We defined them as dRearRight and dFrontRight. Then we classify these distances in some particular ranges as the membership function for fuzzy logic inputs.

The figure 3.2a and figure 3.2b shows the membership functions dRearRight and dFrontRight.

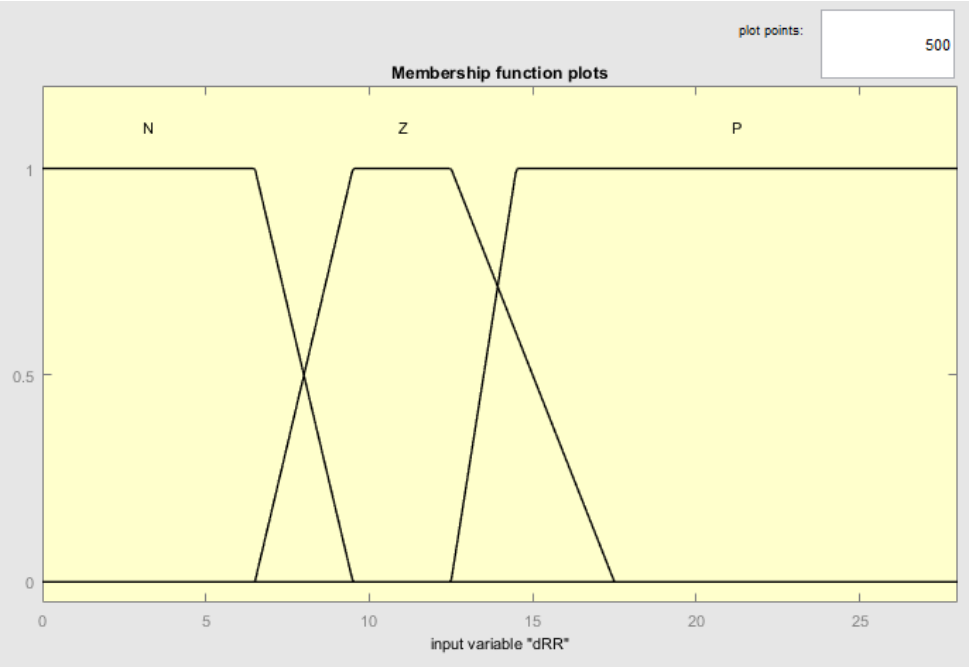


Figure 3.2a. Membership function plots of dRearRight

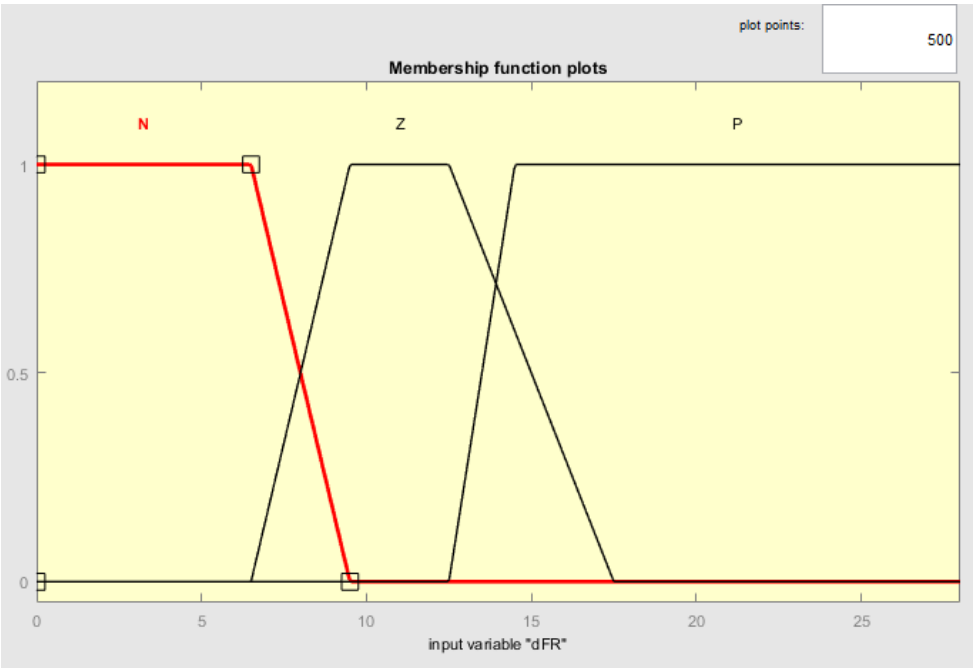


Figure 3.2b. Membership function plots of dFrontRight

The output of wall alignment fuzzy logic is the steering angle of a car. Car will turn right when the steering angle is negative, turn left when the steering angle is positive and go straight forward when steering angle is zero. The width for each interval can be adjusted to be associated with the dynamic environment. The centroid method is used

in de-fuzzification process. The table 3.2 shows the fuzzy rules between inputs and outputs of fuzzy logic.

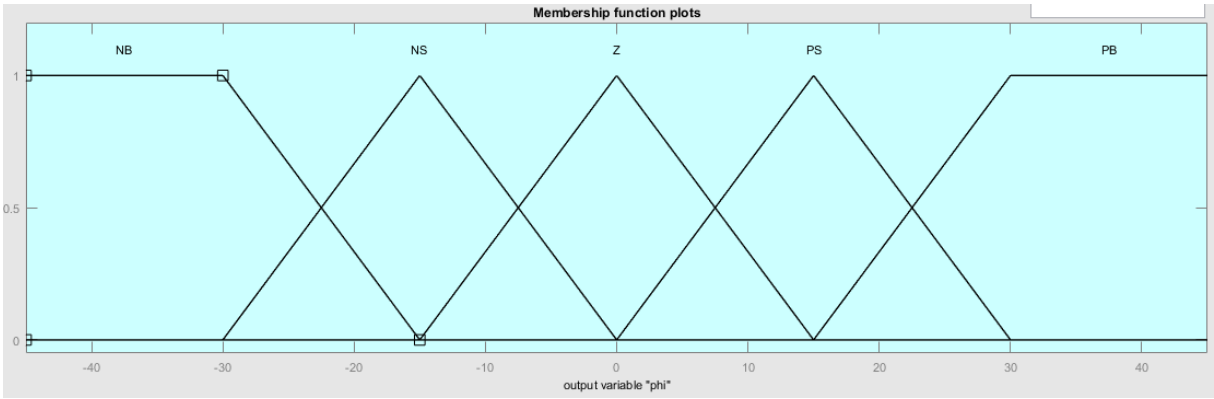


Figure 3.2c. Membership function plots of ∅

Rule base of fuzzy system

dF / dR / ∅	N	Z	P
N	PS	NS	NS
Z	PS	Z	NS
P	PS	PS	NS

Table 3.2. Fuzzy Logic Rule base

3.3. Parking slot scanning

This state will occur when the distance from the car front's wheel to the wall is greater than threshold distance (for example 30 centimeters in parallel parking) or the car's front wheel passes the parking slot. This threshold distance is the width of the parking slot. The measuring process is only started as the car's rear wheel passes the parking slot. The measuring process is ended when the car's rear wheel moves away from the parking slot or the distance from car's right rear to wall is in safe distance. If the gap

of a parking slot is less than parking length threshold the car will begin wall alignment state again to search for the next parking slot.

3.4. Self-parking: Parallel parking and Garage Parking

3.4.1. Parallel Parking Trajectory

We use two main semicircles to model the car's parallel parking trajectory. From Ackerman's Steering Theorem, the distance between the point located at half width of the car aligned with the two front wheels to the center of radius is R. Also, the maximum turning angle of the inner wheel is 33º and given by the equation $\theta_i = \text{atan}(\frac{L}{R})$ (L is length of car). The turning angle of the outer wheel is given by $\theta_o = \text{atan}(\frac{L}{W+R})$ (W is the width of the car) and $\theta_i > \theta_o$. with θ_i given as 33º:

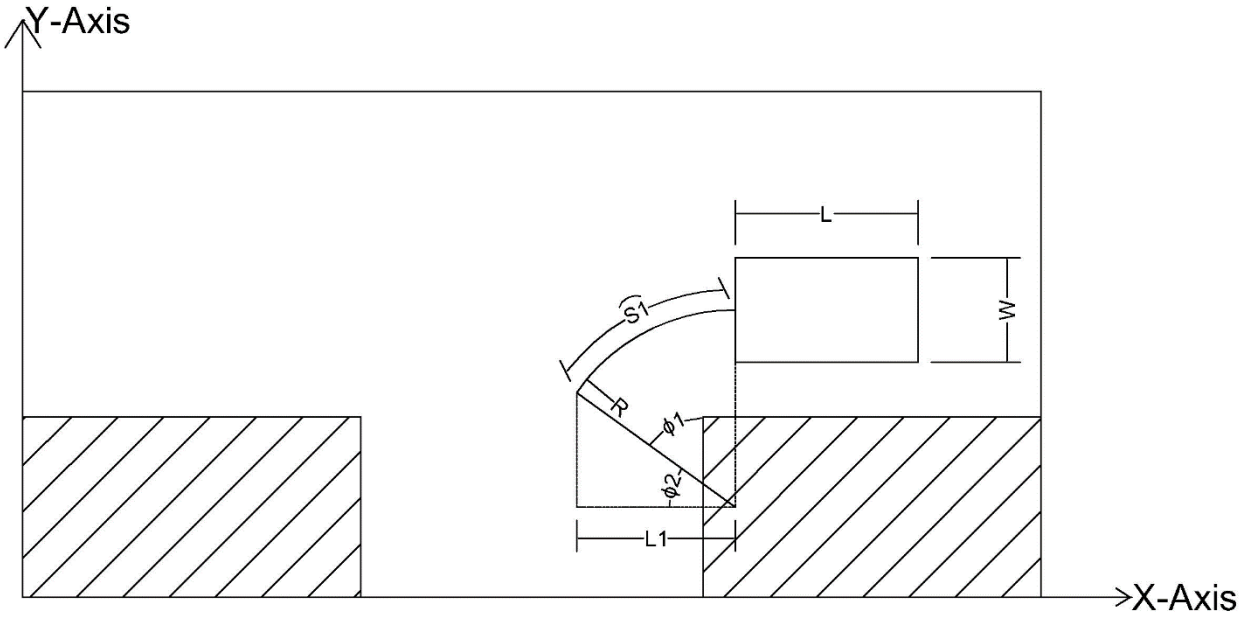


Figure 3.4.1a. Car's trajectory of the first semicircle phase when reversing to the parking lot.

$$R = \frac{L}{tan\theta_i} = \frac{26}{\tan(33°)} = 40\ (cm)$$

For the first half of the trajectory, the car will turn with turning radius of R where the center of the circle is located a distance $R - W/2$ from right rear of the car.

This trajectory will be terminated once the robot has traveled a horizontal distance of L1. The distance S1 which it has traveled corresponds to the angle $\emptyset_2$. This angle can be calculated by finding $\emptyset_1$:

$$\emptyset_2 = \cos^{-1}(\frac{R-L/2}{R}) \text{ then } S_1 = R\,\emptyset_2$$

Or we can terminate this phase by measuring the distance from car rear to the wall. The car will finish first semicircle when the distance from car rear to the wall is approximate 45 centimeters.

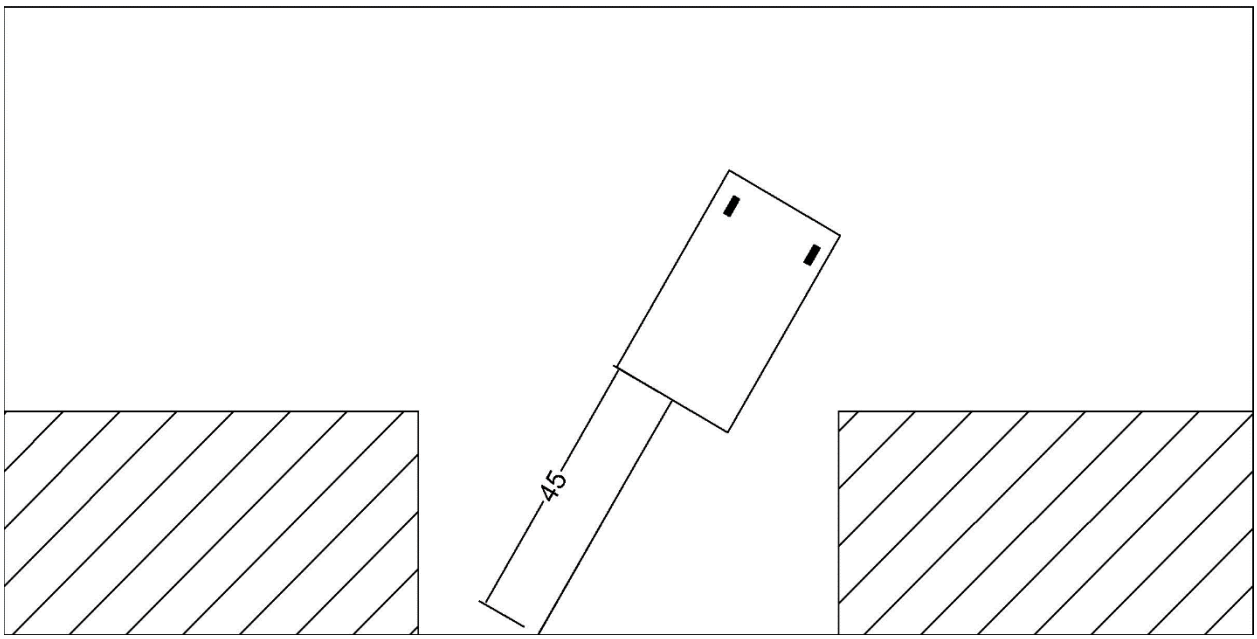


Figure 3.4.1b. Car finishes first semicircles at the point 45 centimeters from the wall

But the car is not in the parking lot yet so we straight the wheel and let the car back off $\Delta L = 10$ centimeters to ensure its rear is in the parking lot.

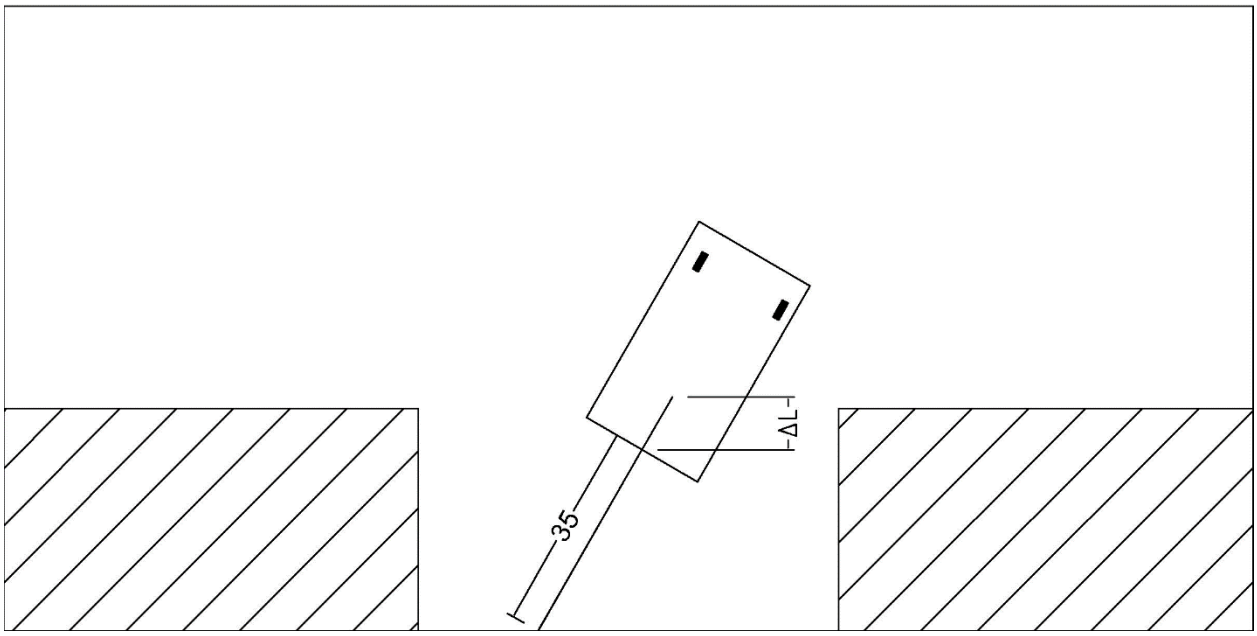


Figure 3.4.1c. Car's rear is in the parking lot.

Now we go to the second main semicircle of the car trajectory. For the second half of the trajectory, the car will turn in a semicircle again but this time in the opposite direction but with the same turning radius, R. This is shown on the following figure. The trajectory will be terminated once the car has nearly hits the behind car(wall).

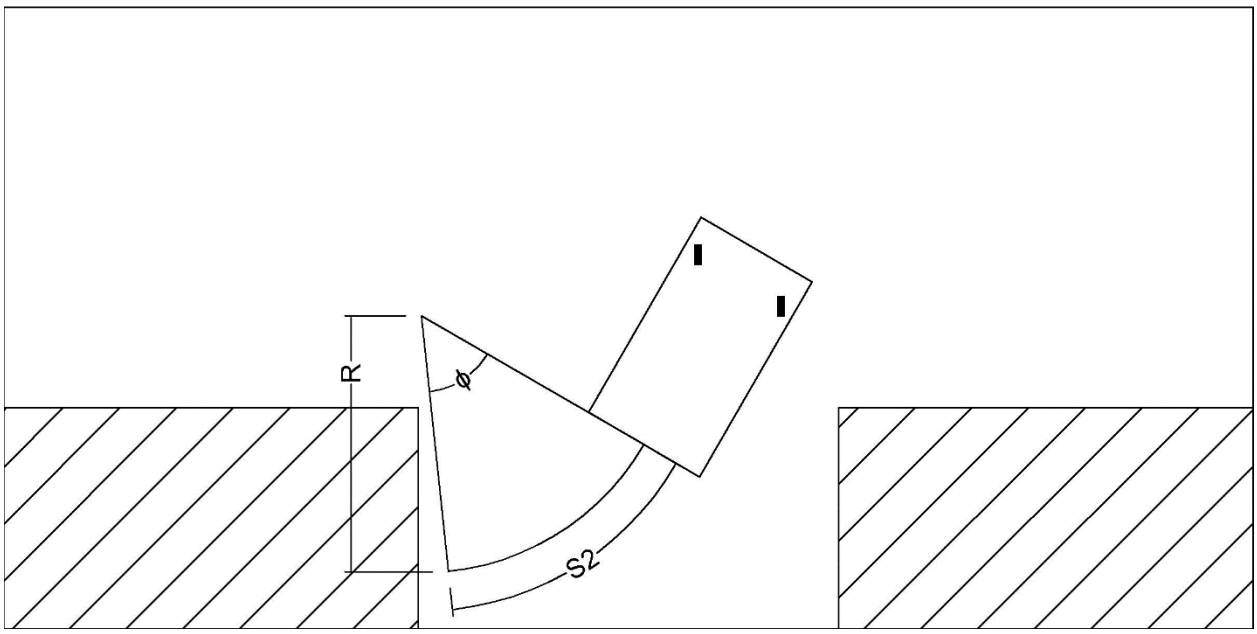


Figure 3.4.1d. Car's trajectory of the second semicircle phase.

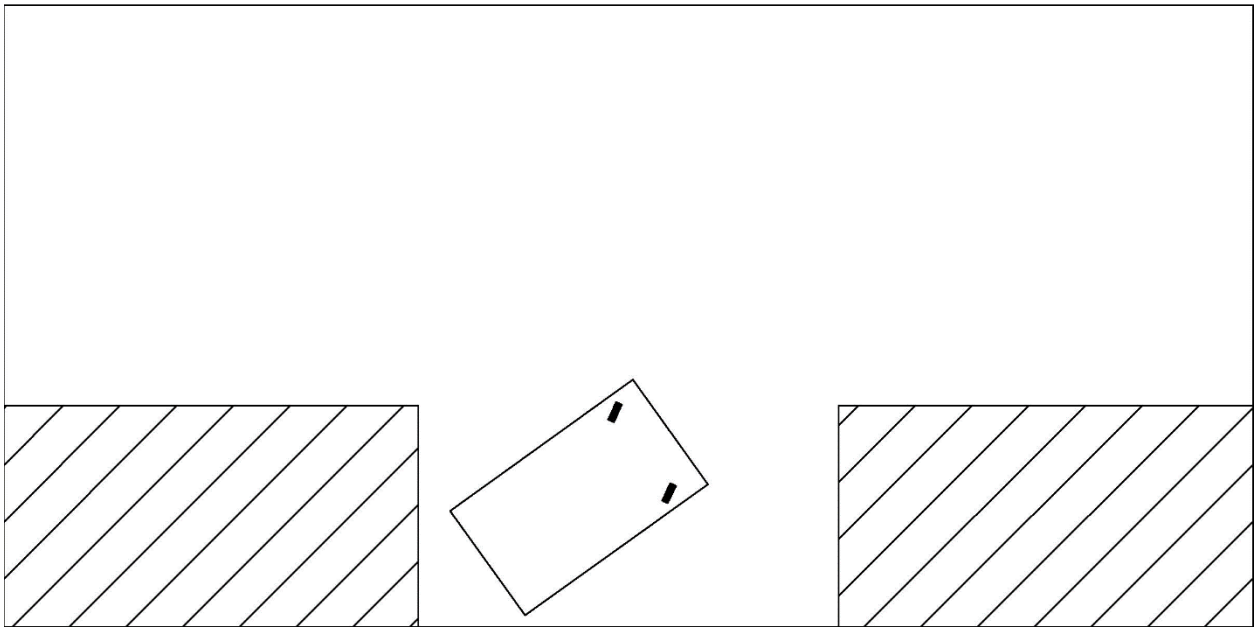


Figure 3.4.1e: Car's position when finishes second semicircle phase.

After completing the whole main trajectory, the car is not yet in the required position which is in the center of the parking lot and the car stays away from the wall from 10 – 15 centimeters. Therefore, the car translates into the Wall Alignment state to let the car in the required position. There is little trick that we use to apply wall alignment fuzzy logic in this state. When the car moving forward, the direction angle of the car remains the same as defuzzification process. But when the car moving backward, the direction angle of the car is opposite signed of defuzzification process. By doing that, we can take advantages of the wall alignment fuzzy logic to let the car in the required position. Be aware that the car will change state when it nearly hits the front car (wall) or the back car (wall). The Figures below describes the trajectory of this state.

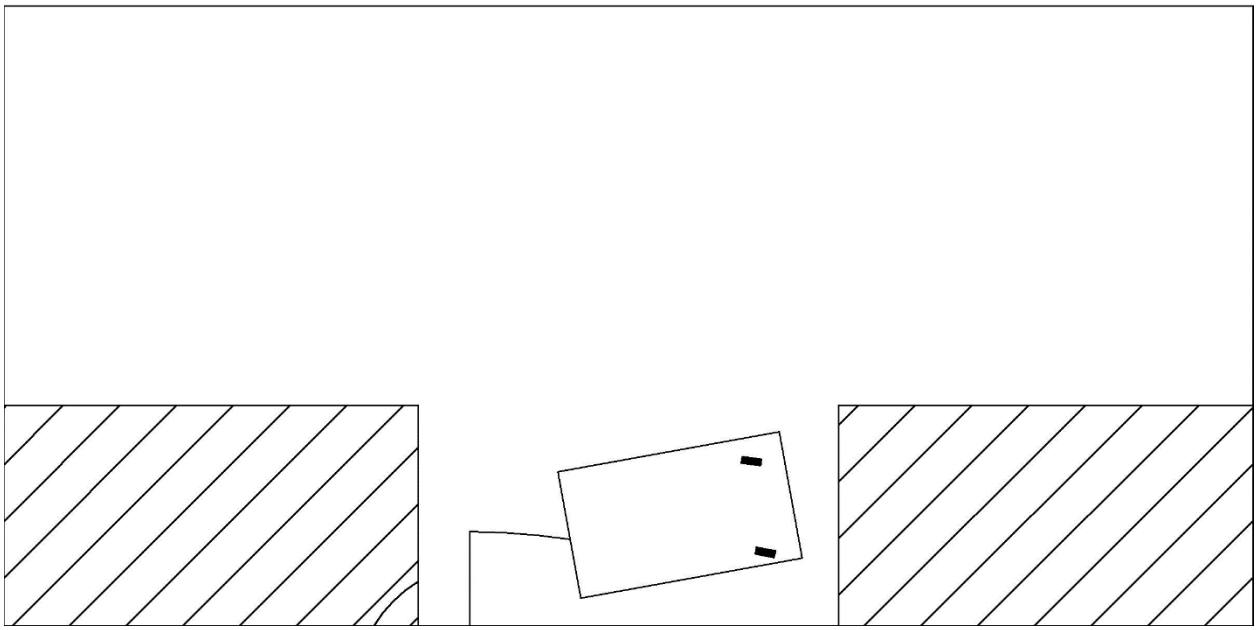


Figure 3.4.1f. Wall Alignment process when car moving forward.

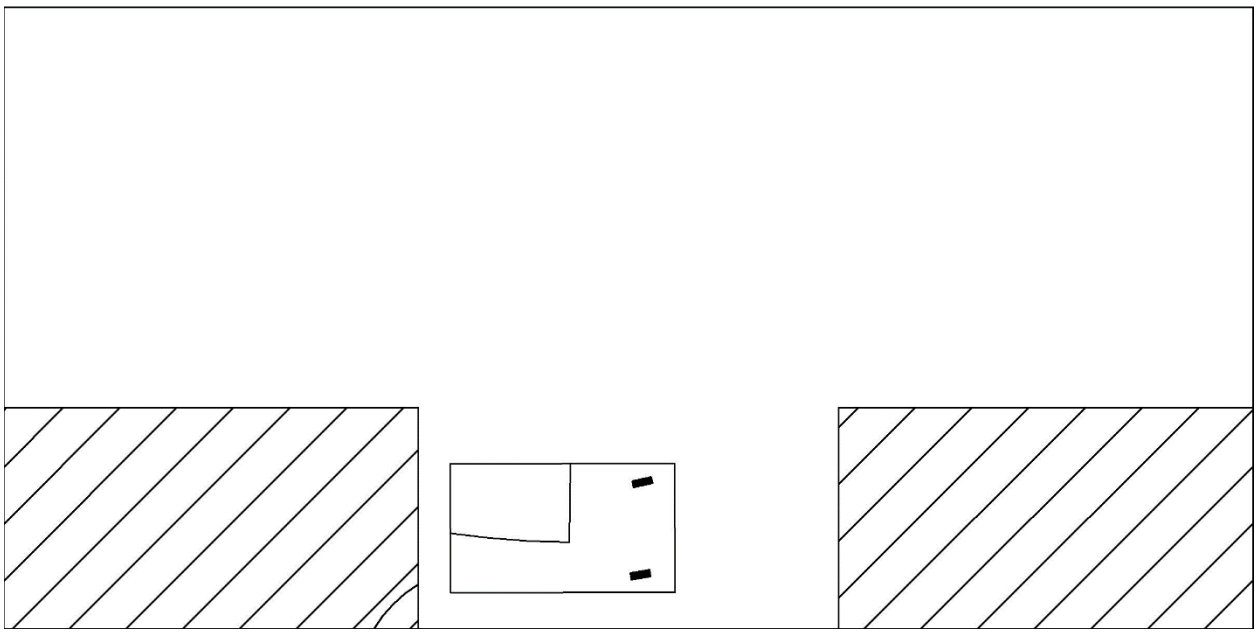


Figure 3.4.1g. Wall Alignment process when car moving backward.

Now the car stays away from the car the distance from 10 to 15 centimeters. The car need to move forward or backward to align itself to the center of the parking lot.

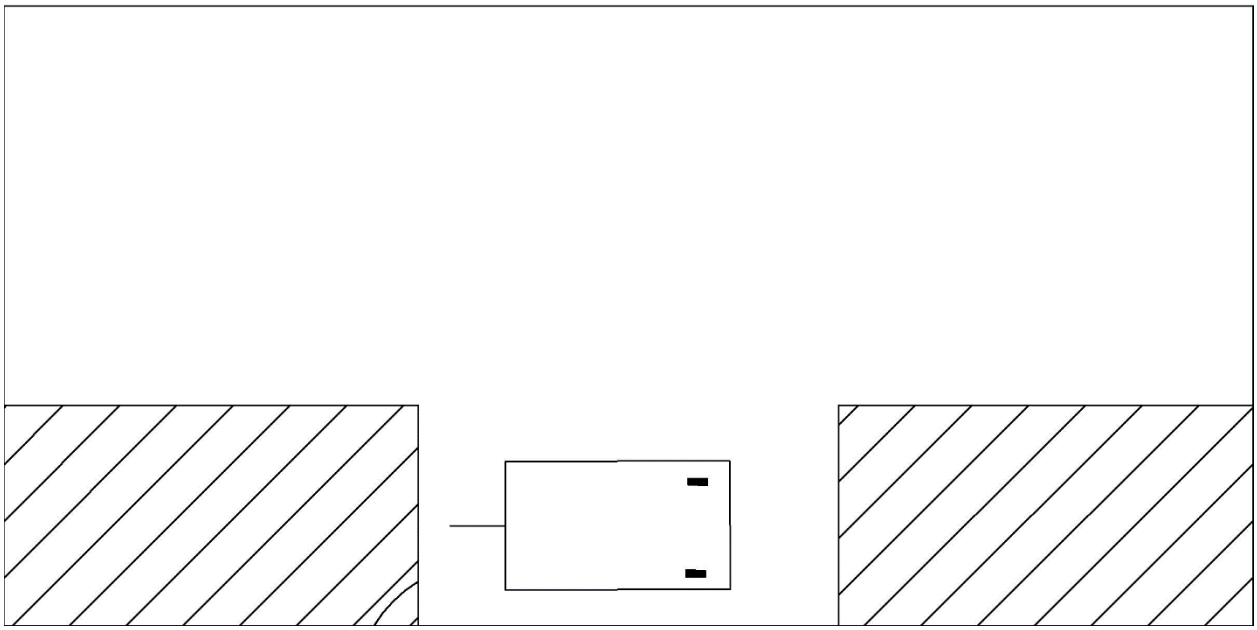


Figure 3.4.1h. Car is at the required position.

Finally, by combining the trajectory of its states, we got the full trajectory of autonomous parallel parking process as the figure below.

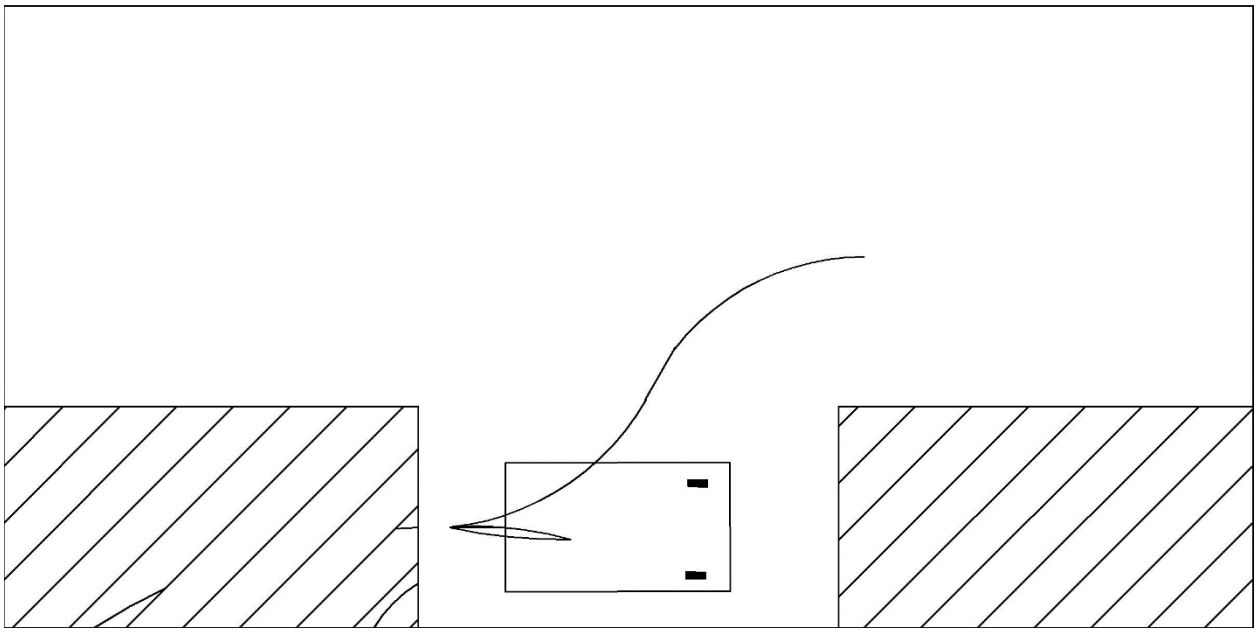


Figure 3.4.1i: Autonomous parallel car parking's trajectory.

The overall path trajectory for autonomous parallel car parking is referenced from paper **[6]** and **[7]**. The path trajectories for autonomous parallel car parking are slightly different for each car. The trajectory in **Figure 3.4.1i** is designed and calculated to be best fitted the real-life kinematic model of RC car that we use to implement our self-parking system on as well as the ambient environment.

3.4.2. Garage Parking Trajectory

Right after the parking slot scanning, the car stops when its rear passes the garage. As displayed in figure 3.4.2a, the car finds itself at the edge of the garage.

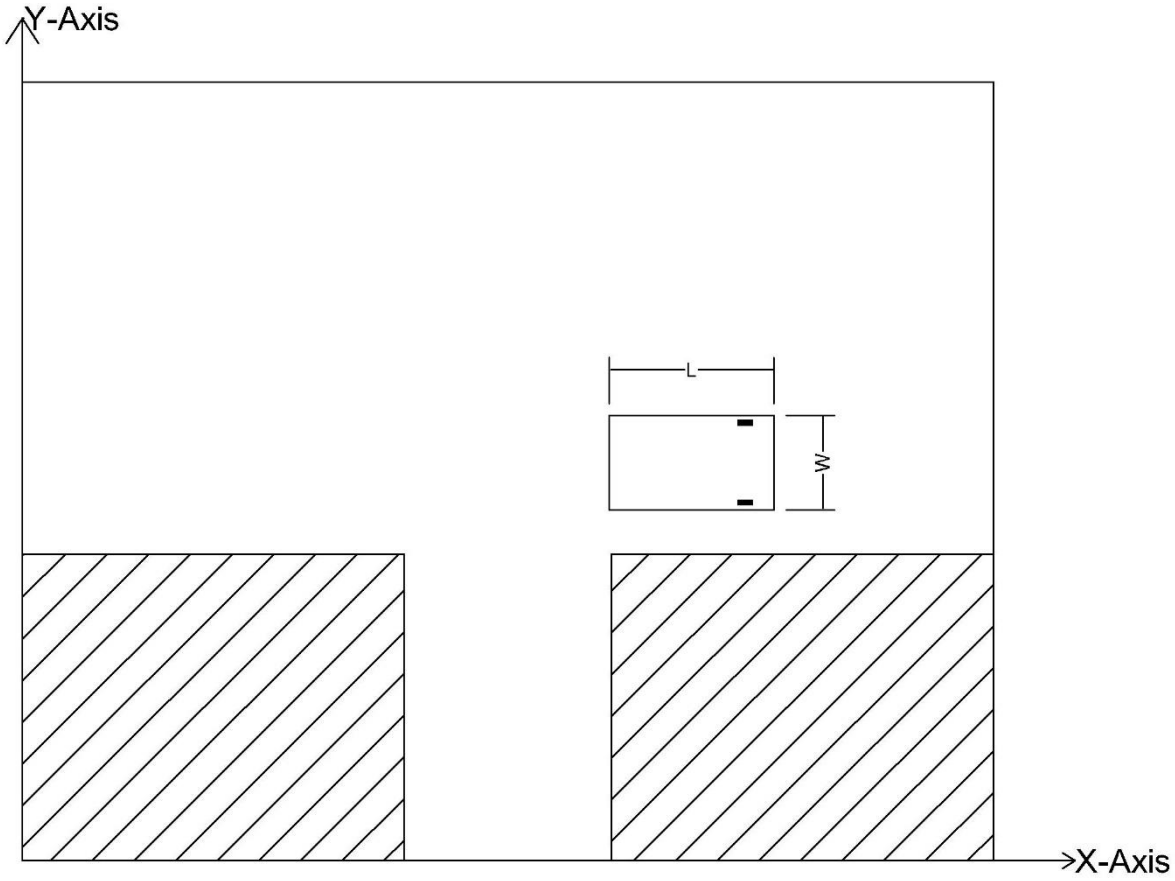


Figure 3.4.2a: Car's initial position for garage parking trajectory

Then the car goes backward for a distance so that it stops at the middle of garage width or the distance readings of rear right and middle right sensors are both greater than 60 cm. Already mentioned in parallel parking, we also apply another semicircle phase (the first semicircle phase in this garage parking) in order for the car to move forward with a steering angle of 33º.

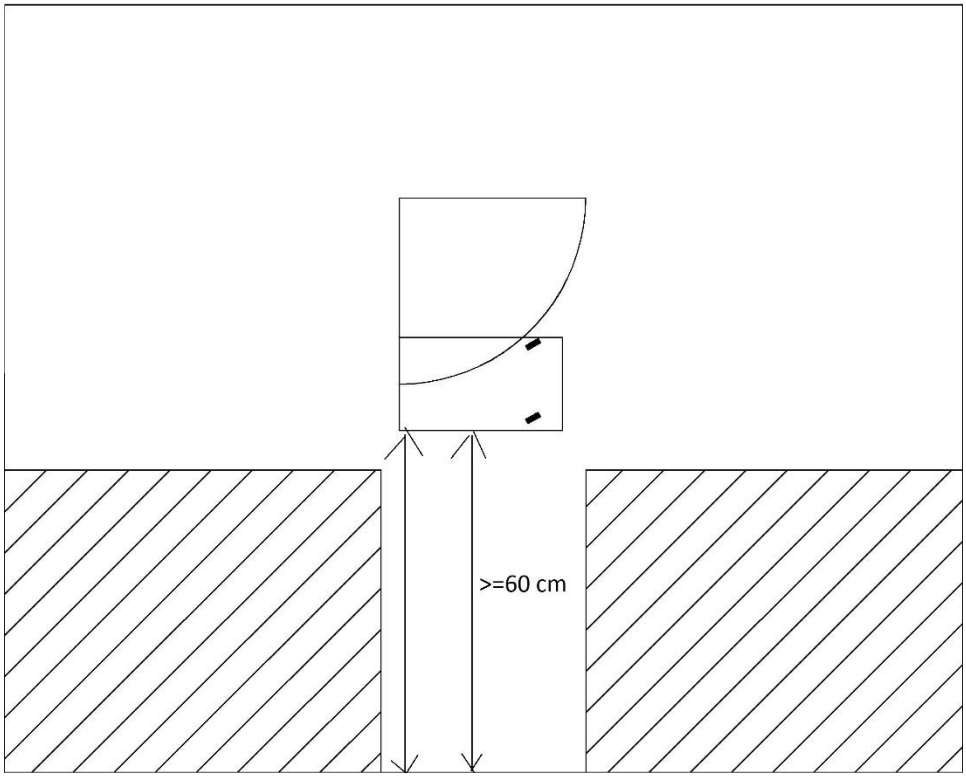


Figure 3.4.2b: Car's trajectory of the first semicircle phase (steering angle of 33º)

This makes the car move away from the garage to a certain posision, where the rear sensor reads a distance greater than 60 cm.

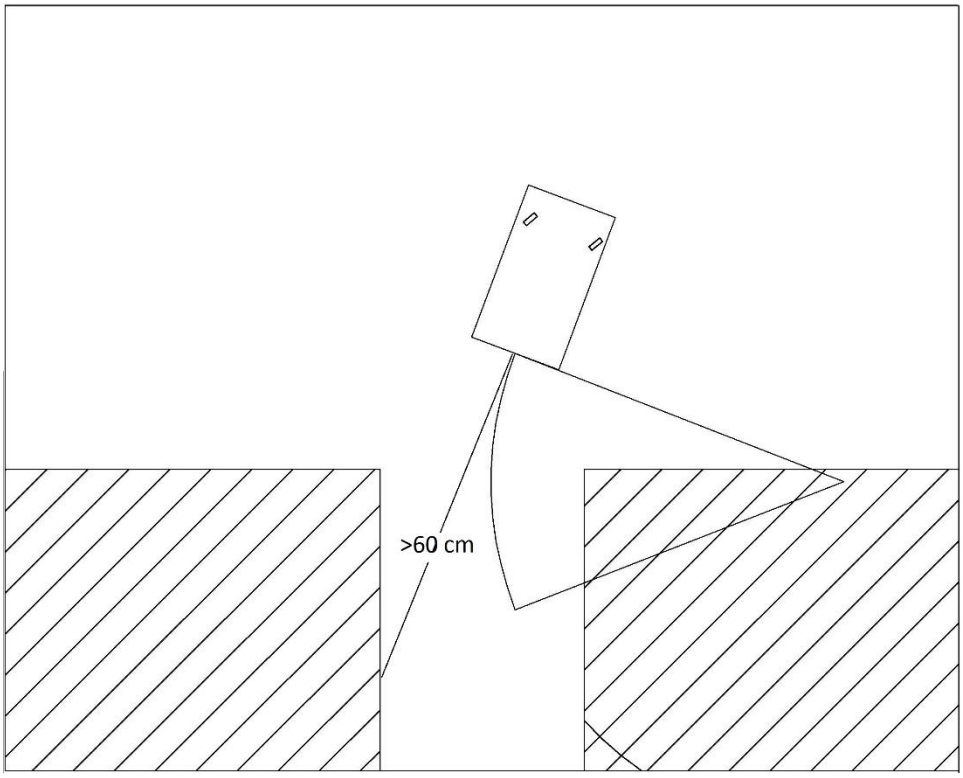


Figure 3.4.2c: Car is at the required position, rear distance greater than 60cm

The second semicircle phase is applied at this position. This state depends heavily on distance readings of three sensors: front-right, mid-right and rear-right sensors. The car keeps going backward for a steering angle of **-33º** until the distance readings of mid-right and rear-right sensors are identical (acceptable deviation of 2cm). In addition, distance reading of front-right sensor must be greater than 95 cm. At this point, we straight the wheel and let the car move backward to the garage.

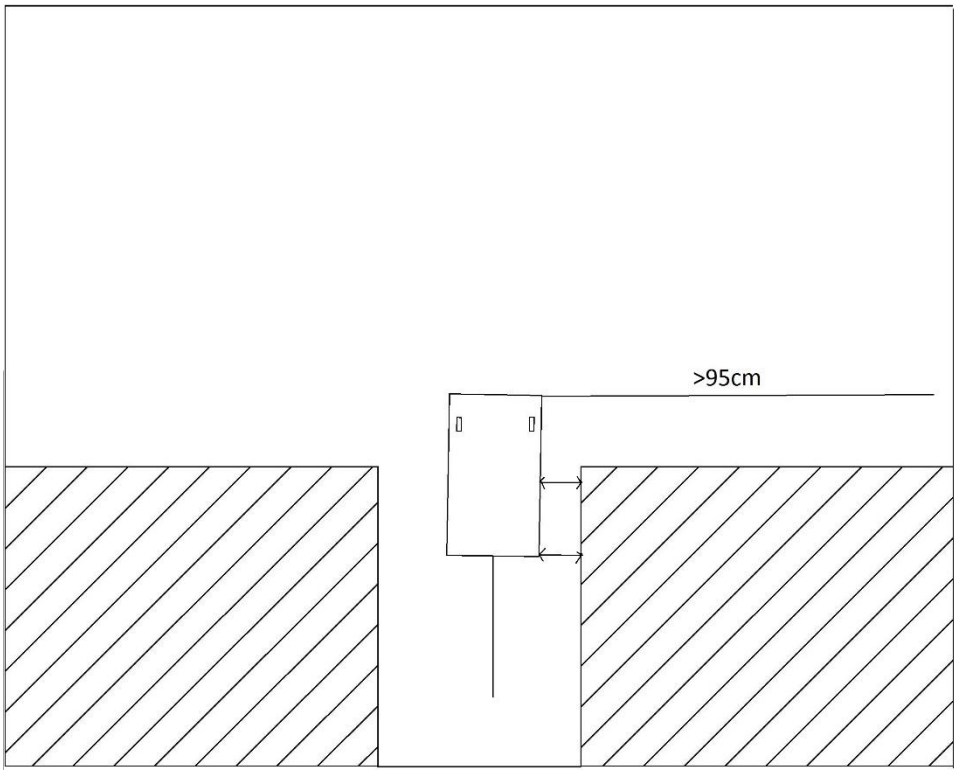


Figure 3.4.2d: Car goes backward right after sensors readings satisfied.

A combination of distance readings from three sensors ensures that the car will park perpendicular to the road, or horizontal to side walls of garage. When the rear sensor returns a distance of equal or less than 15cm, the car will stop and find itself in the middle of the garage.

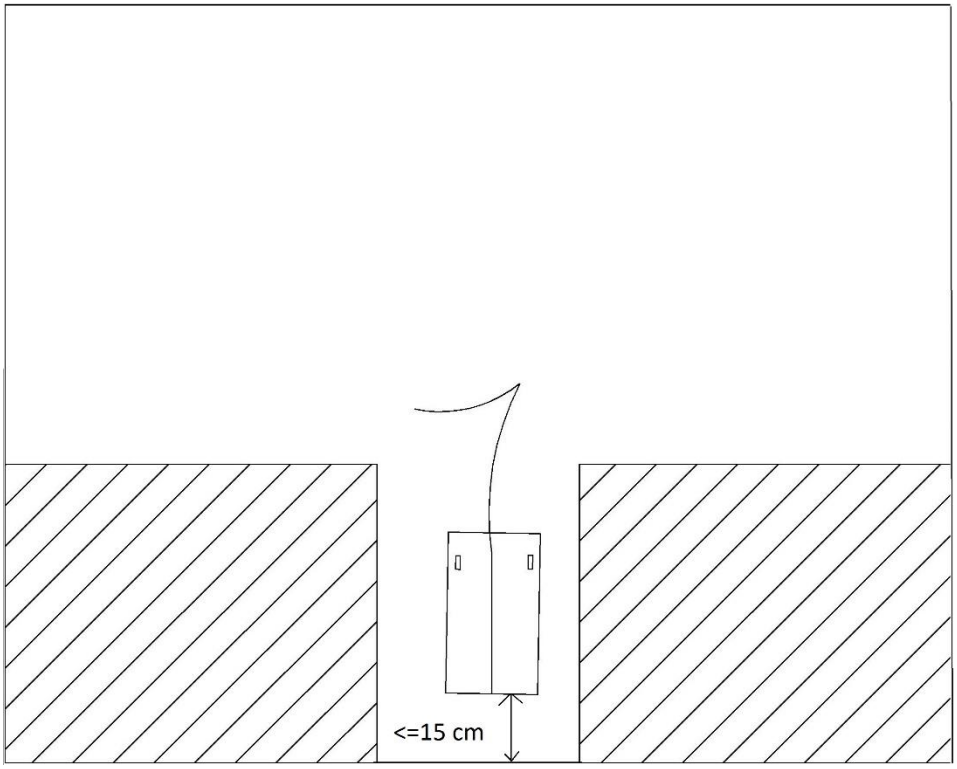


Figure 3.4.2e: Autonomous garage parking's trajectory

CHAPTER 4- EXPERIMENTAL RESULTS AND EVALUATION

4.1. Hardware design

In this section, firstly we will show our hardware design structure of our system. The specification of each equipment will be indicated afterward.

Pin – MCU1	Pin device
0	Echo – Middle Right Sensor
3	Echo – Right Front Sensor
4	Echo – Right Rear Sensor
5	Echo – Front Sensor
6	Echo – Rear Sensor
7	Dir – Easy Driver
8	Trig – all Sensor
9	Step – Easy Driver
10	CAN-SIG – CAN BUS 1
11	MOSI – CAN BUS 1
12	MISO – CAN BUS 1
13	SCK – CAN BUS 1
GND	GND – Battery
GND	GND – all Sensor
GND	GND – Easy Driver
Vin	VCC – Battery
5V	VCC – all Sensor

Table 4.1a. MCU1 PinOut

Pin – MCU2	Pin device
TX	RX - Bluetooth
RX	TX – Bluetooth
3	OUT – Hall Sensor
5	PWM – Servo
10	CAN-SIG – CAN BUS 2
11	MOSI – CAN BUS 2
12	MISO – CAN BUS 2
13	SCK – CAN BUS 2
GND	GND – Hall Sensor, Servo
GND	GND – Bluetooth
Power Jack	Battery
Vin	VCC – Bluetooth
5V	VCC – Hall Sensor, Servo

Table 4.1b. MCU2 PinOut

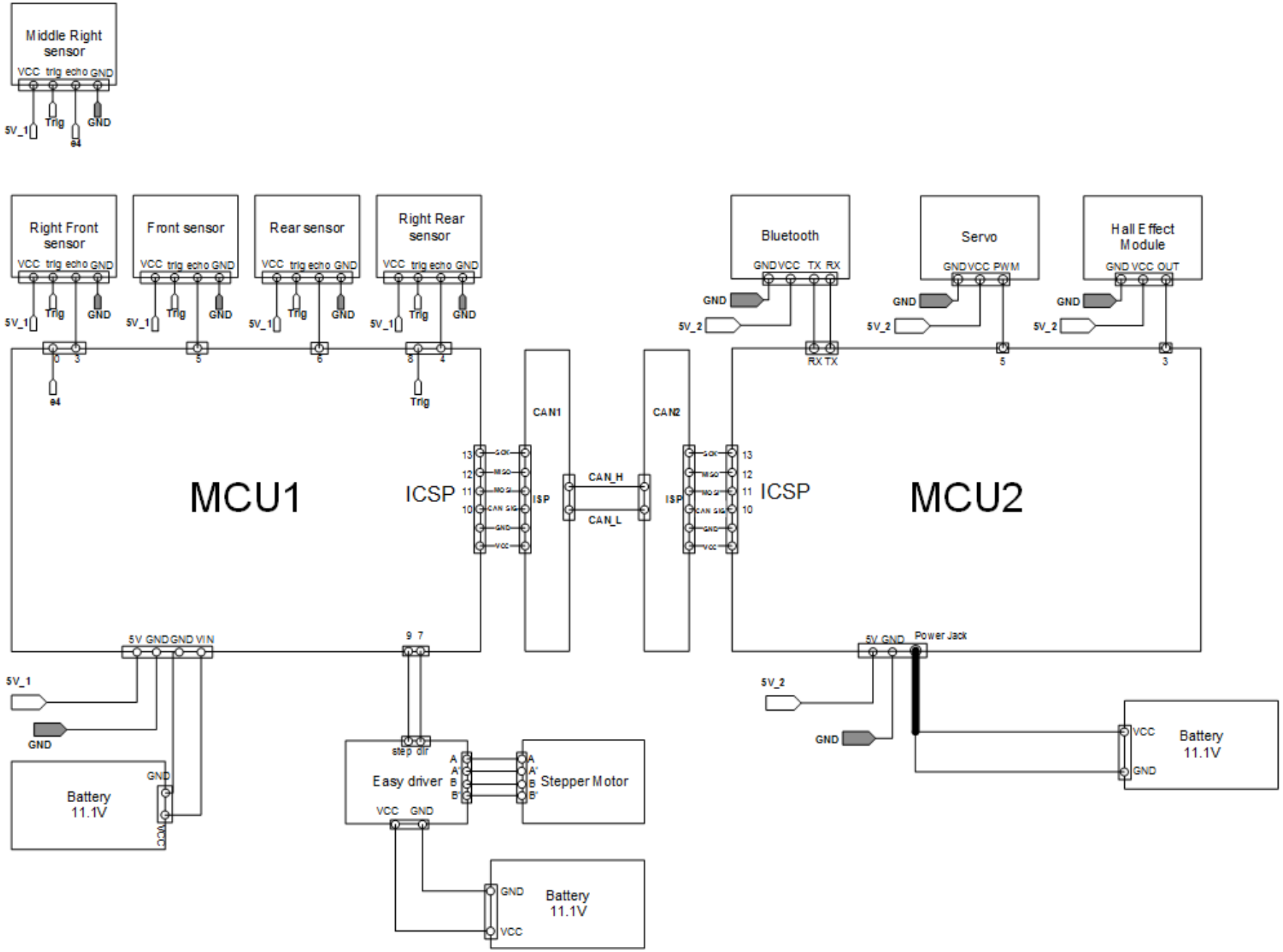


Figure 4.1. Hardware design structure of system

Specification of equipment and corresponding installation

4.1.1. Ultrasonic sensor

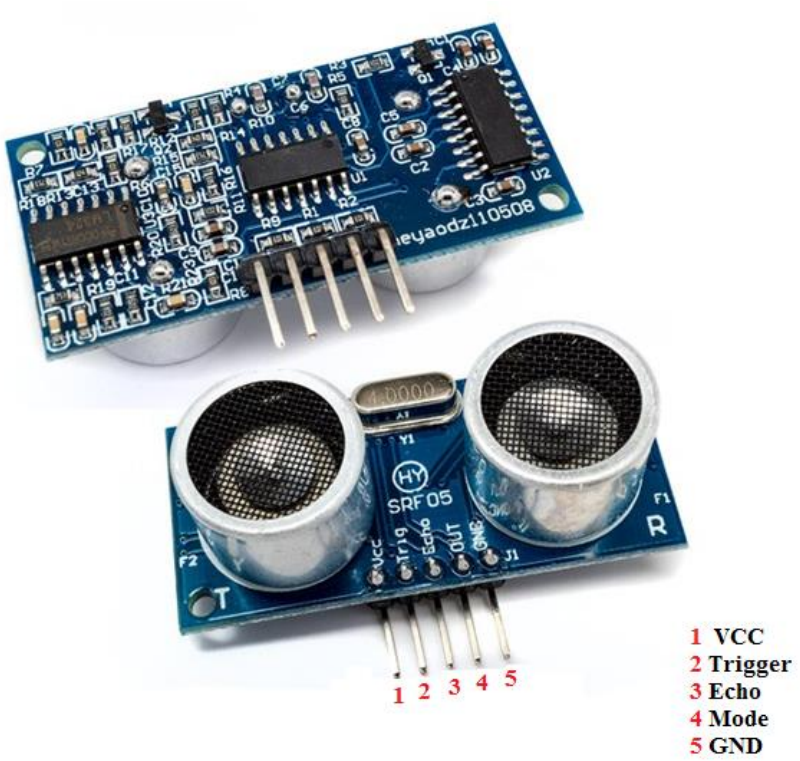


Figure 4.1.1a. An SRF05 module [8]

Voltage	4.5V to 5.5V
Current	10 to 40mA
Frequency	40KHz
Max Range	3 m
Min Range	3 cm
Angle	Up to 15 deg
Sensitivity	Detect 3 cm diameter broom handle at > 3 m
Input Trigger	10uS Min. TTL level pulse
Echo Pulse	Positive TTL level signal, width proportional to range
Size	43 mm x 20 mm x 17 mm height

Table 4.1.1a. SRF05 Specification

The beam pattern of the SRF05 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of the transducers used on the SRF05, taken from the manufacturer data sheet, is shown below (about 50 degree).

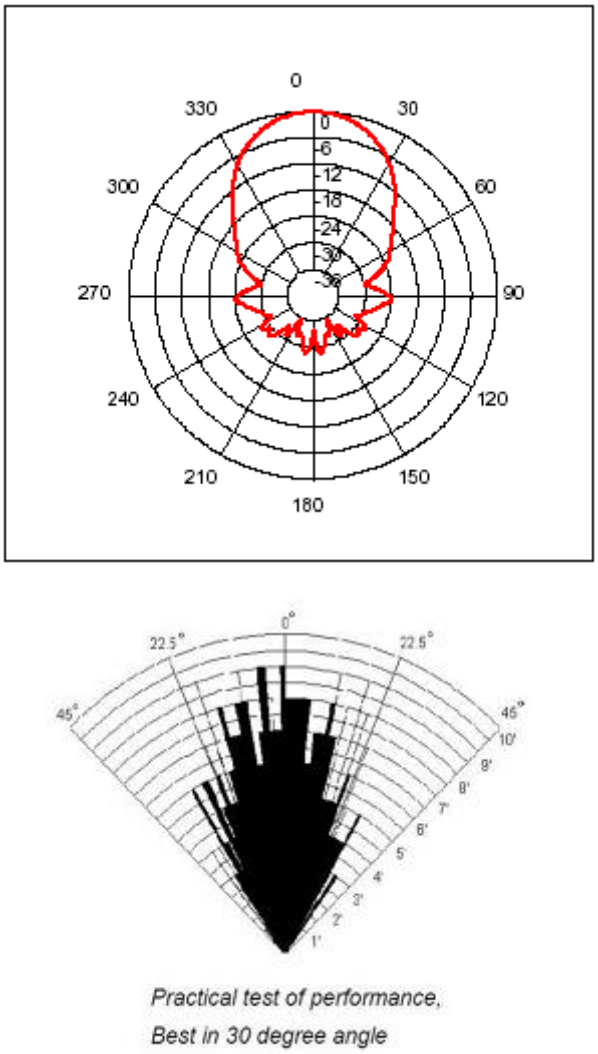


Figure 4.1.1b. SRF05 Beam pattern

The module can be used in two different modes: Dual Pin (Mode 1), Single Pin (Mode2). We are currently using Dual Pin mode. An explanation of this mode is as follows:

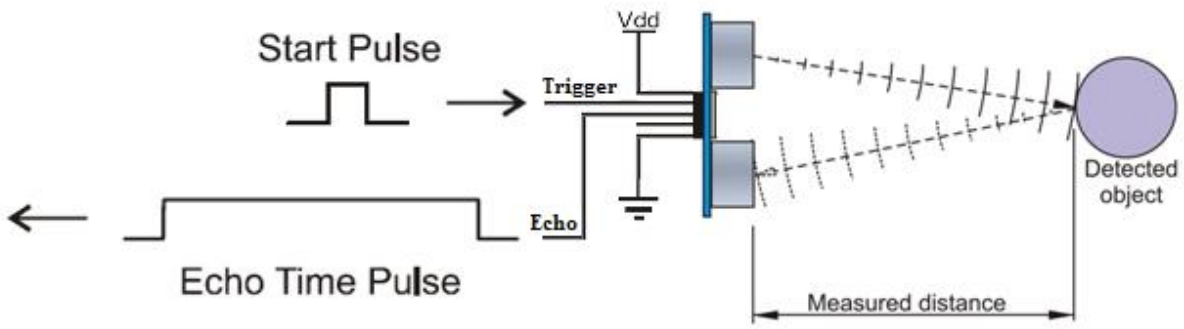
Mode 1: Dual Pin Connection Mode

This mode uses separate trigger and echo pins, and is the simplest mode to use. All code examples for the SRF04 will work for the SRF05 in this mode. To use this mode,

just leave the mode pin unconnected - the SRF05 has an internal pull up resistor on this pin.

VCC	Connect to 5V
Trigger	Connect to output pin of MCU
Echo	Connect to input pin of MCU
Mode	Do not connect
GND	Connect to ground

Table 4.1.1b. SRF05 Pinout (Mode 1)



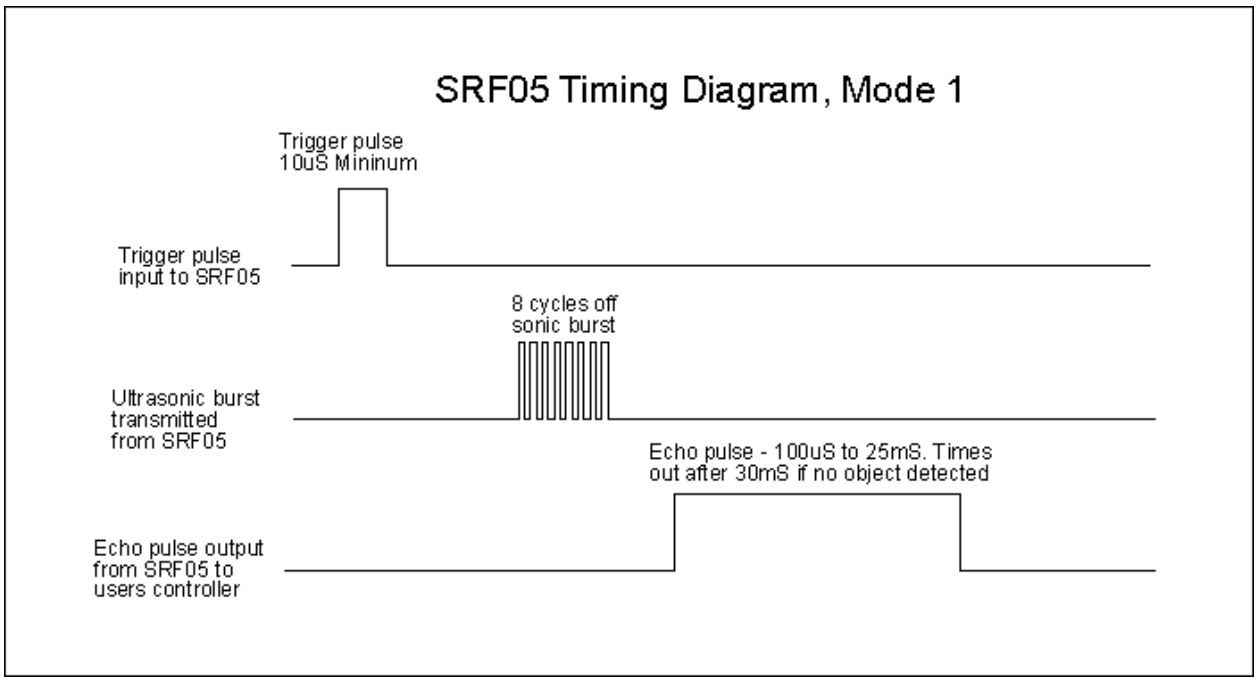


Figure 4.1.1c. SRF05 Implementation and Timing diagram, Dual Pin

4.1.2. CAN BUS Shield

A CAN BUS Shield[9] consists of two main components:

- CAN Controller: MCP2515
- CAN Transceiver: MCP2551

The specification of CAN BUS Shield is as follows:

- Implements CAN V2.0B at up to 1 Mb/s
- SPI Interface up to 10 MHz
- Standard (11 bit) and extended (29 bit) data and remote frames
- Two receive buffers with prioritized message storage
- Industrial standard 9 pin sub-D connector
- LED indicators

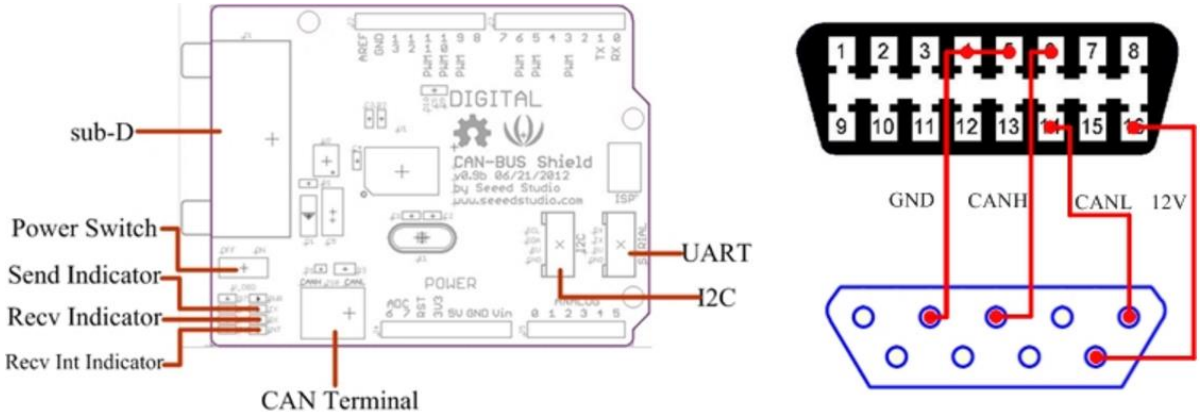


Figure 4.1.2a. CAN BUS Shield hardware overview

D0	Not Used
D1	Not Used
D2	**Receive Interrupt**
D3	Not Used

D4	Not_Used
D5	Not Used
D6	Not Used
D7	Not Used
D8	Not Used
D9	**SPI_CS(default)**
D10	**SPI_CS(selectable)**
D11	**SPI_MOSI**
D12	**SPI_MISO**
D13	**SPI_SCK**

Table 4.1.2a. Digital Pin used

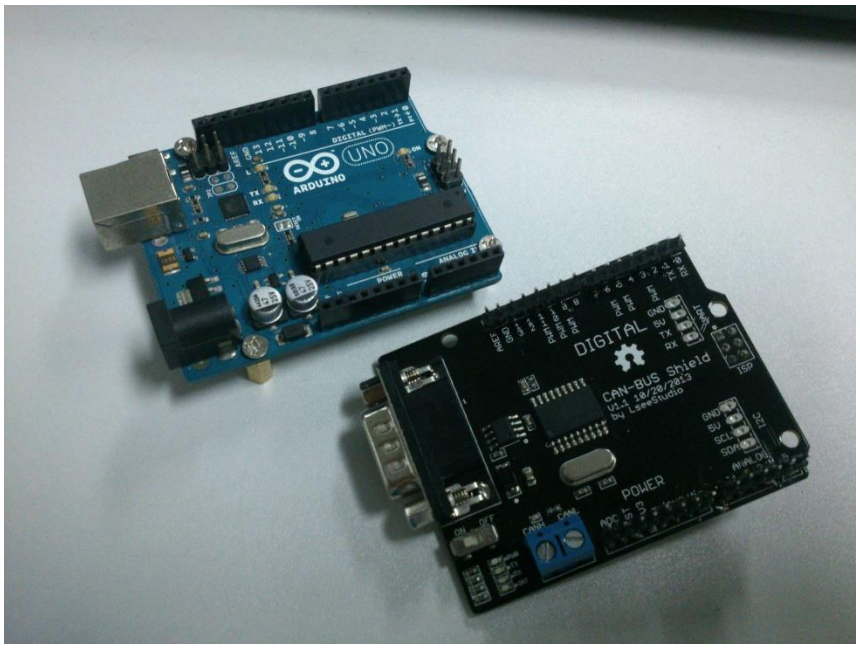


Figure 4.1.2b. Arduino UNO and CAN BUS Shield

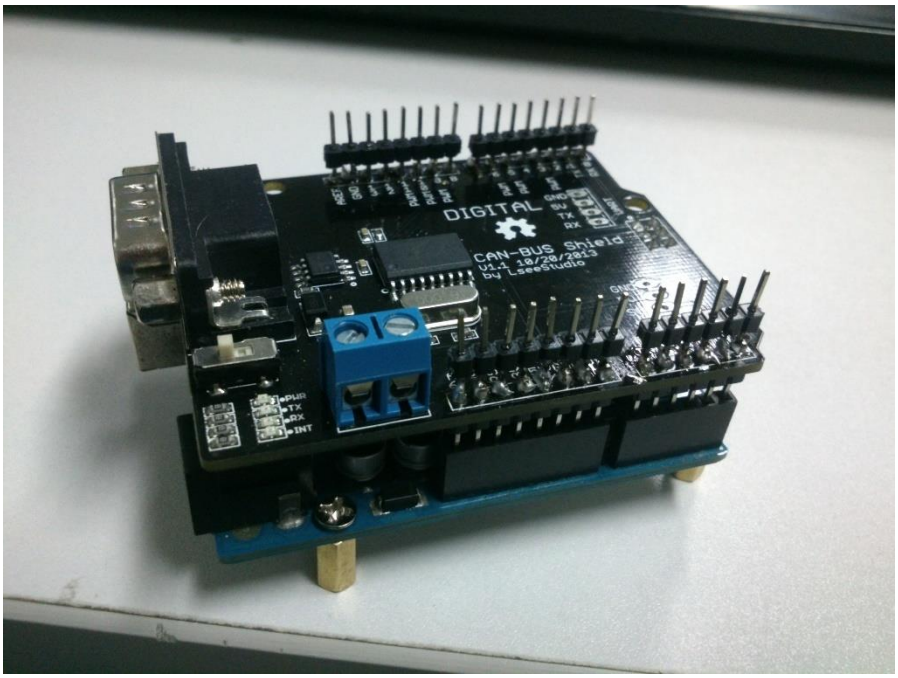


Figure 4.1.2c. A CAN Node after assembling Arduino and CAN BUS Shield

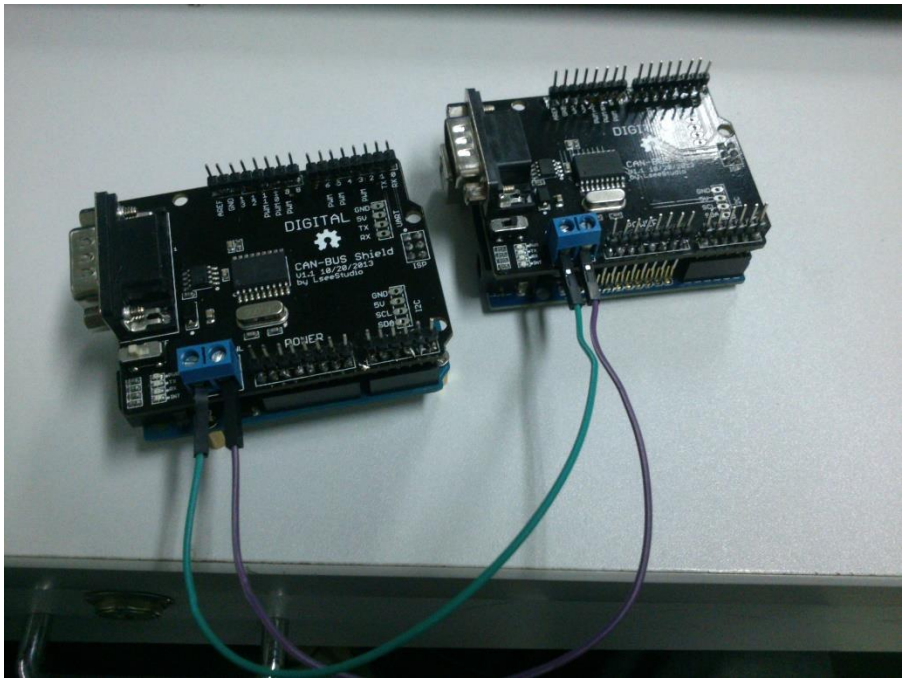


Figure 4.1.2d. Two Nodes connected via CAN High and CAN Low

4.1.3. Easy Driver – Stepper motor driver

The EasyDriver is a simple to use stepper motor driver, compatible with anything that can output a digital 0 to 5V pulse (or 0 to 3.3V pulse if you solder SJ2 closed on the EasyDriver). The EasyDriver requires a 6V to 30V supply to power the motor and can

power any voltage of stepper motor. The EasyDriver has an on board voltage regulator for the digital interface that can be set to 5V or 3.3V. Connect a 4-wire stepper motor and a microcontroller and you've got precision motor control! EasyDriver drives bi-polar motors, and motors wired as bi-polar. I.e. 4,6, or 8 wire stepper motors.

Specification:

- MS1 and MS2 pins broken out to change microstepping resolution to full, half, quarter and eighth steps (defaults to eighth)
- Compatible with 4, 6, and 8 wire stepper motors of any voltage
- Adjustable current control from 150mA/phase to 700mA/phase
- Power supply range from 6V to 30V. The higher the voltage, the higher the torque at high speeds.

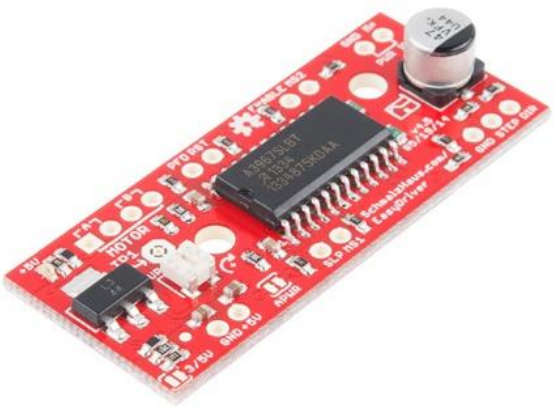


Figure 4.1.3. Easydriver-Stepper motor driver

4.1.4. Bluetooth module –HC05

Module HC-05 are based on the Cambridge Silicon Radio BC417 2.4 GHz Bluetooth Radio chip. This is a complex chip which uses an external 8 Mbit flash memory.
HC-05 is a more capable module that can be set to be either Master or Slave.

KEY	If brought HIGH before power is applied, forces AT Command Setup Mode. LED blinks slowly (2 seconds)
VCC	+5 Power
GND	System / Arduino Ground
TXD	Transmit Serial Data from HC-05 to Arduino Serial Receive. NOTE: 3.3V HIGH level: OK for Arduino
RXD	Receive Serial Data from Arduino Serial Transmit
STATE	Tells if connected or not

Table 4.1.4a. HC05 pinout

Baud Rate	9600 bps
Data	8 bits
Stop Bits	1 bit
Parity	None
Handshake	None
Passkey	1234
Device Name:	HC-05

Table 4.1.4b. HC05 Specification

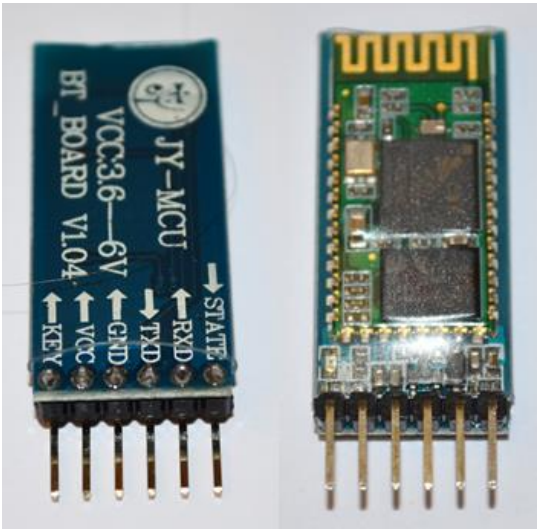


Figure 4.1.4a. An HC05 module (front and back)

4.1.5. Hall effect module

To calculate the velocity or distance, a Hall Effect Module will do. We place it on the rear axle, to be specific, close to the left wheel.

The specifications of Hall Effect Module:

- A signal output indication (LED)
- One single channel output (Low level output signal)
- Sensitivity adjustable (fine-tuning a variable resistor)
- The output switches circuit boards (MCU can be connected directly)
- Size: 20 x 32 x 11mm
- Main chip: LM393, 3144 Hall sensor
- Operating voltage: 5V DC

VCC	5V
GND	Ground
Output	Input pin of MCU

Table 4.1.5. Hall Effect Module pinout

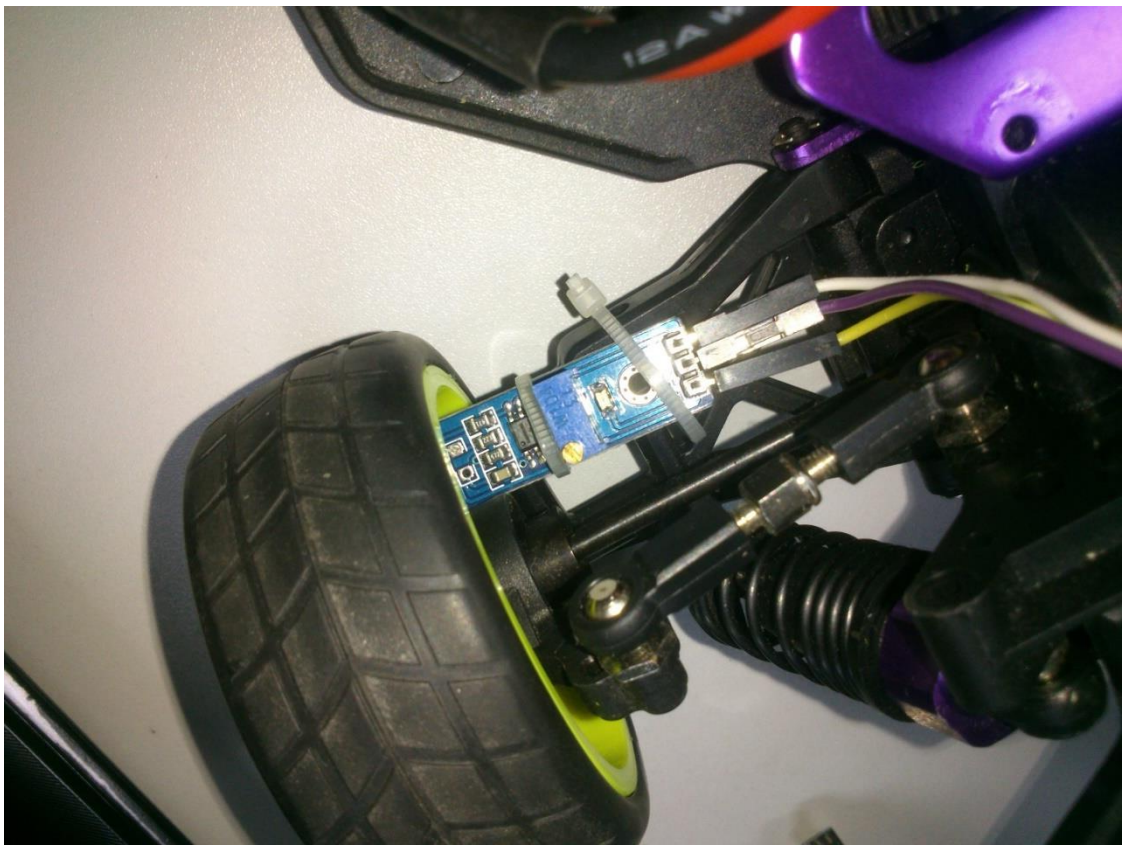


Figure 4.1.5a. A Hall Effect Module installed on rear axle

Inside this car wheel, we place 5 magnets with glue so that the car wheel and these magnets become one part. The reason we have to use 5 magnets is that the pulse is generated more often => Reduce the gap between consecutive values. We still utilize the formulae (1) in **section 2.7;** however, the difference is that the Distance has to divide by 5.

The following figures give us a closer view of the combo Hall Effect Module + Magnets.

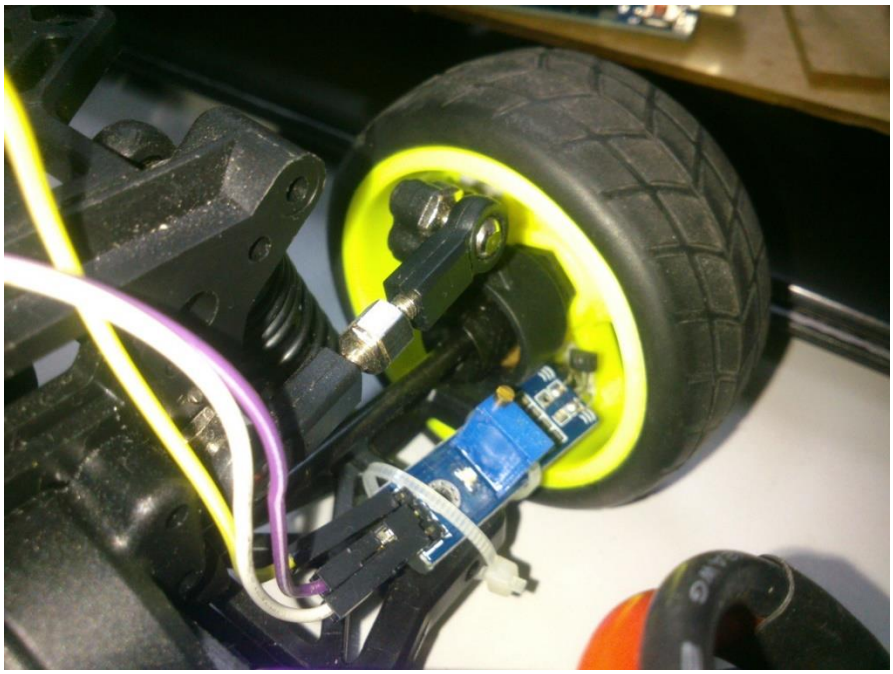


Figure 4.1.5b. **Another view of Hall Effect Implementation**

Figure 4.1.5c. **Closer view of Hall Effect Implementation**

4.1.6. Power Supply

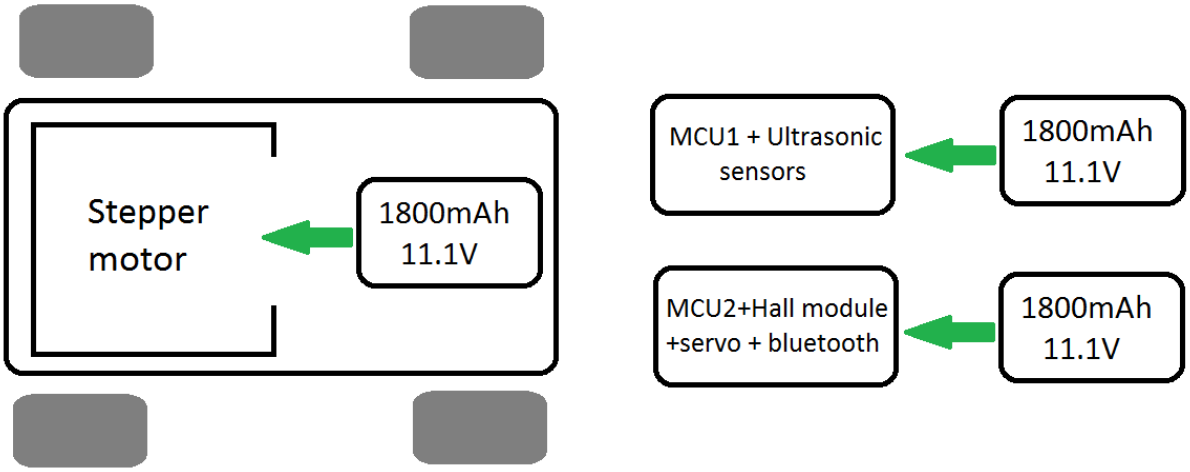


Figure 4.1.6a. Power supply for components

According to figure 4.1.6, it is necessary to have three batteries supplying the components of our system.

Figure 4.1.6b. LiPo 1800mAh/11.1V

4.1.7. Garage

For parallel parking, the size of parking spot is 40x65 cm (length x width).

For garage parking, the size of garage is 60x45 cm (length x width).

Figure 4.1.7a. Parallel parking

Figure 4.1.7b. Garage parking

4.1.8. Equipment Placement

We design a sheet made of plastic on the car. The mica plastic sheet consists of two parts which are the base and sensor stabilizers.

Sensor stabilizer

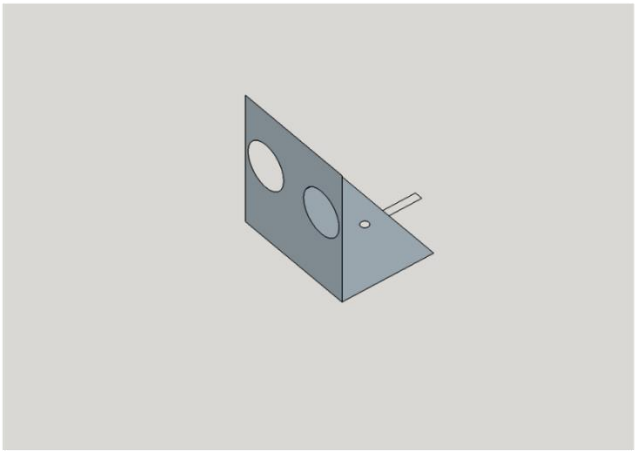


Figure 4.1.8a. 3D view of a sensor stabilizer

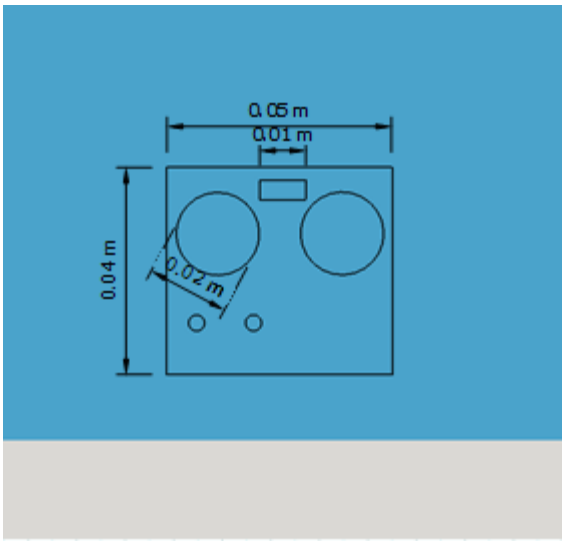


Figure 4.1.8b. Size of Sensor stabilizer

We have five “sensors stabilizer” used to install five ultrasonic sensors on the car, three sensors on the right, two sensors in car front and rear.

All sensors are placed perpendicular and longitudinal to the base, so as to calculate the distance from the car to the walls of garage.

We installed sensors onto the car base as displayed in figure 4.1.8c and we did not install them this way for nothing.

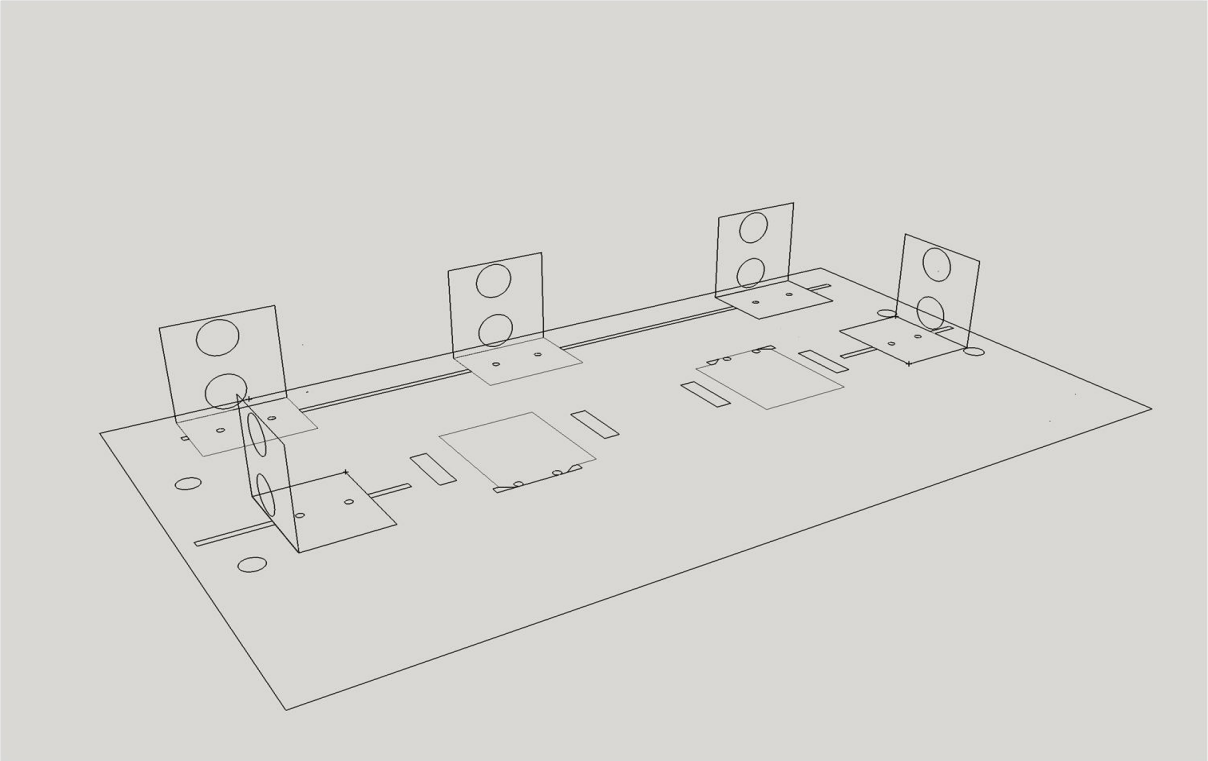


Figure 4.1.8c: 3D view of the base

The purpose is to ensure that our algorithm (parallel and garage parking) works properly. The parallel parking mainly utilizes front and rear sensors, while the garage parking depends heavily on front-right, mid-right and rear-right sensors. While the Mid-Right Sensor is installed in order to let the car quickly and properly align itself parallel to the wall when parking. The others in Front and Rear have their own meaning.

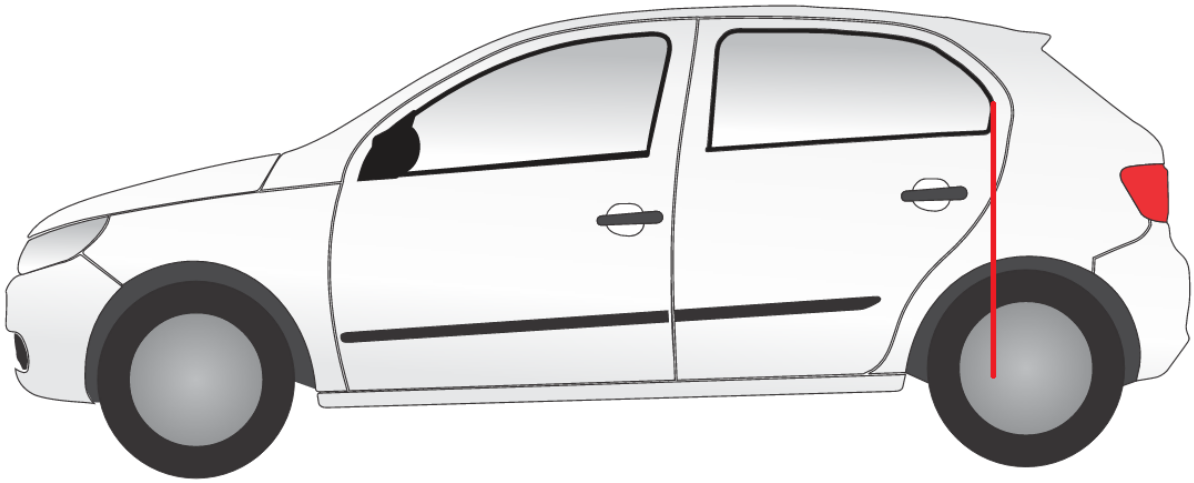


Figure 4.1.8d. Side-rear car window indicates the car rear axle.

In real-life situation, the drivers have their trick when reversing car into the parking lot. Many car manufactures implicitly designed that the side-rear car window would indicate the car rear axle. Therefore drivers can see through the furthest point of the side-rear car to see whether the car rear axle has passed the standing car or not; accordingly, they can easily drive it to the parking lot without colliding the standing car. In this model we ultilize this feature to install the rear-right sensor. The same reason why we install ultrasonic sensor in the front wheel. Furthermore, installing sensors at these points guarantee the symmetry of overall car system as well as acknowledging the moving direction of a car compared to the wall.

It took us a month to adjuct the position of sensors, more than five prototypes of sensor stabilizers and car base to reach the final version.

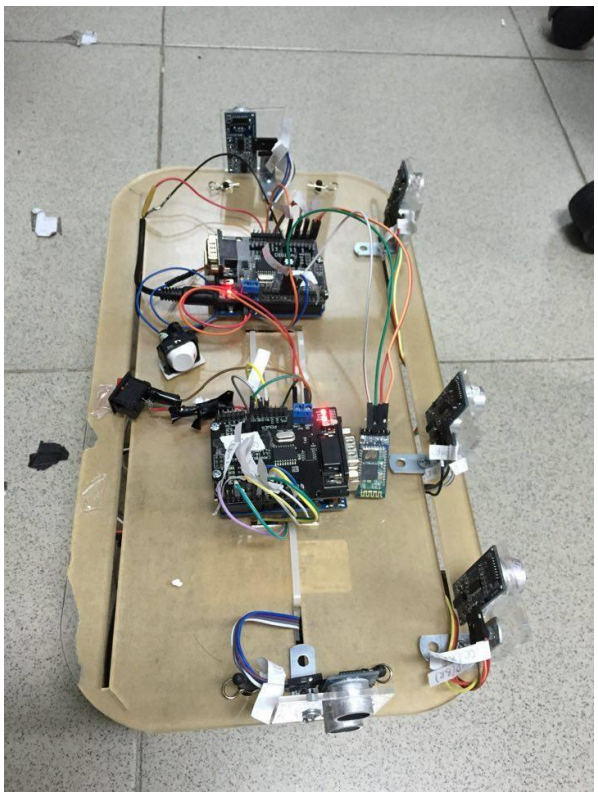


Figure 4.1.8e. Car model with assembled equipment

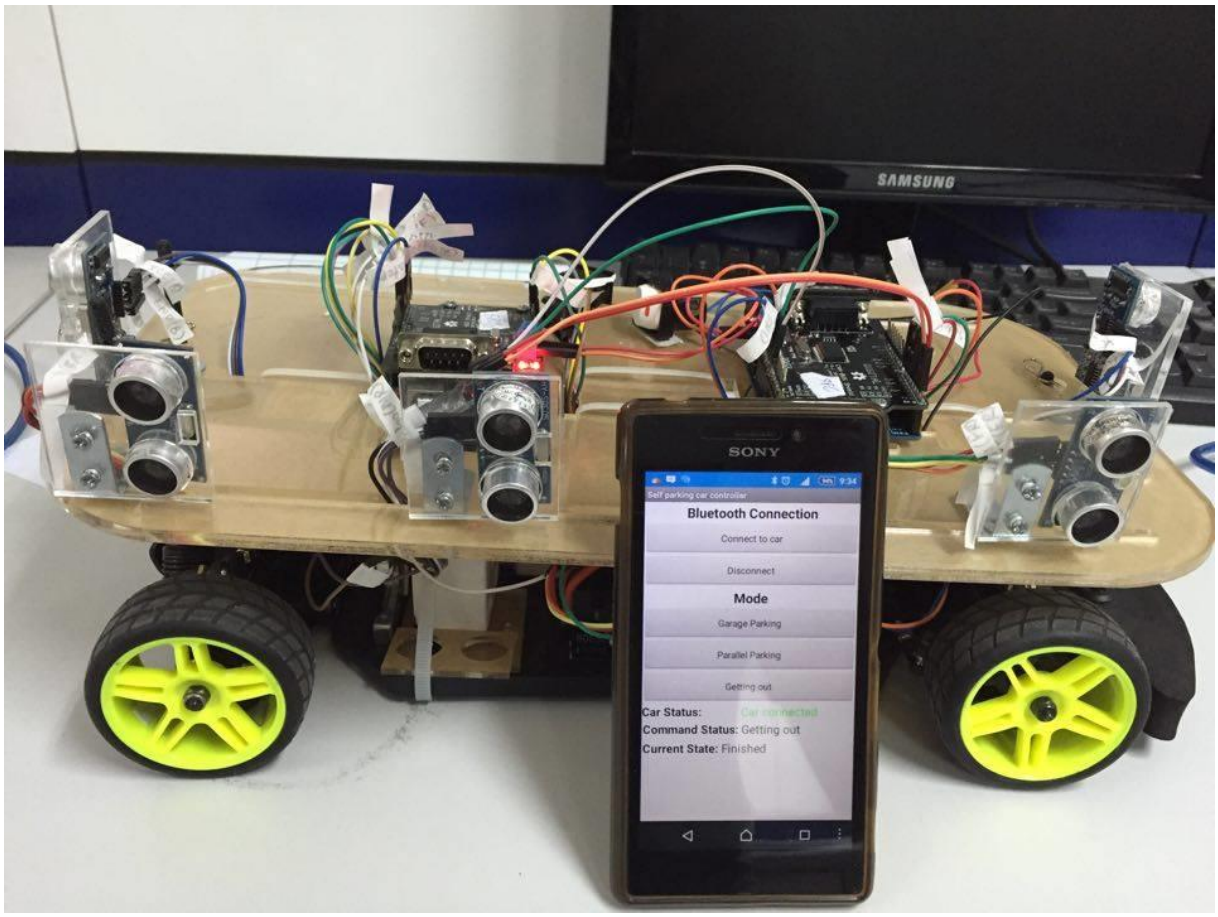
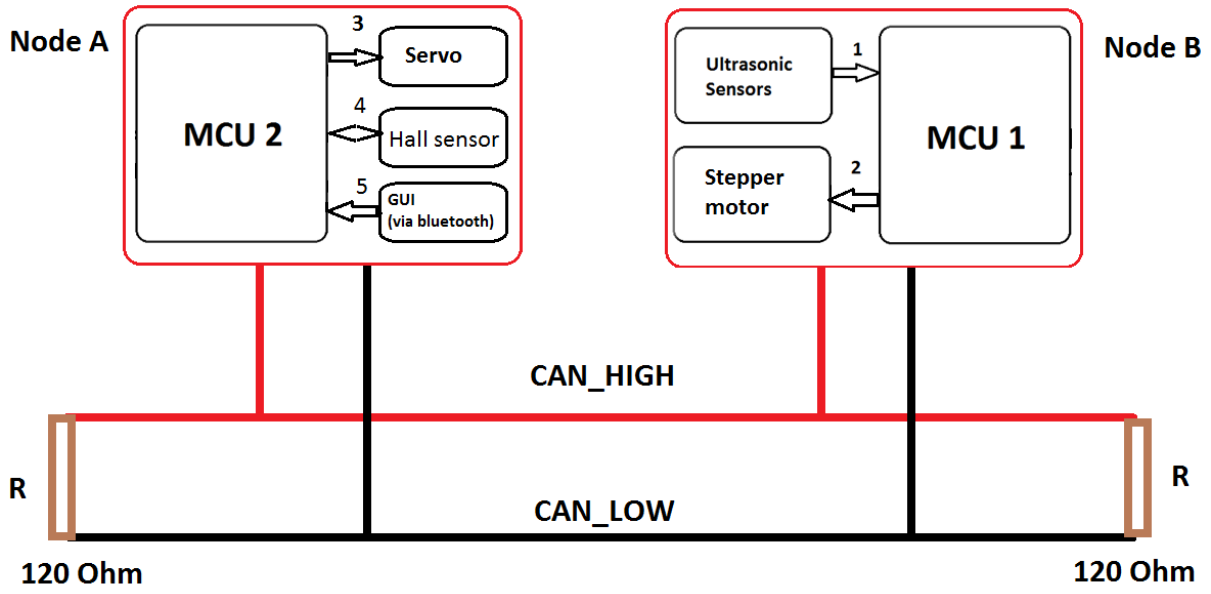


Figure 4.1.8f. Car model and Android application

4.2. Software design

4.2.1. CAN transmitting and receiving

Let's recall the System Overview illustrated by **figure 3.1a.**



From Node B sending a CAN Message to Node A, the eight data bytes of this Message is as shown:

Message (8 data bytes)							
dF	dFR	dMR	dRR	dR	0	0	0

Table 4.2.1a. 8 data bytes of a Message transmitted from Node B to Node A

Where:

- dF: Distance from the car front to the wall, for example.
- dFR: Distance from the front-right side of the car to the wall.
- dMR: Distance from the mid-right side of the car to the wall.
- dRR: Distance from the rear-right side of the car to the wall.
- dR: Distance between the car rear and the wall.

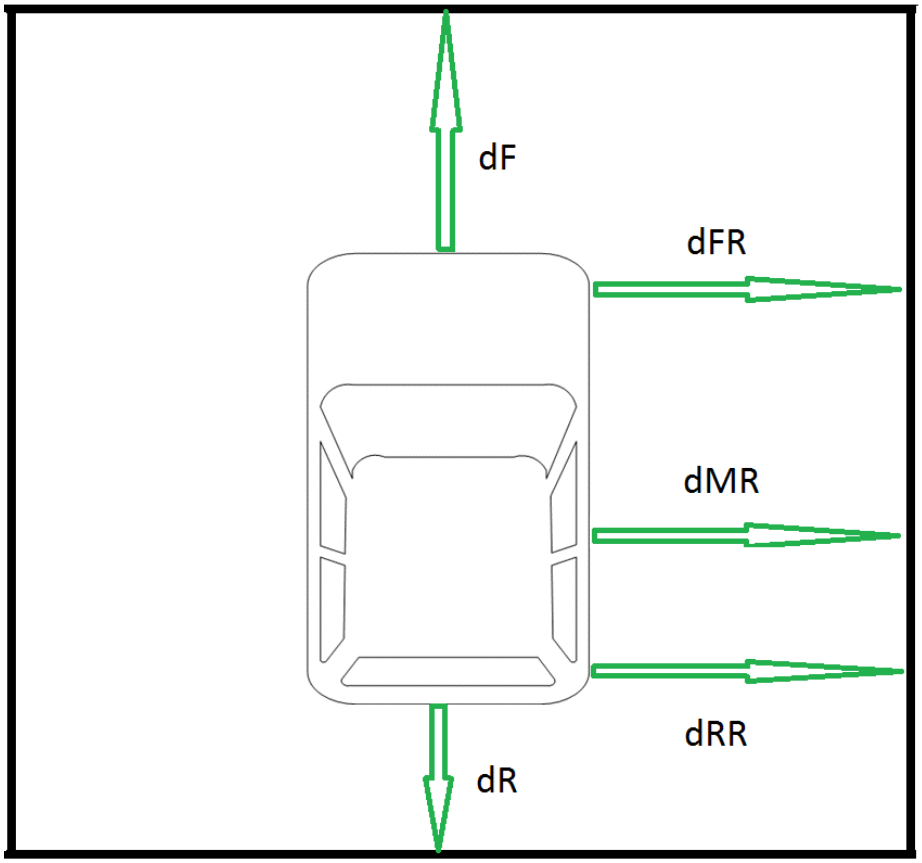


Figure 4.2.1: Distances from car to wall

Also, from Node A sending another CAN Message to Node B, its content (only consider 8 data bytes) is:

Message (8 data bytes)							
Mode	0	0	0	0	0	0	0

Table 4.2.1b. 8 data bytes of a Message transmitted from Node A to Node B

Mode: Its value varies from 0 to 8. Upon the values of Mode, the Easydriver that control the stepper motor will determine whether the car goes forward, backward or stands still.

From 0 to 3: The car goes backward, the speed decreases as the value increases.

The car stands still when the value is 4.

From 5 to 8: The car goes forward, the speed increases as the value increases.

Via CAN bus, Slave transfers a Message containing 4 distances mentioned above to Master. Accordingly, Master will yield the steering angles and speed that depends on the distances.

4.2.2. Task Scheduler

In our project, tasks are executed at a specific period of time. We need to time the execution of tasks more precisely than just having a list of functions in a loop. In order to make a system as predictable, and therefore reliable, as possible it is desirable to use a time-triggered scheduler. The idea is to use a simple scheduler that runs on microcontroller, using timer interrupt to execute tasks with microsecond precision. Because of the limitation of ATmega328P specification, it is hard for MCU to run a complicated Operating System. There implementing a simple task scheduler with time trigger method on ATmega328P which requires less CPU usage, is the best timing method to run tasks.

Task Scheduler running on this project is triggered by Timer2 channel A. The Timer2 are configured to trigger a timer interruption every 1 millisecond or we often call it 1 unit “tick”. Every tasks have a period that how often the task is executed. The ISR iterates through each task in the schedule and executes any task that is ready to run. Doing this ensures that tasks are executed with precise timing (as precise as the ATMEGA timer allows). Now we create Task Jobs that will be executed at 10 ticks or 100 ticks. Task Job is the combination of tasks that will happen on specific period of time. To execute these Task Jobs, the scheduler begins to count variables JOB_10ms or JOB_100ms to 10 ticks or 100 ticks respectively. When these variables reach 10 ticks or 100 ticks, the tasks in task job are executed. Job task at 10 ticks is higher priority than job task 100 ticks. These job tasks are represented on the timeline in the figure below:

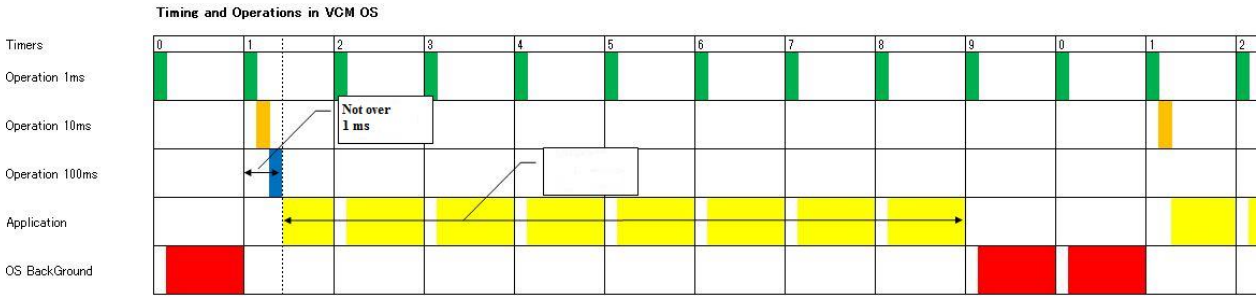


Figure 4.2.2. Task Scheduler Overview

We should be aware of the execution time of each job task. Every functions in Job task must be done in 1 tick. Therefore, the tasks in scheduler are recommended to run I/O functions which require less instruction and CPU execution time rather than calculation tasks. The calculation tasks are implemented in Application Layer in loop function.

4.2.3. Matlab Simulation

This simulation[10] serves the purpose of testing car kinematic and trajectory of parallel and garage parking processes.

We use the simulation to modify and finalize our parking algorithm before putting everything on a real car model. The simulation overview is on the next page.

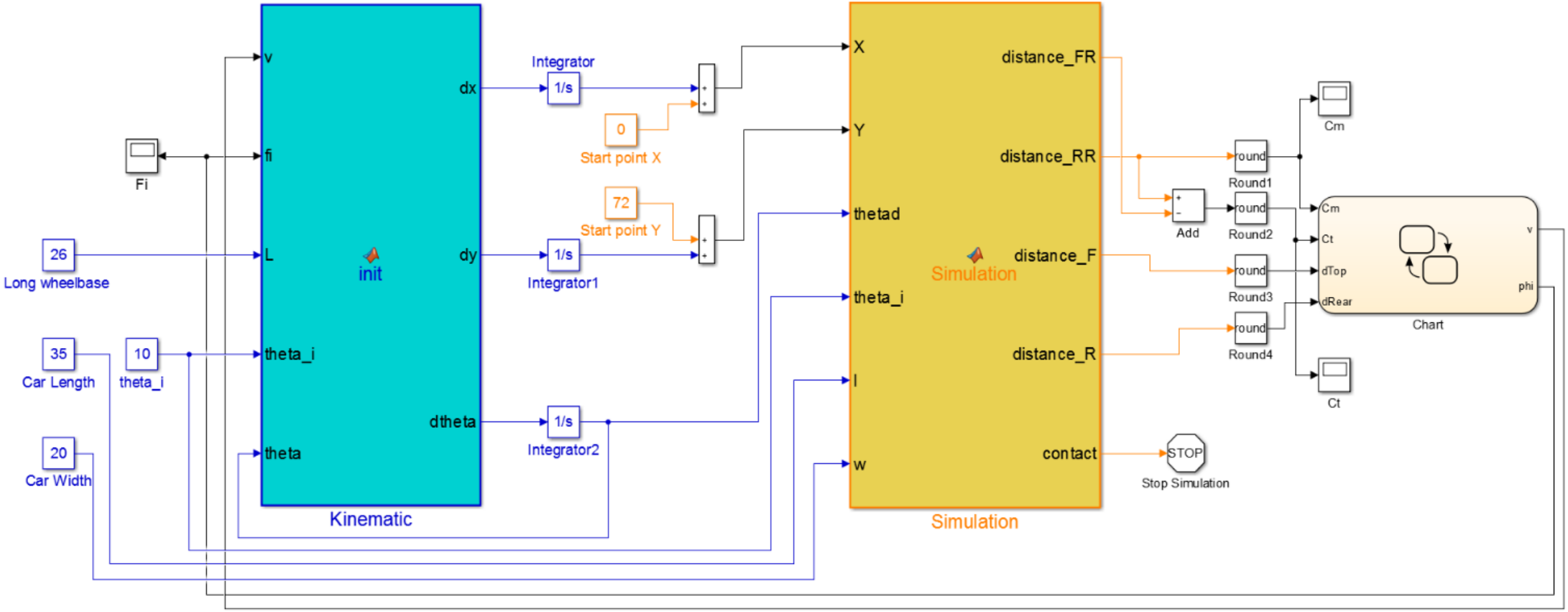


Figure 4.2.3a. Car Simulation Overview

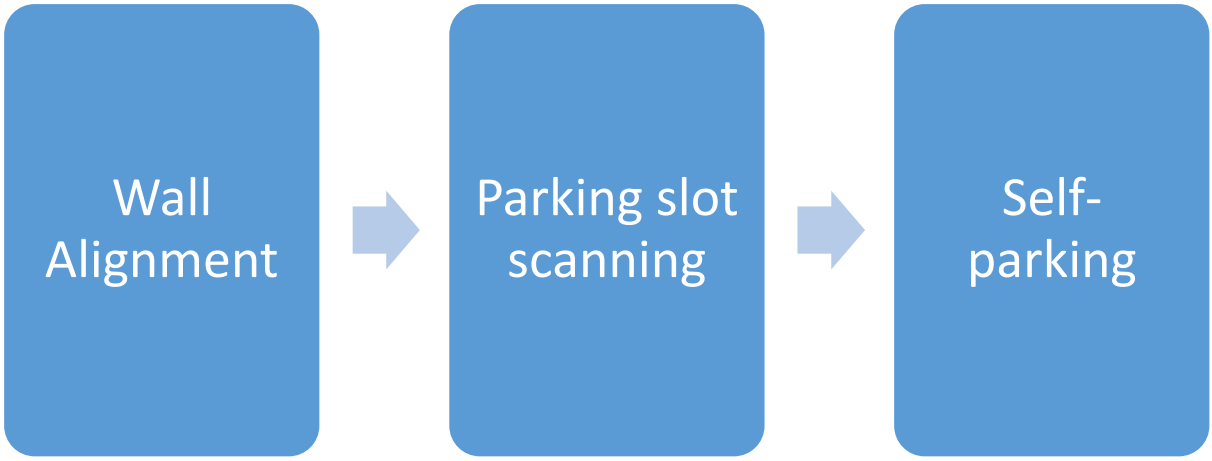
This simulation model has 3 main modules:

- Kinematic Block: this block gets velocity, car wheelbase length, initial heading angle, heading angle and steering angle as inputs. The output of this block will be the rates of change of x, y, θ. After that these values will be integrated so that we can obtain values of $, y, \theta$.

- Car Plotting Block: this blocks gets values of x, y, θ, θ_0 and dimensions of a car as inputs to plot car on simulation. The outcomes of this block are the distances from car's head, rear, right front side and right rear side to the walls. This block also carries the simulation map. This map includes the coordinates of the lines which we called them walls for our car to park on. To calculate the positions of a car, we have used many trigonometry equations to solve them and then plot car animation in figure. Furthermore, this blocks can measure the distances from the car to the wall. This function helps us simulate the operation of ultrasonic sensors when measuring distances.

- State Flow Chart: this block does the most important task in our simulation. This chart contains the finite state machines system that we depend on that states to park our car automatically. The inputs of this state flow includes the distance from car's right rear side to wall (dRearRight), distance from car's right front side to wall (dFrontRight), distance from car's front to wall (dTop) and distance from car's ear to wall (dRear). The chart are responsible to calculate steering angle using fuzzy logic and signed velocity for car direction and speed. We will introduce about this state flow chart in more details in the next part.

Let’s take a look at **figure 3.1b** again which represents the self-parking algorithm.



The following figures illustrate state flow of these three steps: Wall alignment, parking slot scanning and self-parking.

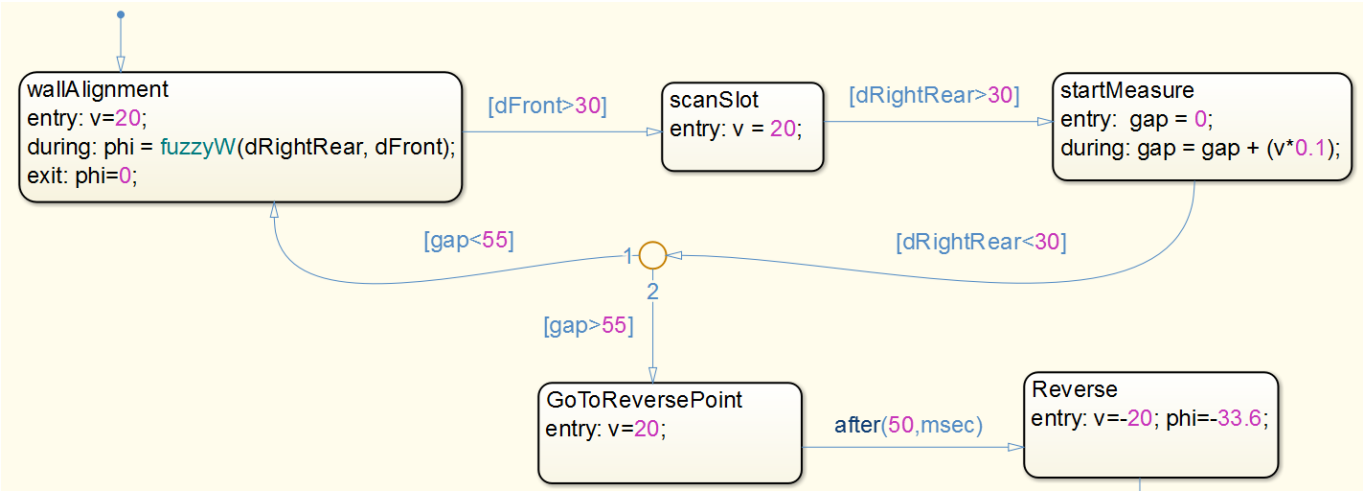


Figure 4.2.3b. State Flow of Wall Alignment and Parking Slot Scan Process

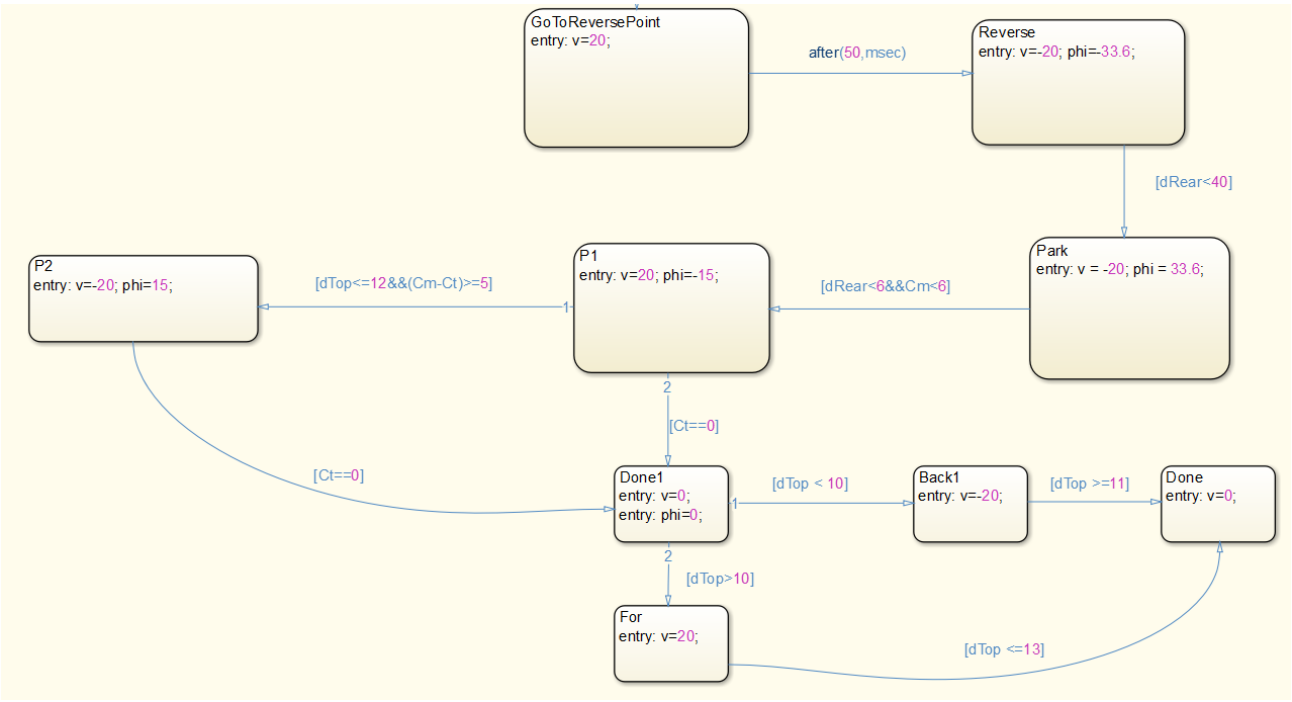


Figure 4.2.3c. Flow chart of parallel parking

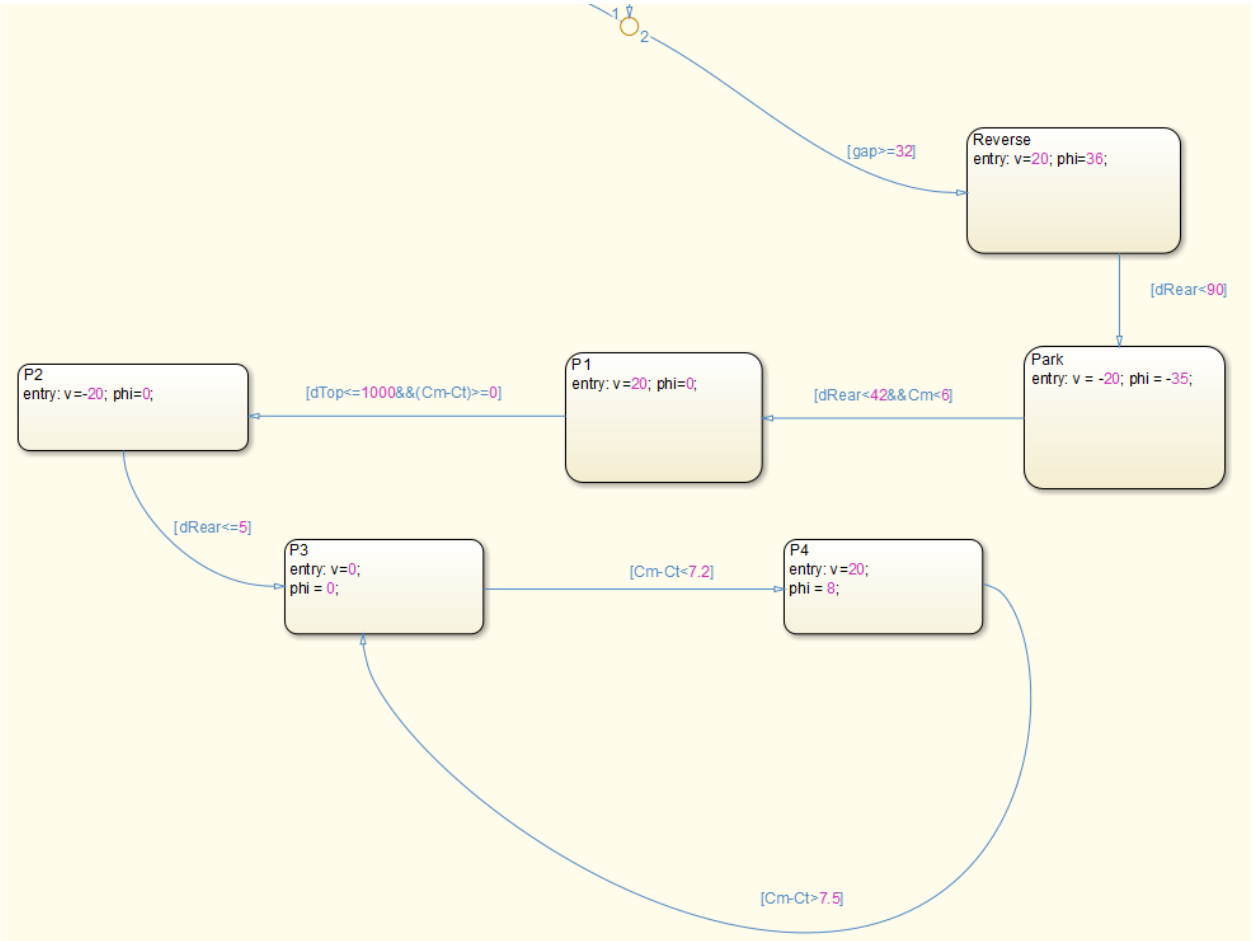


Figure 4.2.3d. Flow chart of garage parking

4.2.4. Android Application GUI

This mobile application helps users choose which parking process the car will operate, as there are two possible situations: parallel parking and garage parking. The application and car connect to each other via Bluetooth. The following figure represents the graphic user interface of the application.

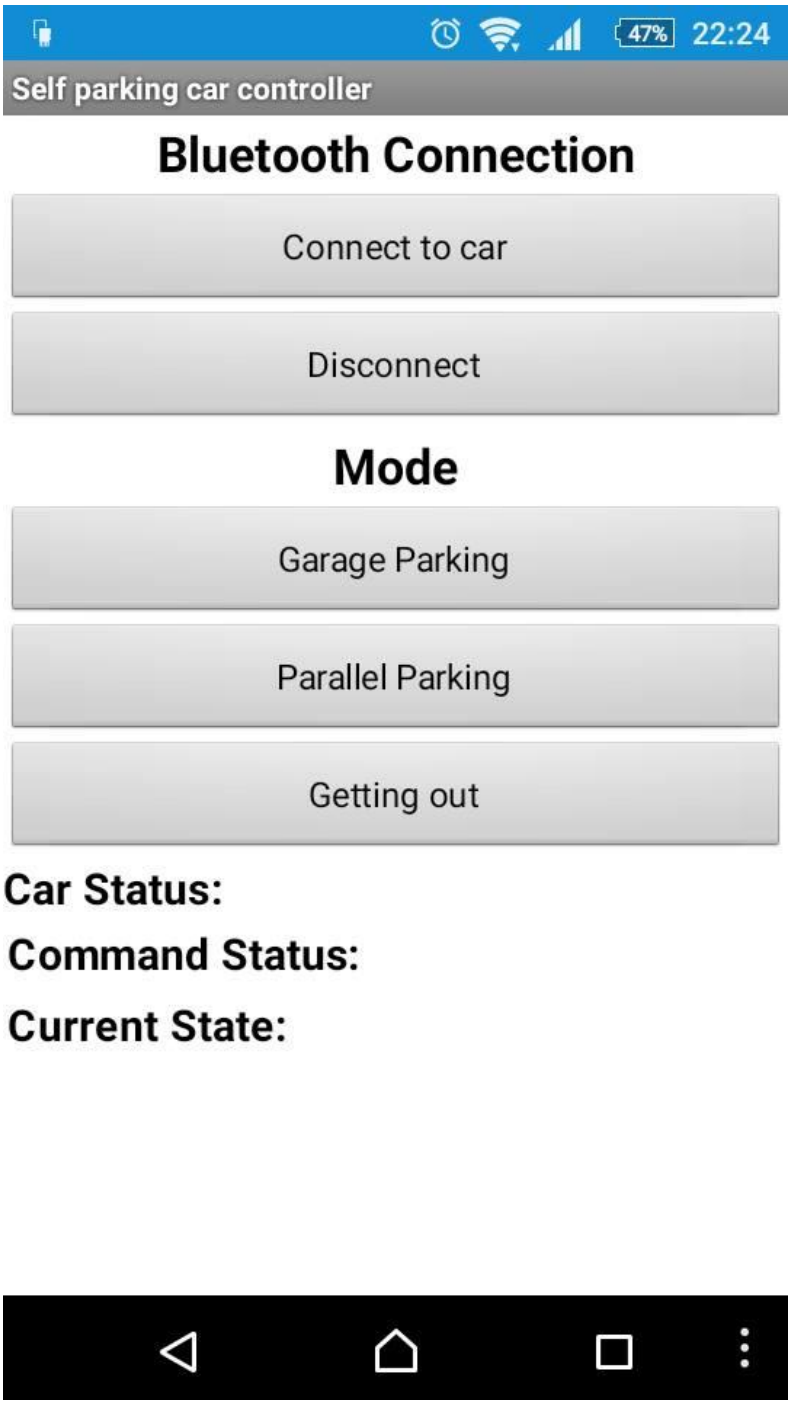


Figure 4.2.4. Android Application GUI

The application has three distinctive parts:

- Bluetooth Connection:
  - Connect to car: Press this button will display a list of Bluetooth-compatible devices, choose a desired device,i.e HC05 (Bluetooth module installed on car).
  - Disconnect: Stop connection between the application and car.
- Mode:
  - Garage parking: Press this button to have the car operate garage parking process.
  - Parallel parking: Press this button to have the car operate parallel parking process.
  - Getting out: Press this button only when the car is already inside the parking spot. The car will automatically get of the parking spot.
- Car Status Indicator:
  - Car status: Displays “Car connected” or “Car disconnected”, represent whether the car connects to application or not.
  - Command status: Displays which mode the car is operating: “Parallel parking” or “Garage parking”.
  - Current state: Either display “Processing” if the car is handling the parking process or “Finished” when the car already finished the task.

4.3. Simulation and Experimental Result

4.3.1. Simulation Result

This section represents the results of the Matlab simulation mentioned in chapter 4.2.3. The section demonstrates the effectiveness of the navigation system using fuzzy control scheme in a Matlab GUI. Here the system structures, i.e. size of vehicle robot and size of space are created based on information from the real experiment. When the vehicle passes a space that is not long enough to park, it continues moving and ignores the space. In the other hand, if the vehicle passes a reasonably long space, it will start the parking process. The threshold in this simulation is also represented Finite State Machine for Autonomous Parallel Parking or Garage Parking in the chapter above to

guarantee a collision-free trajectory. The designed trajectory generated from the chapter above is shown in figures below.

Parallel Parking Simulation

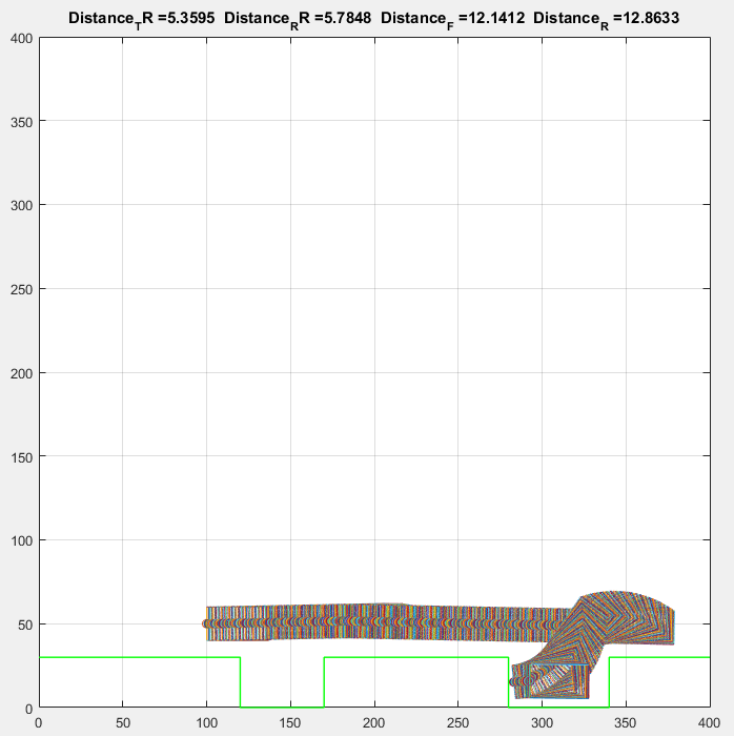


Figure 4.3.1a. Parallel Parking Full Process (including wall alignment and scanning slot)

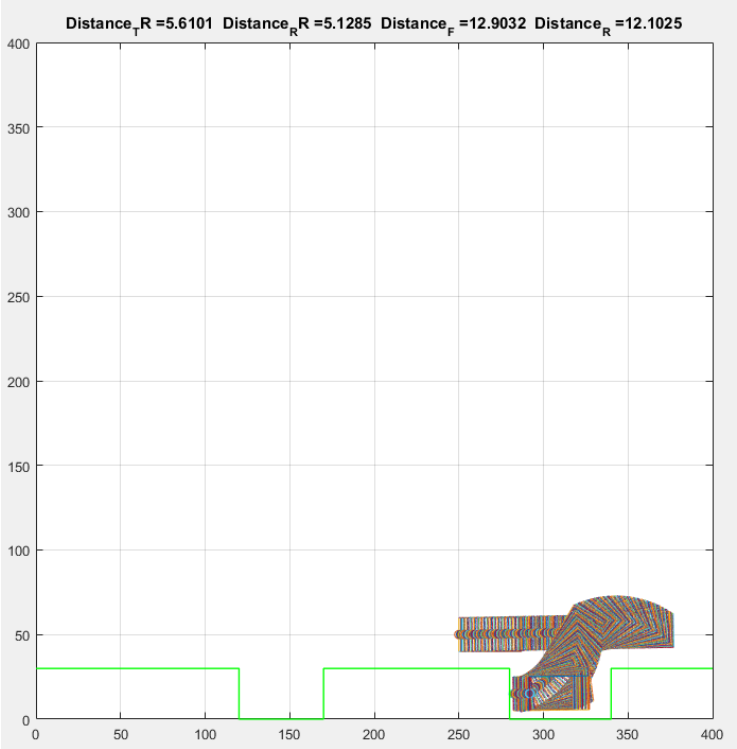


Figure 4.3.1b. Parallel Parking trajectory

Garage Parking Simulation

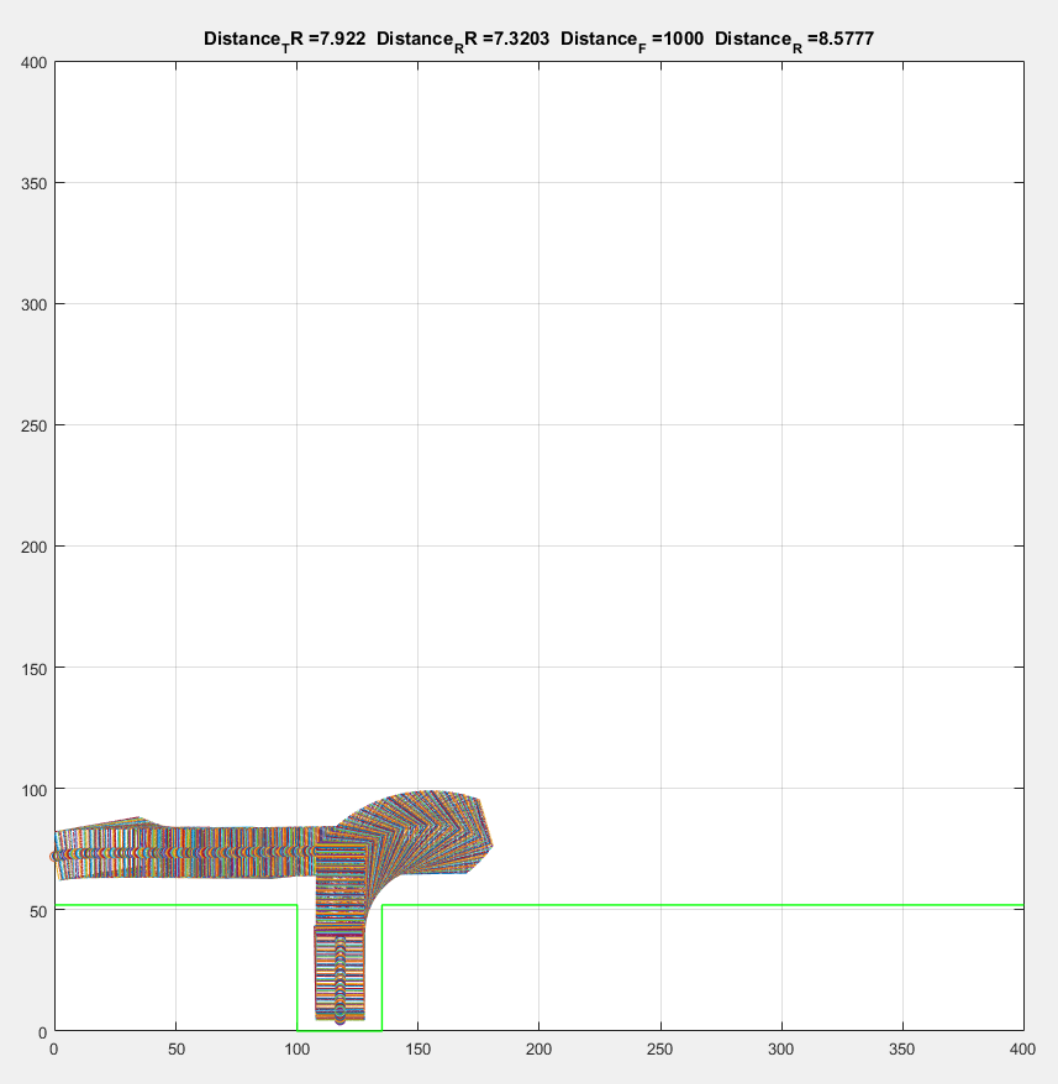


Figure 4.3.1c. Garage Parking trajectory

4.3.2. Experimental Result

The experiment has been conducted as follows: the two parking lots with a dimension of (L × W) 60 × 45 cm and 45 × 60 cm are consecutively built, also the car model with a dimension of 35× 20 cm.

Parallel Parking

Similar to the simulation result, the robot ignores the first space by moving pass and continues searching for a next one. It decides to begin parking process at the longer

space as seen. In this section, only parallel parking process is introduced and **Figure 4.3.2a** shows all the parking process step by steps.

- In step 1, the car stops at the reversal point.
- Step 2, the car reverses into the parking lot with a fixed full right direction angle.
- Step 3, the car straight the driven angle when the car's rear is 45 centimeters away the wall and keep moving back in 10 centimeters.
- Step 4, the car changes its direction angle to full left and keep moving back until nearly hit the back wall.
- Step 5, the car translates into wall alignment fuzzy logic state to align its body 10-15 centimeters along the wall.
- Step 6, the car is still in wall alignment fuzzy logic state but opposite in direction and velocity to align its body 10-15 centimeters along the wall.
- Step 7, there might be some extra state of moving backward and forward during alignment process.
- Step 8, the car stops at the required position.

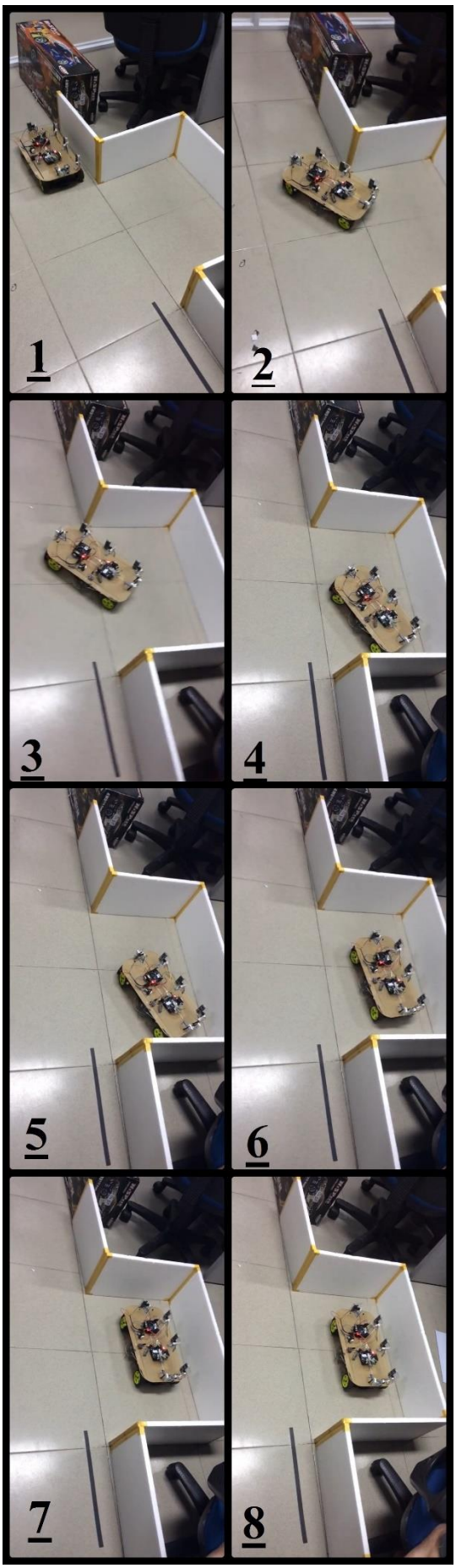


Figure 4.3.2a. Autonomous Parallel Parking Process[11]

Garage Parking

It takes a total of 6 steps to finish this task. The steps are shown in **figure 4.3.2b** and explained in detail below.

- Step 1, the car stops at the reversal point (the car rear matches the left wall of garage), then it go backward until the rear right and middle right sensors both read a distance >=60 cm and the distance it goes must be approximately equal to 30 cm.
- Step 2, the car takes a full left turn and moves away from the garage until the rear sensor returns a distance of 60 cm (or greater).
- Step 3, the car changes its direction angle to full right and keep moving backward.
- Step 4 to step 5, the car keeps moving backward with a full right direction angle, until rear right and middle right sensors return identical distances and the distance read by front-right sensor is greater than 95 cm, the car straights the wheel.
- Step 6, the direction angle is currently zero and the car moves backward to go inside of garage. The car stops when the rear sensor reads a distance less than or equal 15 cm.

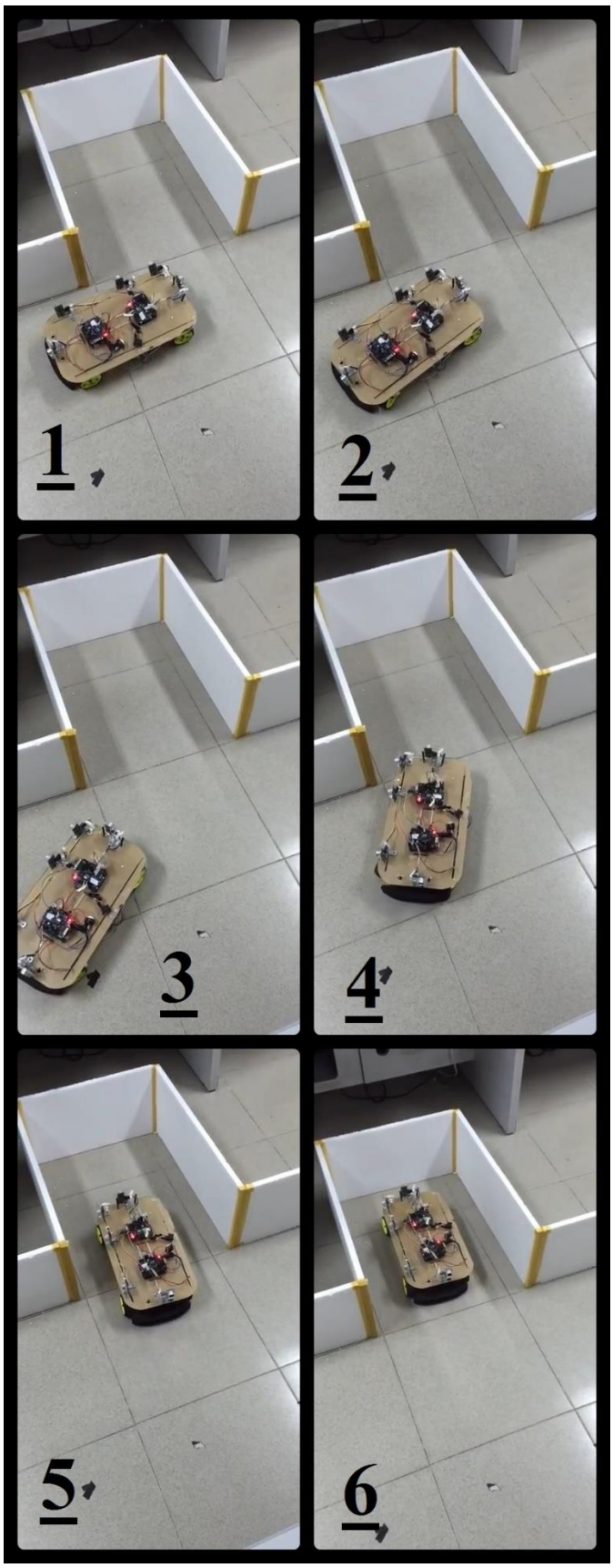


Figure 4.3.2b. Autonomous Garage Parking Process[12]

After many experiments, the car has successfully parked into the lot.

For parallel parking, the operation takes about 30 seconds. The car stays 10 – 14 centimeters away from the wall and 12 – 15 centimeter away from the front and back wall.

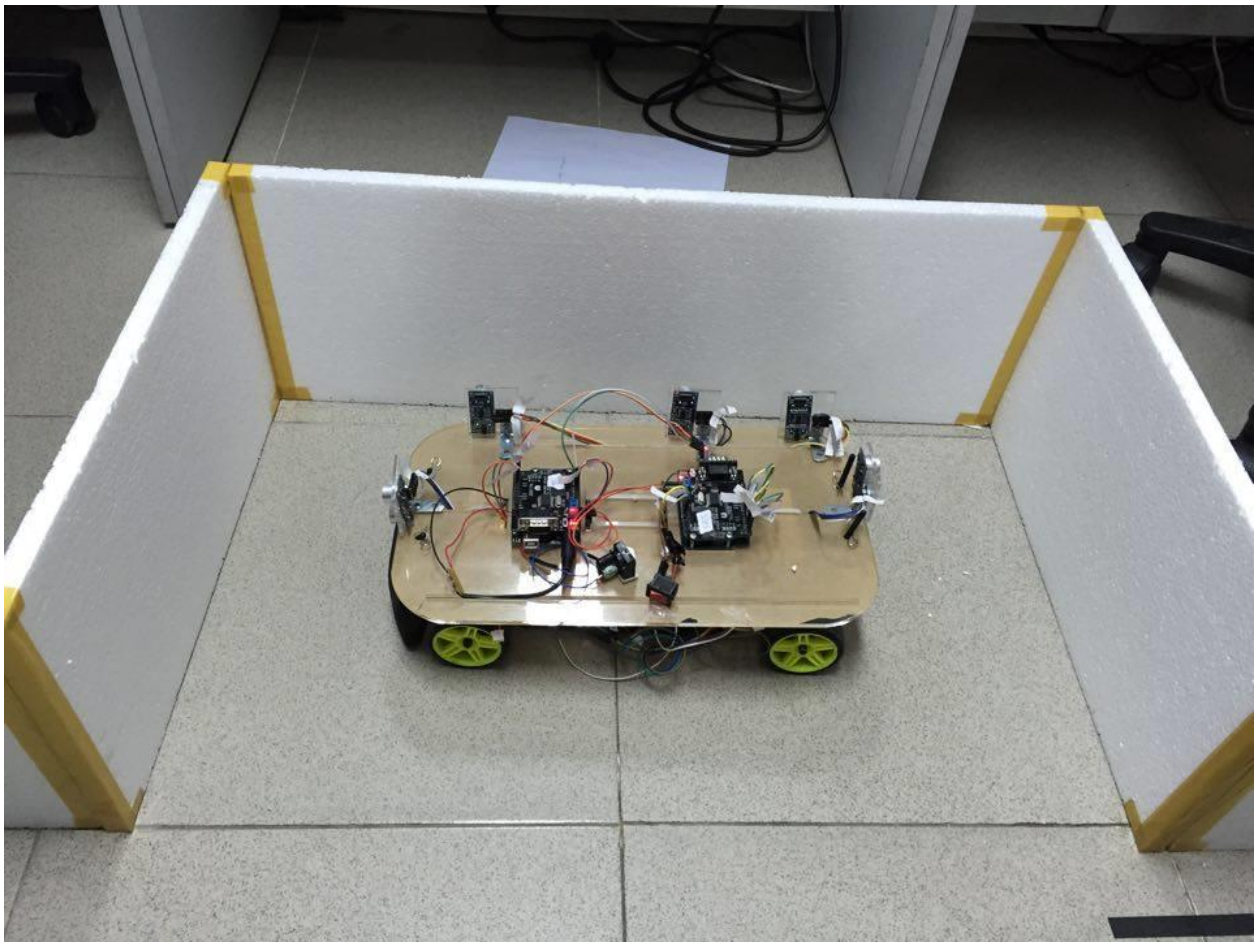


Figure 4.3.2c. Experimental Result of Parallel Parking

For garage parking, the operation takes about 25 seconds. The car also stays 10 – 14 centimeters away from the side wall and roughly 15 centimeter away from the back wall.

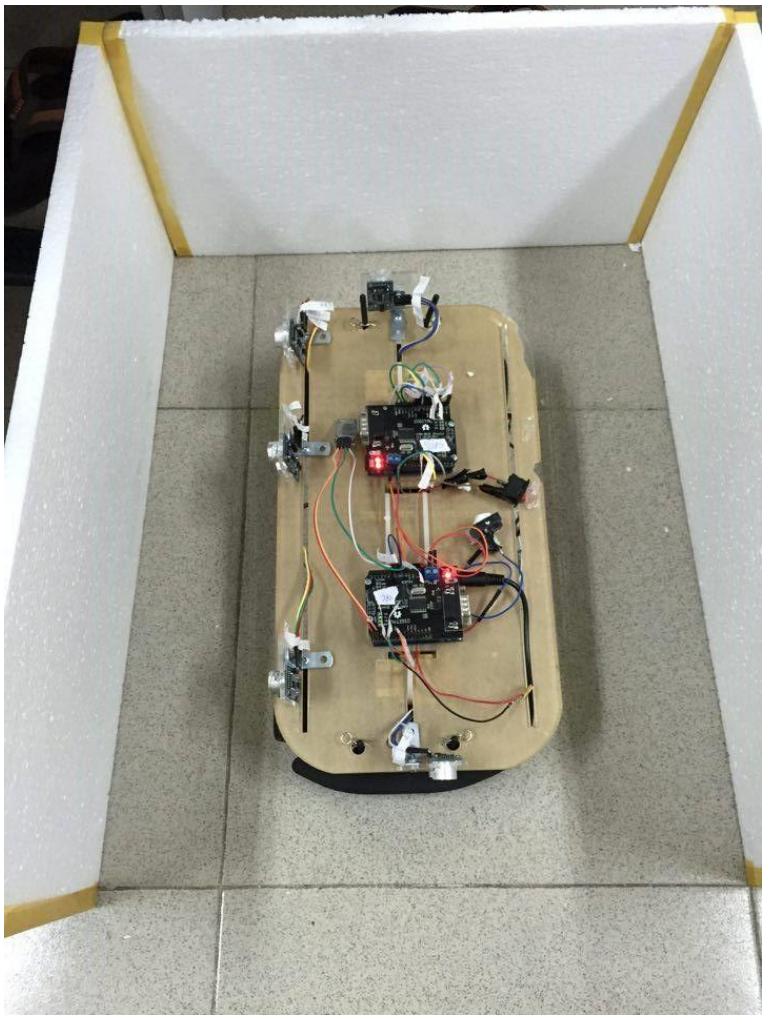


Figure 4.3.2d. Experimental Result of Garage Parking

4.4. Evaluation

		Content	**Result**
Specifications	1	The car moves slowly but very steady.	Success
	2	The car changes its direction and speed flexibly.	Success
	3	The car communicates with Bluetooth devices.	Success
	4	The user interferes during auto parking process.	On progress
	5	The measurement system (Ultrasonic & Hall encoder)	Need to improve
	6	Autonomous Parking time requirement (under 90 seconds)	Success
	7	Required position of Car in Parking lot	Success
	8	Avoiding obstacles on moving direction	Future plan
	9	Apply CAN network for communication	Success
	10	Apply Hall sensors as encoders	Success
	11	Apply Fuzzy Logic Controller	Success
	12	Task Scheduler	Success
Features	1	Autonomous Parallel Parking	Success
	2	Autonomous Car Releasing for Parallel Parking	Success
	3	Autonomous Garage Parking	Success
	4	Autonomous Car Releasing for Garage Parking	Success
	5	Reserve and park to desired slot	Future plan
	6	Mobile Android Application	Success

CONCLUSION

This study installed ultrasonic range sensors around the car-like robot successfully, and used micro control unit, servo motor, Bluetooth transmission module and other hardware to complete the outdoor automatic roadside parking system. The system uses Android, in combination with smart phone or tablet PC. The users can use a portable device to control the car-like robot. According to the simulation results of computer software and the photos of consecutive actions taken in the actual car-like robot experiment, the proposed car-like robot can execute the user's command smoothly.

When user issues command for car-like robot to perform either autonomous parallel parking or autonomous garage parking. The wall alignment fuzzy logic controller keeps the car stay 10-15 centimeters away the wall. When the reasonable parking space is determined and the ready position for parking is reached, the vehicle moves towards the parking lot with the ready-calculated trajectory. The overall automatic parking system is completed until the car-like robot enters the parking space and reaches appropriate parking position. In addition, if the automatic parking is implemented by image processing, it might fail to work normally as the outdoor environment is affected by many light sources. Therefore, this study only used ultrasonic range sensors for detection, which is not affected by light sources.

This car-like robot is operated according to the control instruction issued by the user via smart phones or tablet PCs. Differing from general control modes using heavy computer or radio-frequency circuit which may have interference of the same frequency band, the Bluetooth of smart phone is matched with the Bluetooth device of car-like robot. As the Bluetooth transmission is one-to-one transmission, there will not be external interference other than range constraint. The users can install the car-like robot application program developed by this study in intelligent devices with Android system to control car-like robot, knowing the robot's state on the screen.

For future study, to increase to accuracy of rage sensors, we should replace ultrasonic range sensors by laser scanners. Laser scanners provide more accurate measurement method than ultrasonic and they are less affected by noise. Furthermore, Laser Scanner provide more information from environment such as the angle from car to obstacles.

Therefore, despite of their high cost, Laser Scanners are best solution for autonomous self-parking car.

BIBLIOGRAPHY

[1] J.P. Laumond, "Nonholonomic Motion Planning for Mobile Robots", Tutorial Notes, May 1998.

[2] MathWorks, "Foundation of Fuzzy logic", MATLAB documentation, Version 2016a, Web, http://www.mathworks.com/help/fuzzy/foundations-of-fuzzy-logic.html

[3] CAN BUS Wikipedia, Web, https://en.wikipedia.org/wiki/CAN_bus#Standards

[4] Robert BOSCH GmbH, "CAN Specification", Version 2, datasheet.

[5] Henk Boterenbrood, CAN Message Format, Web, http://www.nikhef.nl/pub/departments/ct/po/doc/gpcan-html/node8.html

[6] Zadachyn, Viktor, and Oleksandr Dorokhov. "CALCULATION OF OPTIMAL PATH FOR PARALLEL CAR PARKING", Transport and Telecommunication, 2012, Volume 13, No 4, 303–309.

[7] Necip Gürler , Fırat Yılmaz Cevher , Alpaslan Yufka , and Osman Parlaktuna. "Direct-Motion Parallel Parking for a Vehicle with and without a Trailer", International Symposium on Innovations in Intelligent Systems and Applications, 21-24 June 2010, Kayseri & Cappadocia, TURKEY.

[8] Robot-electronic, SRF05 Technical Specification, Web, http://www.robot-electronics.co.uk/htm/srf05tech.htm

[9] SeeedStudio, SeeedWiki, Product Web, http://www.seeedstudio.com/wiki/CAN-BUS_Shield

[10] MathWorks, "Simulink Documentation", MATLAB documentation, Version 2016a, Web, http://www.mathworks.com/help/simulink/

[11] Le Thai Xuong, (2016, June 8), "Parallel", retrieved from https://www.youtube.com/watch?v=kSnWTgdeqx4

[12] Le Thai Xuong, (2016, June 8), "Garage", retrieved from https://www.youtube.com/watch?v=3OXdseULVs0