

DESIGN OF IMPROVED GRID FOR TURTLE ROBOT

by

Muhammad Zulhilmi Bin Adnan
(Supervisor: Abu Bakar Sayuti Bin Hj Mohd Saman)

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronic Engineering)

SEPTEMBER 2013

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

DESIGN OF IMPROVED GRID FOR TURTLE ROBOT

by

Muhammad Zulhilmi Bin Adnan

A project dissertation submitted to the
Department of Electrical and Electronic Engineering
in Partial Fulfillment of the Requirement
for the Degree Bachelor of Engineering (Hons)
Electrical and Electronic Engineering

Approved by

(Abu Bakar Sayuti Bin Haji Mohd Saman)

UNIVERSITI TEKNOLOGI PETRONAS

TRONOH, PERAK

September 2013

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own concept as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

(MUHAMMAD ZULHILMI BIN ADNAN)

ABSTRACT

Turtle robot is known as a two-wheel mobile robot which can be programmed to do various task such as line following, grid following, wall or obstacle detection and the list goes on. Common turtle robot has the capability of performing line following function only. Another turtle robot's capability is to do grid following, but this function is limited to the designed grid which is 90 degree grid. The objectives of this project are to design an improved grid including new algorithm that suitable for the use with a turtle robot which allows more than straight line and 90 degree grid. The method used in this project is by exploring several implementation of turtle robot with its grid design. This is the followed by developing improved grid design with new algorithm which is then tested continuously to ensure its functionality working flawlessly. As the result, turtle robot now has the capability to follow 45 and 135 as well as 90 degree grid following. In order to follow the grid designed, it must be preprogrammed with only its orientation and the coordinate. It is recommended to improve the algorithm of this turtle robot so that in can follow grid with less mistakes and to achieve its ultimate goal which is for learning purposes.

ACKNOWLEDGEMENT

First and foremost I would like to express my humble gratitude to Allah SWT for His blessings and guidance; I am able to finish my Final Year Project. I would like to sincerely thank to Universiti Teknologi PETRONAS (UTP) especially Electrical and Electronic Engineering Department for the best services, essential support and facilities in completing the project

Special thanks to my Final Year Project supervisor, Mr. Abu Bakar Sayuti Bin Hj Mohd Saman for his best and sincere guidance, ideas and teaching to make it possible for to finish the project accordingly. His support, motivation and willingness to share his knowledge and expertise have given a very important experience during the development of this project.

I would also like to thank UTP staff and Lab Assistants that helped me to complete the project especially in providing tools and equipment as well as the component needed to build the project. I really appreciate their great help and information sharing for me to obtain best solution possible for small problems happened. There is no other staff that has their fondness.

Last but not least, deepest gratitude to my beloved family and friends in UTP for their continuous support and encouragement which enabled me to do my best for this project. I hope after the completion of this project, all the findings and knowledge that I shared through this report can be useful for everyone for study and personal research work. Thank you and May Allah bless all of you.

<u>NO</u>	<u>TABLE OF CONTENT</u>	<u>PAGE</u>
	CERTIFICATION OF APPROVAL	ii
	CERTIFICATION OF ORIGINALITY	iii
	ABSTRACT	iv
	ACKNOWLEDGEMENT	v
	Table of Content	vi
	List of Figures	vii
	List of Tables	viii
1	INTRODUCTION	
1.1	Project Background	1
1.2	Problem Statement	2
1.3	Problem Identification	3
1.4	Significant of the Project	3
1.5	Objectives and Scope of Study	3
1.6	The Relevancy of the Project	4
1.7	Feasibility of the Project within the Scope and Time Frame	4
2	LITERATURE REVIEW	
2.1	PIC Microcontroller variation and functionality	5
2.2	Finite State Machine	6
2.3	Algorithmic State Machine Chart	7
2.4	Pseudo Code and C code programming structure	7
2.5	Turtle Robot components and diagram	8
2.6	Grid solution	9
3	METHODOLOGY	
3.1	Research Method	11
3.2	Project Activities	12
3.3	Tools and Equipment	19
3.4	Key Milestone	20
3.5	Gantt chart	20
4	RESULT AND DISCUSSION	
4.1	Data Gathering and Analysis	21
	- Pseudo Code and Finite State Machine	
	- Algorithmic State Machine Chart and Calculation	
4.2	Experimentation and Modeling	28
	Demonstration on YouTube	
4.3	Prototype	30
	- Several route test on grid	
5	CONCLUSION & RECOMMENDATION	35
6	REFERENCES	37
7	APPENDICES	38

List of Figures

Figure 1: Some of PIC Microcontroller Chip Configuration	5
Figure 2: Example of Simple Finite State Machine Task Flow Chart	6
Figure 3: Some of the example of Pseudo code	7
Figure 4: PR23-R2 - Multifunction Mobile Robot Component	8
Figure 5: Ring structure of grid solving solution	10
Figure 6: Research method integrated with project activities flow chart	11
Figure 7: Project activities flow chart	12
Figure 8: Algorithmic State Machine chart for the Line-Following code	14
Figure 9: First solution adding 45 degree grid to 90 degree grid	15
Figure 10: Second solution adding 45 degree grid to 90 degree grid	16
Figure 11: Algorithmic State Machine chart for 90 degree turn	18
Figure 12: Line and Grid design stages	21
Figure 13: Line width according to the sensors position	21
Figure 14: 90 degree grid design with coordinate	22
Figure 15: Currently confirmed grid design	23
Figure 16: Program code/Algorithm development flowchart	24
Figure 17: Experimenting line-following function	29
Figure 18: Experimenting grid following function	29
Figure 19: Line-following Mat	30
Figure 20: 45, 90 and 135 degree Grid Design	30
Figure 21: The 1st tested pattern/route	32
Figure 22: The 2nd tested pattern/route	34

List of Tables

Table 1: Basic State table of the Sensors detection	17
Table 2: List of Tools and Equipment used	19
Table 3: Delay allocation for each sub-function	27
Table 4: Delay calculation	28

INTRODUCTION

1.1 Project Background

Robotics is a branch of technology world. It deals with almost the entire things related to design, construction, operation and application of robots. These also include computer systems for their control, sensory feedback, and information processing. Since robotics has become main part in our daily life, it is now expanding quickly as the research goes on. The goes the same for the community that is now trying to make robotics as a hobby to invent new things and simplify their everyday lives.

There are several main component of robotics which is power source, actuation, sensing, manipulation, locomotion and environmental interaction and navigation. This project is more related to these main components which are sensing, locomotion and navigation. After all it is a turtle robot which is moving (locomotion) by following line (sensing) and navigate itself throughout the line or grid given (navigation). Thus, these main parts need to be developed accordingly achieve certain functionality.

These three main parts are very much related to the mobile robot. On the other hand, mobile robot now is in deep research for its autonomous function used in order to move from a place to another place. This interesting part of robotics attracts hobbyist to try and develop several mobile robot that function according to several environment in a small size or scale. As time goes by, this type of robot had attracted teachers and academician to teach and make programming become more interesting.

Turtle robot or on the other name is mobile robot is a two-wheel mobile robot which can be programmed to drive its movement on a plane. Normally, this mobile robot is pre-programmed by using specific tools and devices so that the robot will move according the desired direction and movement. On the other hand, there are several purposes of this mobile robot. First, it is indeed a fun activity for any hobbyists whom are interested to the things related to robot.

Second, this mobile robot can be used as educational tool to teach programming to young children. There are other purposes such as this robot is being used as the simulation of a space-vehicle on a conditional space and many more. In this project, we are more focus on the use of this mobile robot as educational tool in teaching programming. We are going to improvise the capability of the normal mobile (turtle) robot that are usually used in educational program

This project will improvised the grid used by the turtle robot with some added capability that will be explained afterwards. Since this mobile robot will be used as an educational tool for teaching, algorithm of this robot will be enhanced because new ability has been added to the mobile robot. Both improvised grid and programming will produced better toolkit for teaching young children about programming.

Kids nowadays are exposed to the advance technology around them, and soon things that was previously complicated such as programming will have easier approach, for example in this case is to teach the young children. Furthermore, with the existent of mobile robot as their toys will attract them to learn more about programming. This learning experience is a much better because it has hands-on practical as well as training the logic behind the programming.

1.2 Problem Statement

The movement's accuracy of the robot can be increase in moving from certain direction to another direction. Grid is chosen in order to improve the accuracy of the movements. On the other hand, we have only 90 degree grid and line following capability. There are some parts of the turtle robot need to be modified in order to change the 90 degree limit. These modifications include the design of the grid that will be used for the movement and navigation of the turtle robot.

1.3 Problem Identification

In this project, we are going to change the turtle robot limit which is 90 degree only turn on a designed grid by adding capability to perform 45 degree turn. Therefore, there are few things related to the turtle robot that need to be modified. The grid requires improvement in order to perform 45 degree turn as well as the programming part of the turtle robot. Somehow, other physical parts of the turtle robot such as line sensors need also significant modification.

1.4 Significant of the Project

This project has at least some significant benefit to the young children and the researcher. The main purpose of the turtle robot in this project is for educational toolkit. Improvising the turtle robot will stimulate young children to learn new logic behind of each operation of the turtle robot. This will then attract the interest of the young children to learn how to program and try a lot more combination of movement of the turtle robot.

Turtle robot usually can be used as simulation for researcher. For example, simulation of new road system which is the grid simulated as the road and the turtle robot as the road user's vehicle. Provided with sufficient technology, user's vehicle may be automatically driven using artificial intelligent and Global Positioning System (GPS). Another example is that mobile robot can be used to access any unachievable space or condition by normal human being such as outer space or deep-sea.

1.5 Objectives and Scope of Study

The objectives of this project are as stated below:

1. To design an improved grid suitable for use with a turtle robot that allows more than straight line and 90 degree turn.
2. To develop a new algorithm for a turtle robot to use on the new grid

1.6 The Relevancy of the Project

This project is relevant in mostly in term of advanced learning and research. Nowadays, complicated things can be simplified and explained in many ways with the help of a lot of learning tools such as presentation slide, model or prototype, thinking skill tools and the list goes on. Turtle robot project is by right may attract and help young children to learn programming and the logic behind it. Improvising the turtle robot may enhanced the children learning in which they can learn more about programming and not just as previously which is limited with simple programming instruction.

On the other hand, there is much to research in term of preprogrammed or autonomous robot. It is believe that the percentage of human physical work regarding heavy or repetitive works which is being handled by the robot is increasing. Thus, a significant advance in robot research may in several ways improvise the life and productivity of human being.

1.7 Feasibility of the Project within the Scope and Time Frame

From the scope and time frame given, this project is feasible to be finished within the range given. It will be two semesters which are equivalent to 28 weeks to do the entire necessary thing related to the project. This project is actually an ongoing project. Thus, it has many source and material available whether in the internet or information resource center.

LITERATURE REVIEW

In our project, we would like to use PIC Microcontroller because of its price, function and user-friendly type. There are several PICs family which is PIC10FXXX, PIC12CXXX/PIC12FXXX, PIC16C5X, PIC16CXXX, PIC17CXXX and PIC18CXXX. These PICs family has their own special function and capabilities. Those numbers represent bits used in each microcontroller. Some characters in their name explain it specific function and revision (Dogan, 2006).

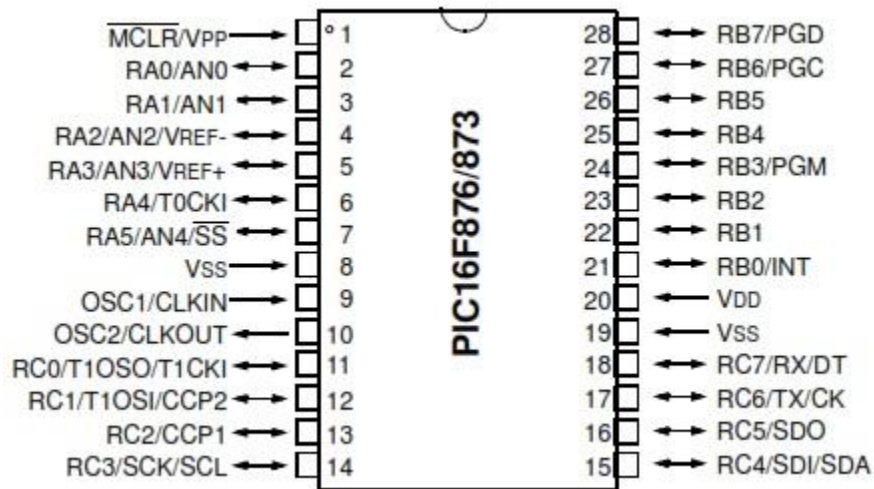
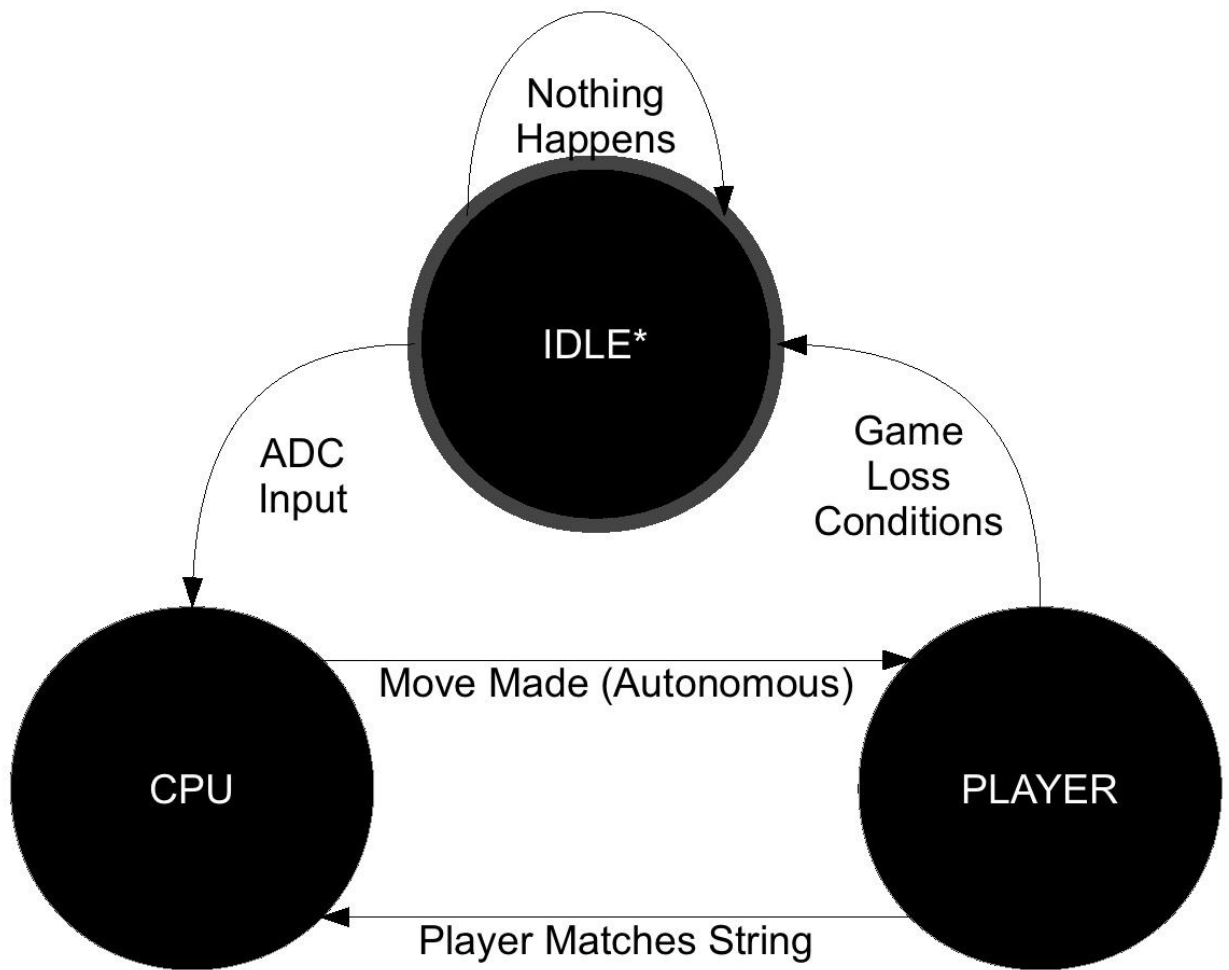


Figure 1: Some of PIC Microcontroller Chip Configuration

Single-board computer is much more complex in which it has its own operating system that can be used as a programming platform to control the turtle robot accordingly. Basically, this type of pre-programming feature is based on the fixed event that we want the turtle robot to operate accordingly. It is called an event-based programming which is basically pre-programming the turtle robot based on the feedback event that happened currently and next event which will be done by the robot.

This flow of programming is further explained in term of microcontroller tasking and operations. Basically there are several concept used by microcontroller in implementing the control algorithm which are state machine and real time operating system (RTOS). Simple constructs used to perform several activities in sequence is called state machine. Real time operating system is multi-tasking kernel which controls the allocation of period of time for each task before it stopped and replaced by another task (Dogan, 2008). In this project we would like to use state machine which simpler than RTOS.



*: Starting State

Figure 2 Example of Simple Finite State Machine Task Flow Chart

As we relate to the programming, turtle robot required stimuli or an event that can bring feedback so that next operation can be conducted. Our turtle robot will need a designed grid as a guided path to done any operation. This design has its own specification in which calibrated with the feedback devices that will be used in deciding the next event of the turtle robot

Finite State Machine (FSM) is then converted into C programming language by applying certain method which by using Algorithmic State Machine (ASM) flow chart. ASM flow chart uses more detail and complex approach than FSM. However, ASM chart states clearly all the logic or algorithm of the functionality of the turtle robot. Then the C source code is coded accordingly to the logic of the flow chart.

This C source code is the main thing that will programmed to turtle robot to function as its capability. There is the other type of language which purposely used for human reading without understanding the lower level of the C codes. This is called as pseudo code. This is a structural notation for programming that uses verbal description and informal programming structure to explain the codes.

```
task main()  
{  
  while ( touch sensor is not pressed )  
  {  
    Robot runs forward  
  
    if (sonar detects object < 20in away)  
    {  
      Robot stops  
      Robot turns right  
    }  
  }  
}
```

Some intact syntax
The use of a while loop in the pseudocode is fitting because the way we read a while loop is very similar to the manner in which it is used in the program.

Descriptions
There are no actual motor commands in this section of the code, but the pseudocode suggests where the commands belong and what they need to accomplish.

Figure 3: Some of the example of Pseudo code

Figure above shows clearly how C codes in its early of programming it. The same structure of C programming is used, verbal syntax is put first to explain the logic operation while we are reading and programming it simultaneously. This process is followed by putting the actual definition by the hardware in each part of the pseudo code

is presented. The hardware part is also a main important part of this project. We use PR23 DIY Mobile Robot manufactured by Cytron Technology. The mobile robot is controlled by using microcontroller with some feedback by the Infrared sensors. Below is the detail description of the turtle robot:

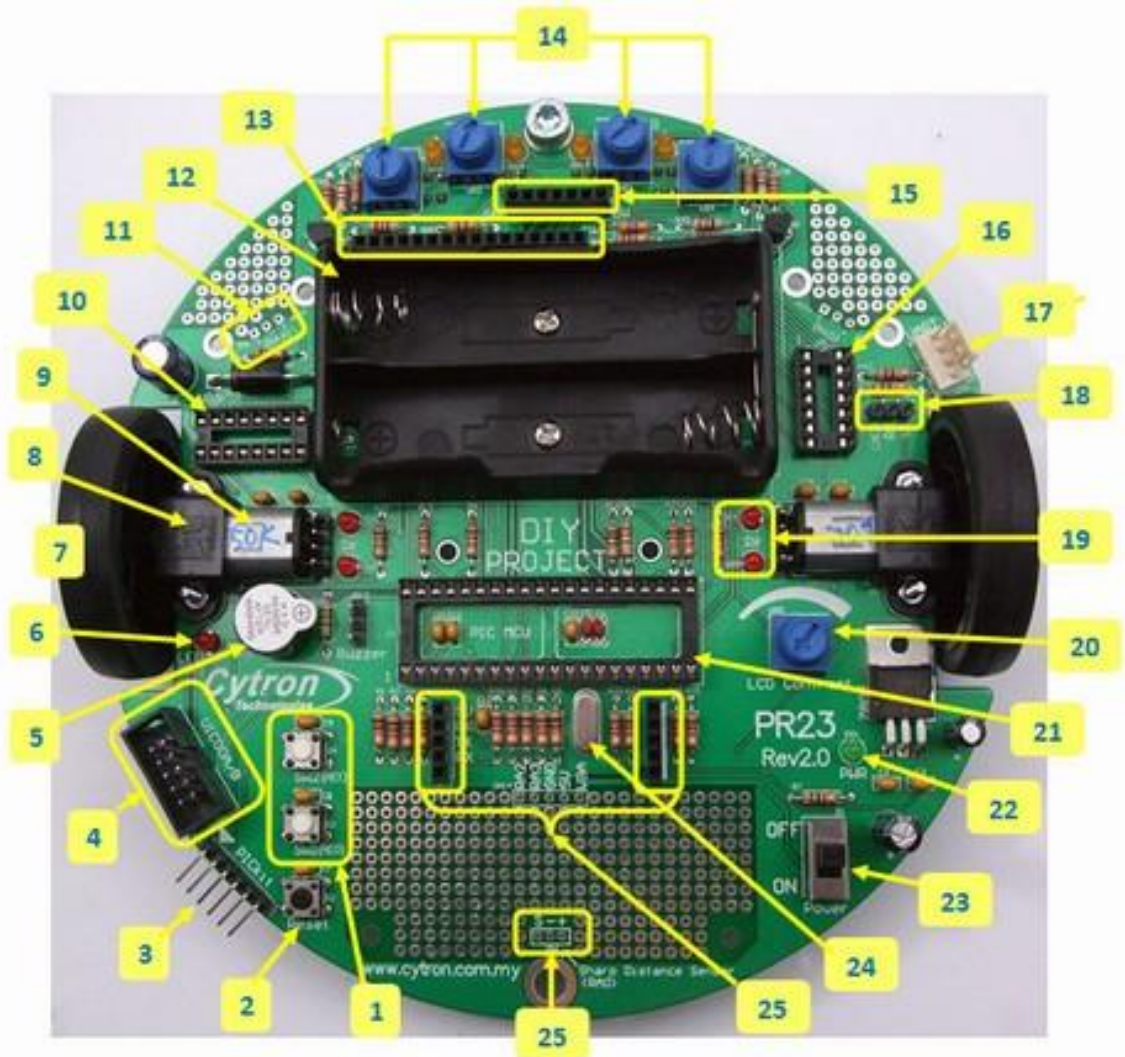


Figure 4: PR23-R2 - Multifunction Mobile Robot Component

Each number represents the components that are listed in the list of components below. The main parts of this Multifunction Mobile Robot are the microcontroller, electric motor, motor driver, batteries, infrared sensors and tires. This robot is powered using two 18650 size 3.7 volt batteries.

Component list:

1. Programmable Push Button (SW1 and SW2).
2. Reset button.
3. PICkit2/3 ICSP header pin (For loading program).
4. UIC00A/B box header (For loading program).
5. Buzzer, sharing same pin with LED, enable using JP12 jumper.
6. Programmable LED (Red), sharing pin with Buzzer.
7. Mini Wheel, 1 pair, left and right.
8. Micro Gear Motor Bracket, 1 pair, left and right.
9. SPG10 Micro Metal Gear Motor, Gear Ratio = 150:1.
10. 16-pin IC socket for L293D, motor driver IC.
11. Pads for IR01A, left and right, optional item.
12. 18650 rechargeable Li-ion, 2 cell battery holder.
13. 16-ways header socket for 2x16 character parallel LCD.
14. potentiometer/preset to calibrate or teach the infrared sensor.
15. 7-ways header socket for EZ1 Ultrasonic Range sensor, optional.
16. 14-pin IC socket for LM324, comparator.
17. 2510-04 connector, for UC00A connection, optional.
18. 3-way header pin, jumper to select RX connection to EZ1 ultrasonic, or to SK and UC00A.
19. A pair of LED indicator, to indicate the status of Micro Metal Gear Motor, left and right.
20. Potentiometer/preset to adjust LCD contrast.
21. 40-pin IC socket for PIC16F877A.
22. Green 3mm LED to indicator 5V power.
23. Main power switch
24. 20MHz Crystal for PIC16F887A
25. A pair of 5-way header socket for Cytron's SK series of board.

Most of grid following robot are in early phase of development. It has several solution to implement the functionality of the grid following. Even, most available grid following robot are for 90 degree grid following. Thus, the movement of robot from a position to another position is still limited to the design of the grid. Some of the researchers believe if grid following robot is further studied, it will give a lot of benefit especially in industries.

Work in iterative manner will make worker feel bored after some time but with the existence of grid solver robot, the work will be more interesting. This will also make the work process faster in term of shifting load properly and worker can do another task simultaneously (Saxena, 2012). This paper proposed a technique to solve the grid following algorithm, the method called ring structure. Below is the representation of the grid in term of grid structure.

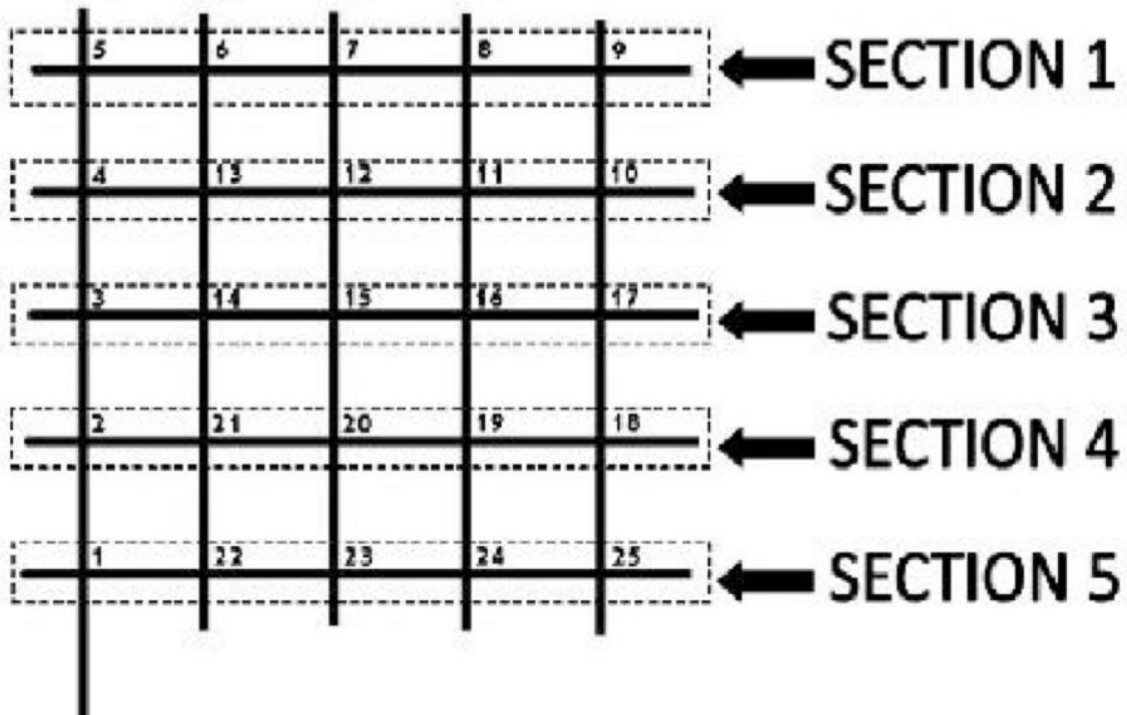


Figure 5: Ring structure of grid solving solution

As shown in the figure above, each intersection of the grid is numbered from 1 until 25 to allow the grid solver robot to go to each of position defined by using number. The grid numbering forms a continuous connection between each of the intersection.

METHODOLOGY

3.1 Research Method

There are a few research methodologies that will be used to conduct the experiment in order to add 45 degree turn to the turtle robot. Here are the project activities as the method of research that will be used to solve this problem.

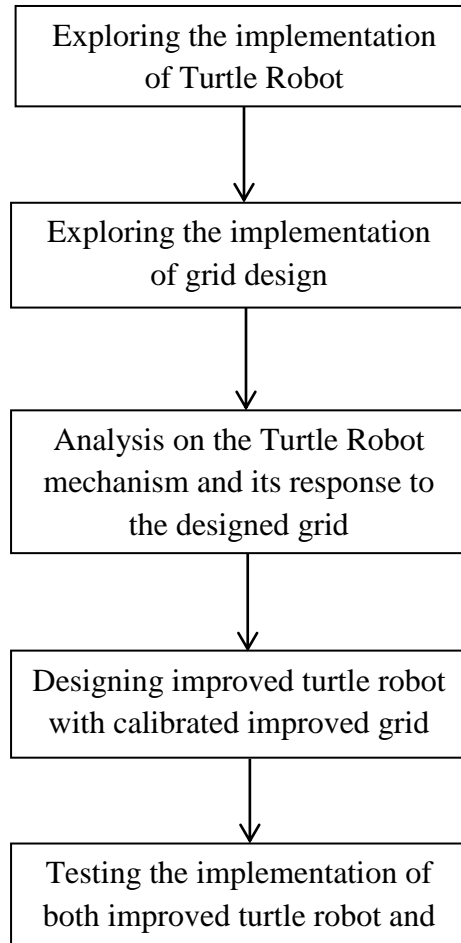


Figure 6: Research method integrated with project activities flow chart

3.2 Project Activities

Some of the specific project activities that are on-going task are represented in the flow chart below. This flow chart rather a specific physical tasks to implement each capability of the turtle robot together with the design improved grid.

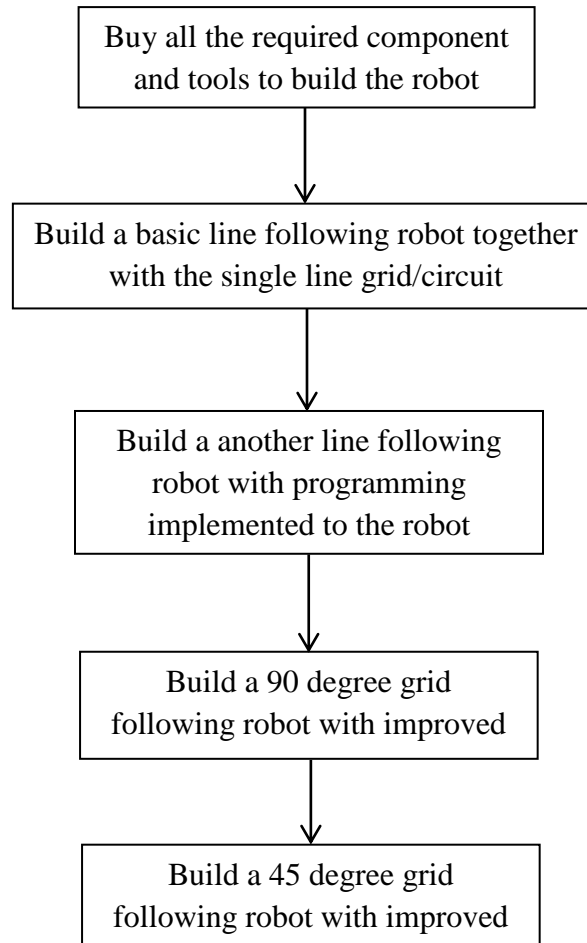
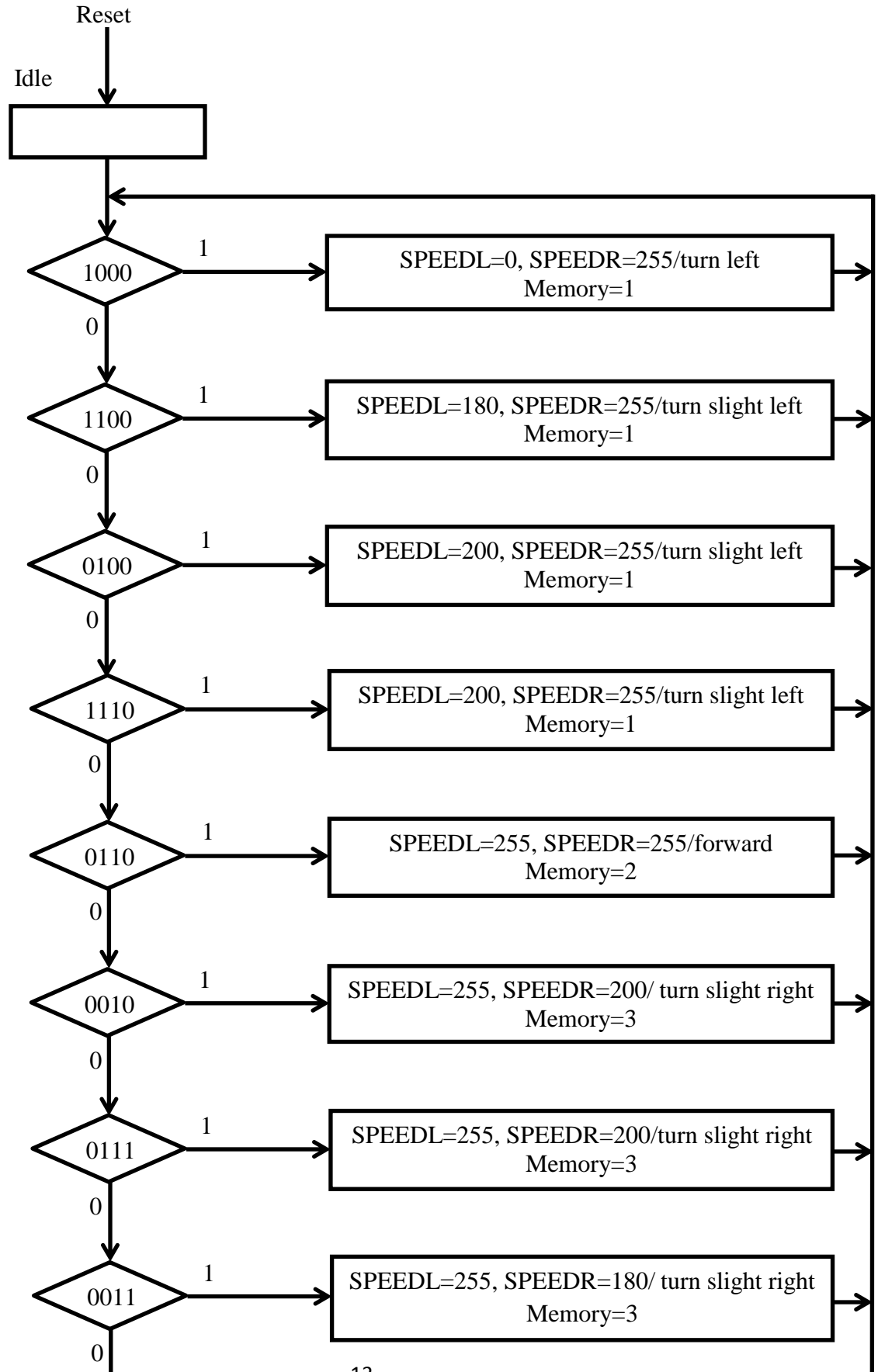


Figure 7: Project activities flow chart

Briefly, first few activities need to be done after buying the robot is to assemble all the part because the DIY came in loose components/parts. Then, line following function is tested on the circuit to check whether the components are functioning properly or not. Next, a new design grid is built as well as developing the 90 degree grid following function. Coordinate and direction function is then added to enable robot navigation on the grid.



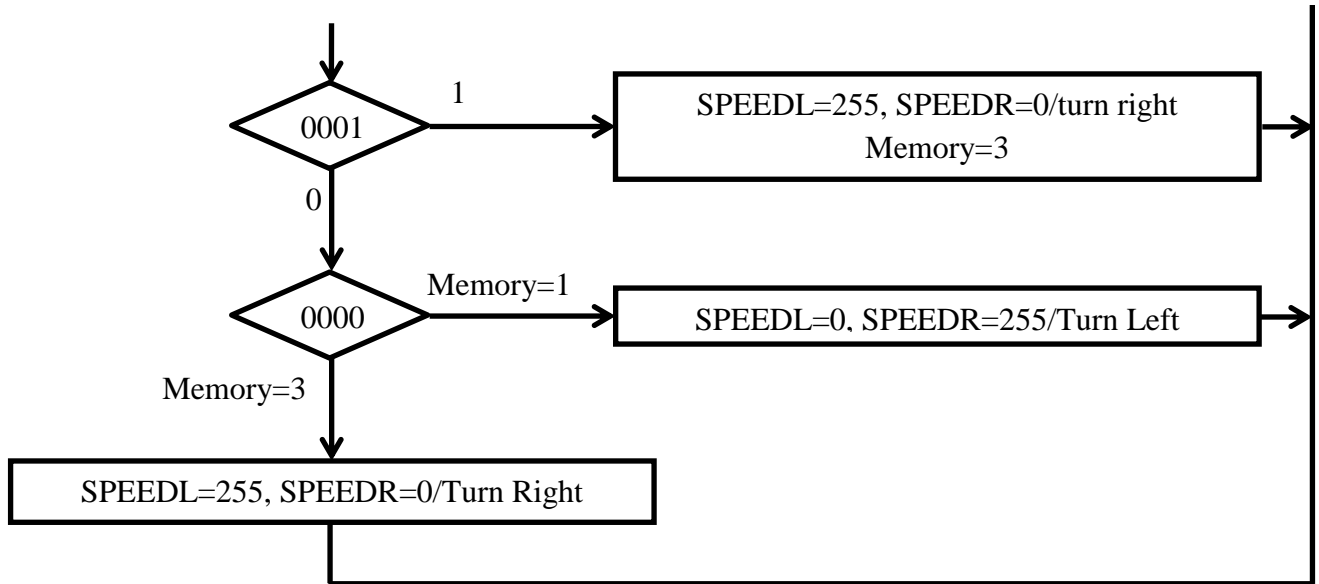


Figure 8: Algorithmic State Machine chart for the Line-Following program code

Basically the algorithm for the line-following functionality is as follow. Details of the flowchart/ (Algorithmic State Machine, ASM) chart:

- a. Sensors are represented as S1, S2, S3, S4 = Sensor_Left, Sensor_MiddleLeft, Sensor_MiddleRight, Sensor_Right arrangement are used in the diamond box (condition box) represented by bitwise for example S1S2S3S4 = 0101 which is Sensor_Left = 0, Sensor_MiddleLeft = 1, Sensor_MiddleRight = 0, Sensor_Right = 1
- b. SPEEDL and SPEEDR mean the speed of the L, Left motor and R, Right motor.
- c. ‘Memory’ functions as to remember last condition of line detected and is given value as follow: left = 1, middle = 2, right = 3 where if the sensors lost the line, the robot will follow the last condition of the sensors.

After that, the testing process/project activities become faster as we just have to simply modify the 90 degree grid following code to enable the 45 and 135 degree grid following. The main program is developed to test is functionality to perform grid following. Several problems are faced during the development of the code. These problems are further discussed in the result and discussion part.

There are several 45 degree grid solutions. The 45 degree grid is also exist in a form of junction, therefore the design and algorithm must be different or the robot must be able to determine which one is the 45 degree or 90 degree junction. Here are some of the solution available for the 45 degree grid design

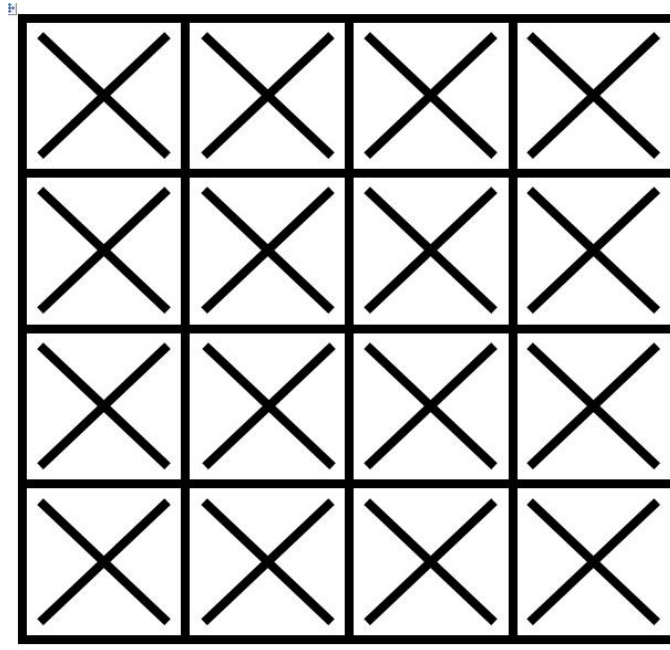


Figure 9: First solution adding 45 degree grid to 90 degree grid

On the other hand, implementation of grid following has to apply pointer and reference to the main program. The problem occurs when the value is not passing properly which has caused some garbage value inside the variable. The problem is then overcome by simulating the code using GNU C compiler and Terminal Desktop Environment in Ubuntu Linux.

These two designs of grid are further tested to see if there is sensors fluctuation or any mistake/fail that will happen during the turn of the mobile robot. The first design of grid causes the sensors to move forward using delay because no line is detected for certain short of period. This had caused the turtle robot orientation diverted out of the black line. The disorientation of turtle robot will cause failure in turning function and other next function to be done.

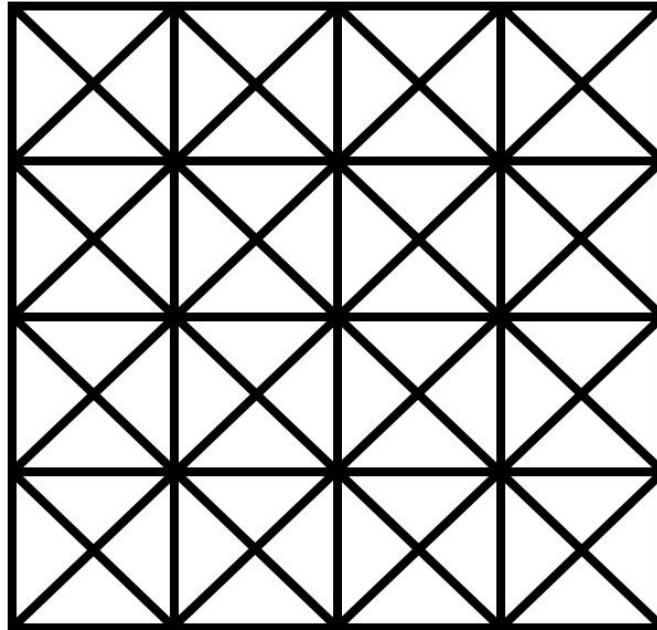


Figure 10: Second solution adding 45 degree grid to 90 degree grid

On the other hand, the second solution of grid following design allows the turtle robot to follow the line accordingly but with certain condition. A delay need to be put when moving forward after the sensors detect a junction. This is to prevent sensors fluctuation so that the robot will turn to certain degree properly and the coordinate counting is increasing accordingly.

These results are further discussed in the results and discussion part. There are many factors that affect the process of turning the robot to a certain degree which are the speed of the robot before, during and after the degree turning, the delay needed for the turtle robot to properly turning and the sensors detection after performing the turn. The sensitivity of the sensors is affected by the speed of the motor and will be further discussed in the next chapter.

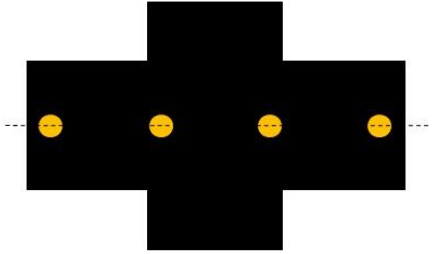
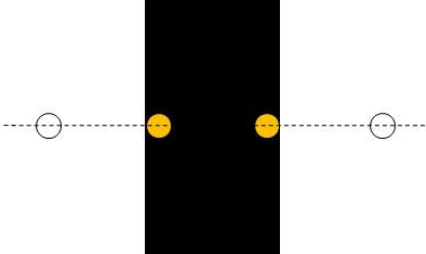
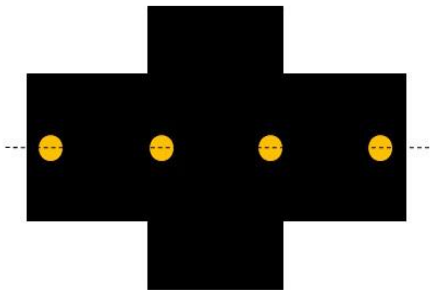
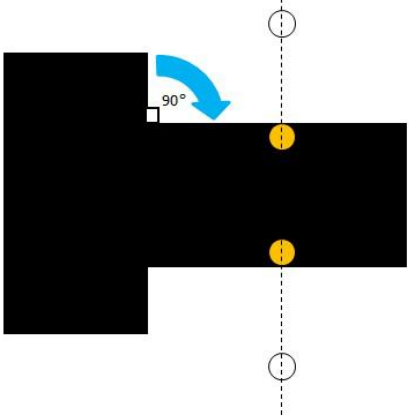
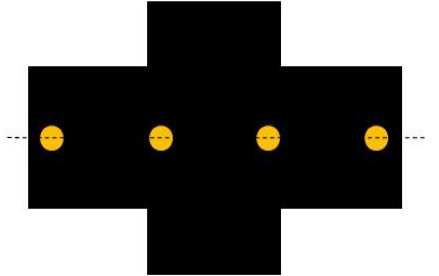
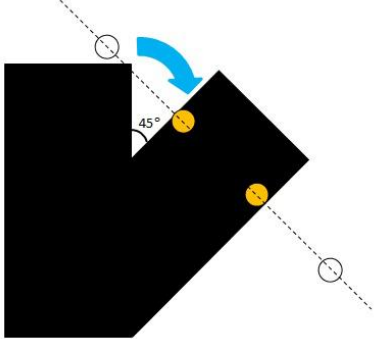
Current State	Next State	Description
 <p data-bbox="289 558 737 663">Sensors detect junction, robot adding coordinate (e.g. coordinate y: 0,0 → 0,1)</p>	 <p data-bbox="764 590 1213 663">Sensor middle left and middle right keep the robot follow the line</p>	<ul data-bbox="1235 239 1430 663" style="list-style-type: none"> - Line following - Function step() - Current state is either detecting junction, of any of these 90°, 45° and 135° degree turns
 <p data-bbox="293 1146 729 1251">Sensors detect junction as above, delay for 400 milliseconds before rotate to right</p>	 <p data-bbox="786 1167 1192 1241">Sensor middle left will stop the (650ms) rotation to 90° turn</p>	<ul data-bbox="1235 680 1430 1104" style="list-style-type: none"> - Turn 90°, to the right - Function right_ninety() - Sensor middle Left will stop the turn and continue with line following (step ()) function.
 <p data-bbox="293 1644 729 1749">Sensors detect junction as above, delay for 400 milliseconds before rotate to right</p>	 <p data-bbox="786 1623 1192 1696">Sensor middle left will stop the (325ms) rotation to 45° turn</p>	<ul data-bbox="1235 1262 1430 1728" style="list-style-type: none"> - Turn 45 ° to the right - Function right_fortyfive () - Sensor middle Left will stop the turn and continue with line following (step ()) function.

Table 1: Basic State table of the Sensors detection

Table above basically explains the major Finite State that will be used to decide the orientation of the robot itself. This table includes the delay, sensors detection, function call and the orientation performed by the turtle robot after the function is called. The turn to certain degree is further added with turning 45, 90 and 135 degree left, turn 135 and 180 degree right. The function call explanation will be discussed in the result chapter.

Last but not least, the positioning method of the turtle robot. In this case we are using coordinate and direction variables which are 4 cardinal directions and 4 ordinal directions. The 4 cardinal directions are north, south, west, east and the 4 ordinal directions are northwest, northeast, southwest, southeast. These directions will limit and determine the position of the turtle robot in term of coordination.

Below is the flow chart of turning to certain degree process. These are step by step condition and switch box that will happen during the process. This basic flow chart will also be used for the development of 45, 135 and 180 degree grid design.

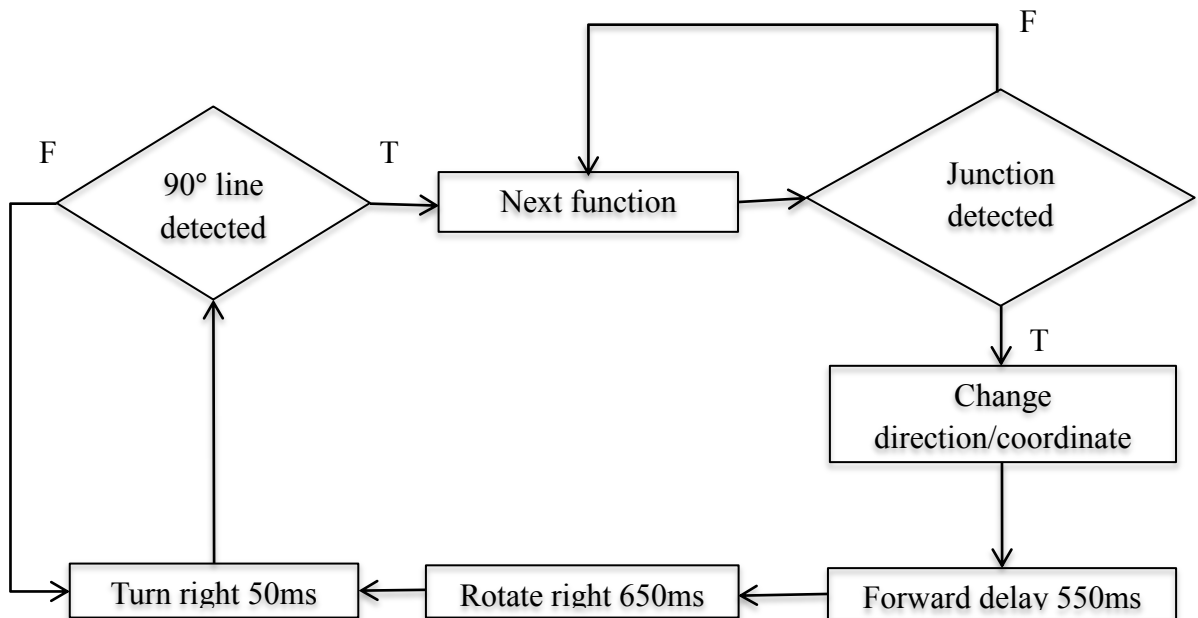


Figure 11: Algorithmic State Machine chart for 90 degree turn

3.3 Tools and Equipment

Category	Hardware	Software
Turtle Robot	Electronic components	
	Soldering kit & lead	
	18650 battery Charger	
	UIC00B Programmer	PICkit 2 v2.61
Grid/Line Mat	Vinyl banner	
	Black Insulating Tape	
	White Foam Board	
Environment	Personal Computer	Windows 7 / Ubuntu 12.04
	Digital Multi meter	MPLAB X v1.95
	Stationaries	XC8/ GNU C Compiler

Table 2: List of Tools and Equipment used

3.4 Key Milestone

Based on the objective with related to the methodology suggested, there are some explanation regarding the key milestone that we are going to achieve.

1. To design an improved grid suitable for use with a turtle robot that allows more than straight line and 90 degree turn.

In this objective, we will implement 45 degree grid to allow turtle robot to turn 45 degree on the new improved grid. Thus, to design a project, we need to study the function of the project thoroughly because the principle behind a functional thing is; it is shaped according to its function.

2. To develop a new algorithm for a turtle robot to use on the new grid

On the other hand, the algorithm needs to be enhanced so that it can correlate with the 45 degree turn implementation. This needs a very careful observation on how the turtle robot responds to the environment of the grid and to configure its specific algorithm according to each event.

3.5 Final Year Project Gantt chart

No	Activities	Week													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Final Year Project I	Selection of Project Topic	█	█												
	Preliminary Research Work -Exploring the implementation of Turtle Robot & grid design		█	█	█	█									
	Submission of Extended Proposal Defense						█								
	Proposal Defense							█	█						
	Project Work Continues -Analysis on the Turtle Robot mechanism and its response to the designed grid -Designing improved turtle robot with calibrated improved grid									█	█	█			
	Submission of Interim Draft Report													█	
	Submission of Interim Report														█
Final Year Project II	Project Work Continues -Assembling DIY Mobile Robot -Building 90 degree grid mat	█	█	█	█	█									
	Submission of Progress Report								█						
	Project Work Continues -Building 45 and 135 degree grid mat -Developing new algorithm and code for new grid design									█	█	█	█		
	Pre-SEDEX / Electrex										█				
	SEDEX											█			
	Submission of Draft Report												█		
	Submission of Dissertation (soft bound)													█	
	Submission of Technical Report														█
	Viva/Oral Presentation														█
	Submission of Dissertation (hard bound)														█

Week 15

RESULT AND DISCUSSION

4.1 Data Gathering and Analysis

Basically, there are three main components in this project. Those components are grid, algorithm or program code and prototype of the robot. The grid is used as the guide for the robot to move from a position to another position. Based on the project activities previously in Interim Report, below are the stages of designing the grid:

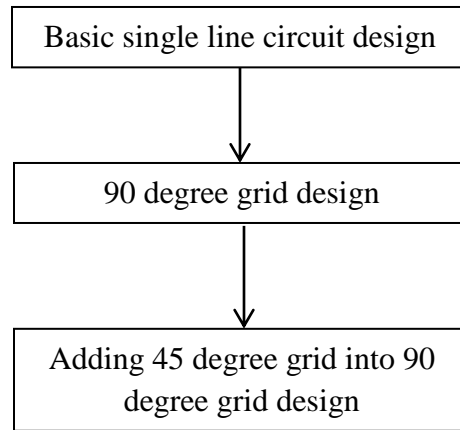


Figure 12: Line and Grid design stages

The details of the grid design are as follow:

- The surface is made of vinyl. This material can be obtained from any unused vinyl banner. Vinyl banner has white surface at the back side of it.
- The line is made of black insulating tape. The specification is determined as described in the manual of the robot prototype. Below is the explanation:

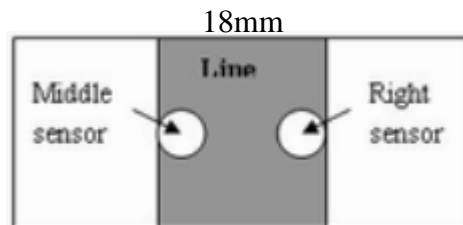


Figure 13: Line width according to the sensors position

The width of the line is equal to the maximum length two sensors. According to the sensors installed on the robot as shown in figure above, the width of the line should be 18mm. In this project, we use black insulating tape with the specification of 6.4m x 18mm x 0.12mm (Length x Width x Height (thickness)) with length is as long as possible.

There is problem with the uneven surface of the vinyl banner. This problem had caused the insulating tape does not stick properly. Furthermore, this will cause the movement of the robot is stopped or diverted when following the line. Thus, below is the solution to make the surface of the vinyl banner become even:

1. Put the vinyl banner on top of ironing board/even surface.
2. Put a wet cloth on top of the banner
3. Start ironing the wet cloth on a small area of the vinyl banner to see the effect

Below is the design for 90 degree grid. The number put in each boxes is the coordinate which will synchronized with the algorithm in term of detecting the number of junction passed by the robot.

0,3	1,3	2,3	3,3
0,2	1,2	2,2	3,2
0,1	1,1	2,1	3,1
0,0	1,0	2,0	3,0

Figure 14: 90 degree grid design with coordinate

The grid design has been decided and tested. Basically this grid can be improved so that turtle robot can do a lot more movement with a lot more junction:

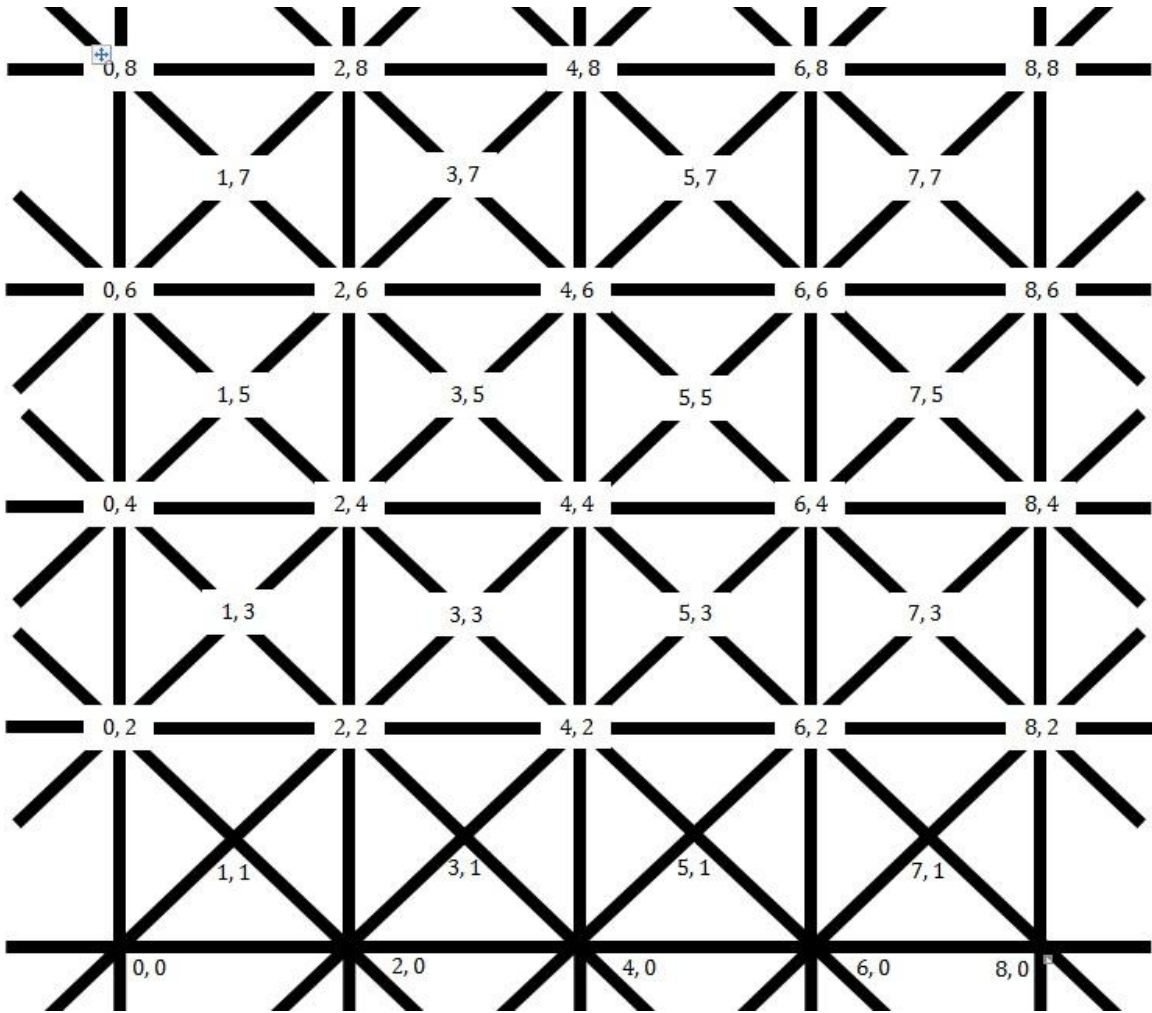


Figure 15: Currently confirmed grid design

Next is the basic state table used to determine the condition of the sensors when doing grid following function. These table and grid map above are used to pre-program the turtle robot according to the coordinate and orientation of the robot that we want. Below is the state table that show the orientation of the turtle robot based on the sensors detection on the grid.

Next part is the program code or the algorithm. According to the project activities, the following flowchart explains the stage of developing the program code:

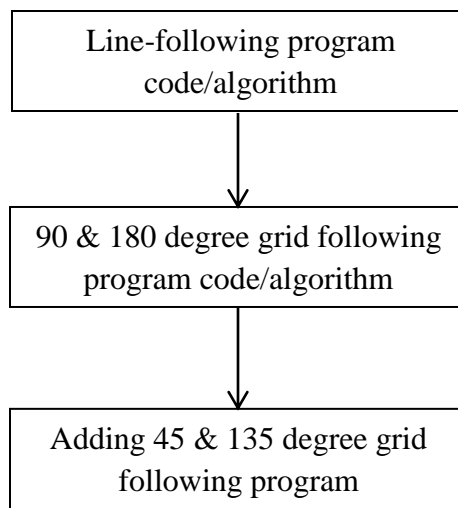


Figure 16: Program code/Algorithm development flowchart

The progress now is on the developing the 90 degree program code. Another important part of the project is the robot itself. Below are the basic parts of the robot that needed to perform the line-following, 90 degree and 45 degree function.

1. Sensors, in this project we used 4 sensors to detect the line
2. Motors, 2 motors which in the left and right is used to move the robot
3. Microcontroller, to run the program code loaded from the computer
4. Battery, usually rechargeable battery with total output around below 12volts
5. Programmer used to program the microcontroller using MPLAB software

Flow chart of the 90 degree turn process is further detail into pseudo code before it is programmed into C language. Below is the pseudo code i.e. the C code structure and verbal explanation of each process:

1. Turtle robot do line following
2. If junction is detected (and next instruction is turn 90 degree right)
 - 2.1 Change coordinate
 - 2.2 Move forward with delay 550 milliseconds
 - 2.3 Rotate to the right with delay of 650 milliseconds
 - 2.4 If middle left sensor detects the 90 degree grid

- 2.5 Stop rotating
- 2.6 Change direction (North to East)
- 2.7 Do next instruction (line following)
- 3. Else (while loop)
 - 3.1 Line following
 - 3.2 If junction detected
 - 3.3 Change coordinate
 - 3.4 Do next instruction

Next is the pseudo code in the form of C code structure, this is when junction is detected and the next instruction is to turn 90 degree right:

```

Task right_ninety ()
{
  Robot run forward 550ms after junction detected
  Robot stop
  Robot rotate to the right 650ms
  Robot turn to the right 50ms to align the sensor to the 90 line
  While (sensor middle right does not detect line)           //do function above
    Sensor detect line
    Robot stop
  If (previous direction is north)
    Then change to East
    Display 'E' on the LCD
  If (previous direction is east)
    Then change to South
    Display 'S' on the LCD
  If (previous direction is south)
    Then change to West
    Display 'W' on the LCD
  If (previous direction is west)
    Then change to West
    Display 'N' on the LCD
  Return
}

```

The delay is specifically calculated for each turn. The calculations include the speed of the rotation, moving forward and turn. Next is another important function in the program which is step function. This function is to move turtle robot from a junction to another junction and at the same time change coordinate of the turtle robot. Coordinate is used to determine the position of the turtle robot on the grid.

The following is a brief pseudo code for step function, followed by the pseudo code in the C code structure.

1. Turtle robot do line following
2. While (sensor left and right do not detect perpendicular line)
3. Do function above
4. If (junction is detected)
 - 4.1 Robot stop
 - 4.2 Change coordinate (either x or y coordinate +/- by 1)
 - 4.3 Move forward with delay 550ms
5. Do line following

Next is the pseudo code in the C code structure:

```
Task step (passing value x and y)
{
  Move forward for about 400ms
  While 1
  {
    Do line following
    If (left sensor and right sensor detect black line)
    Break the loop
  }
  Robot stop
  If (direction is north)
  {
    Increase y-coordinate by 2
  }
  If (direction is south)
  {
    Decrease y-coordinate by 2
  }
  If (direction is east)
  {
    Increase x-coordinate by 2
  }
  If (direction is west)
  {
    Decrease x-coordinate by 2
  }
}
```

```

If (direction is northeast)
{
  Increase x-coordinate by 1
  Increase y-coordinate by 1
}
If (direction is southeast)
{
  Increase x-coordinate by 1
  Decrease y-coordinate by 1
}
If (direction is southwest)
{
  Decrease x-coordinate by 1
  Decrease y-coordinate by 1
}
If (direction is northwest)
{
  Decrease x-coordinate by 1
  Increase y-coordinate by 1
}
}

```

The calculation/specification for the delays is as follow:

Function	Speed of motor (PWM)		Delay (ms)
	Right	Left	
Forward	255	255	*depend
Backward	255	255	*depend
Turn Right	0	255	*depend
Turn Left	255	0	*depend
Rotate 90 Right	230	230	650
Rotate 90 Left	230	230	650
Rotate 45 Right	230	230	325
Rotate 45 Left	230	230	325
Rotate 135 Right	230	230	975
Rotate 135 Left	230	230	975
Rotate 180 Right	230	230	1400

*forward delay in step function is 400ms/550ms while turn right/left function is 50ms

Table 3: Delay allocation for each sub-function

The calculation of the delay is basically important for rotation delay and forward delay after meeting the junction. The calculation of the forward delay is as procedure below:

1. Set the speed of rotation to 255
2. Put rotate right into while 1 loop
3. While turtle robot is rotating at 255 speed, take the time measurement for about 10 readings
4. Find the average time reading
5. Calculate the speed = $2\pi r$ /average time
6. Repeat step for 230 speed of rotation

The speed is then used to calculate the delay for both forward after junction detected and rotation delay for each degree of rotation. The calculation is as follow:

- a. Speed of rotation (PWM = 230) = $\frac{2\pi(7.5 \text{ cm})}{2.5ms} = 18.08cm/s$
- b. Speed of moving forward (PWM = 255) = $\frac{2\pi(7.5 \text{ cm})}{2.6ms} = 18.74cm/s$

These two speeds are then used to calculate the delay as follow:

Sub-Function	In-Function	Delay needed (approx)
Ahead (move forward)	Step	550ms (distance = 10.3cm)
Rotate (right/left)	Turn 45 degree	325ms (distance = 5.88cm)
	Turn 90 degree	650ms (distance = 11.75cm)
	Turn 135 degree	975ms (distance = 17.63cm)

Table 4: Delay calculation

4.2 Experimentation and Modeling

The first stage of program code development and line-following circuit has been done. Both of this line circuit and programmed robot with line following program is then experimented to see if it is working accordingly or not. The experiment results a successful output. The robot is able to follow the line perfectly as shown in picture below.

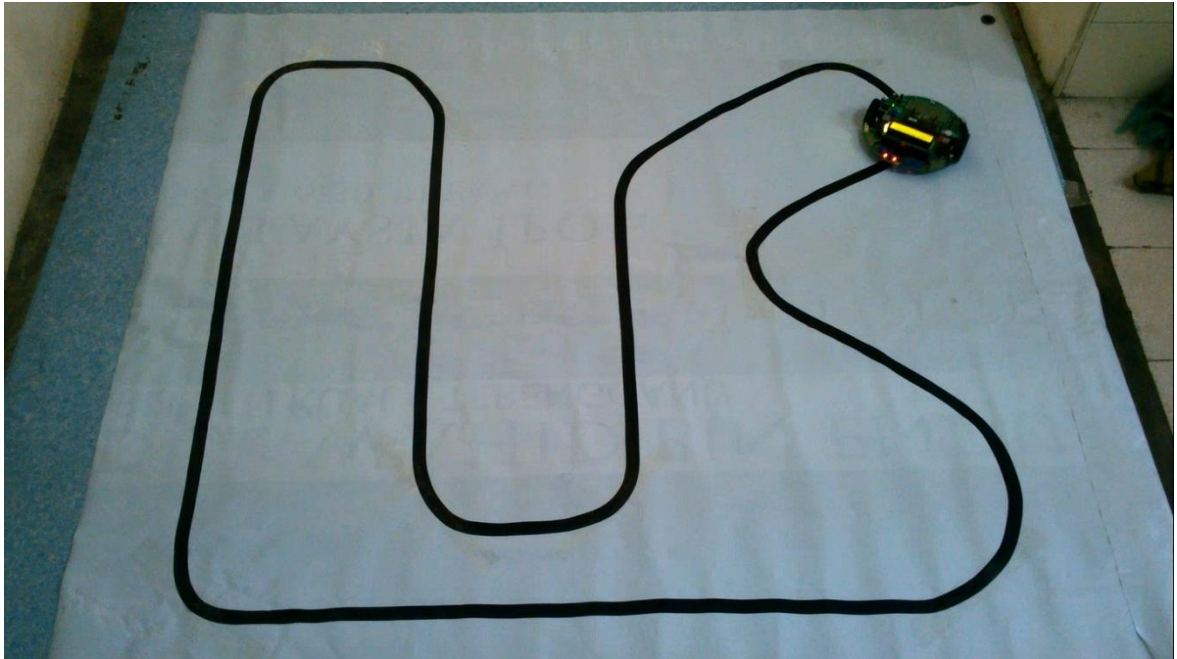


Figure 17: Experimenting line-following function

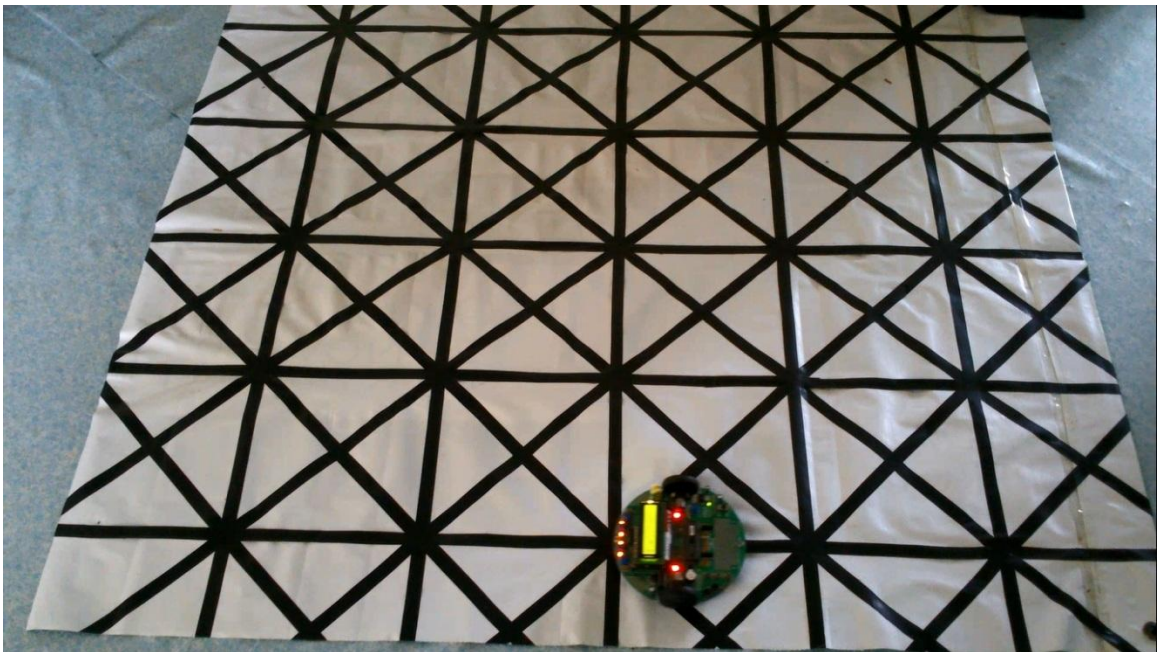


Figure 18: Experimenting grid following function

4.3 Prototype

Here is the line circuit design for the robot. This line circuit will utilize the line-following program code run by the microcontroller.

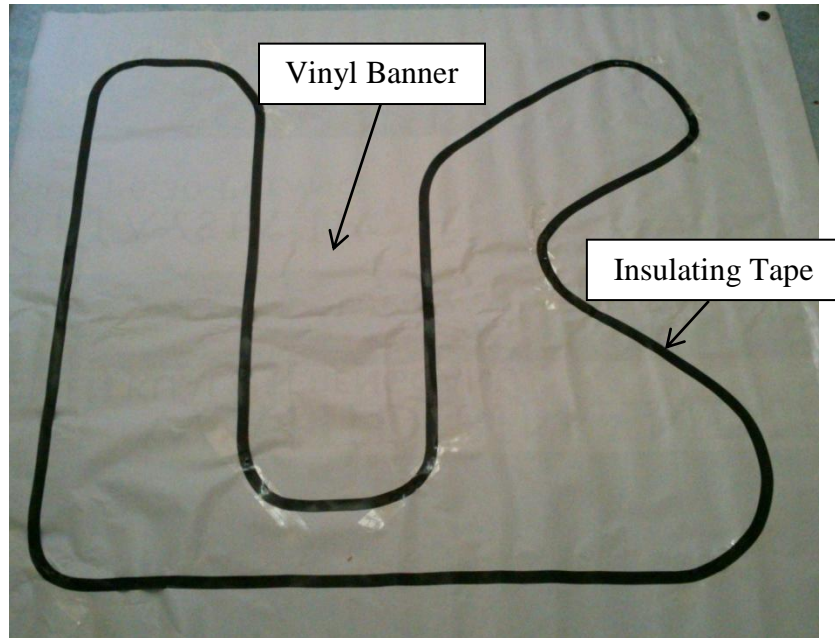


Figure 19: Line-following Mat

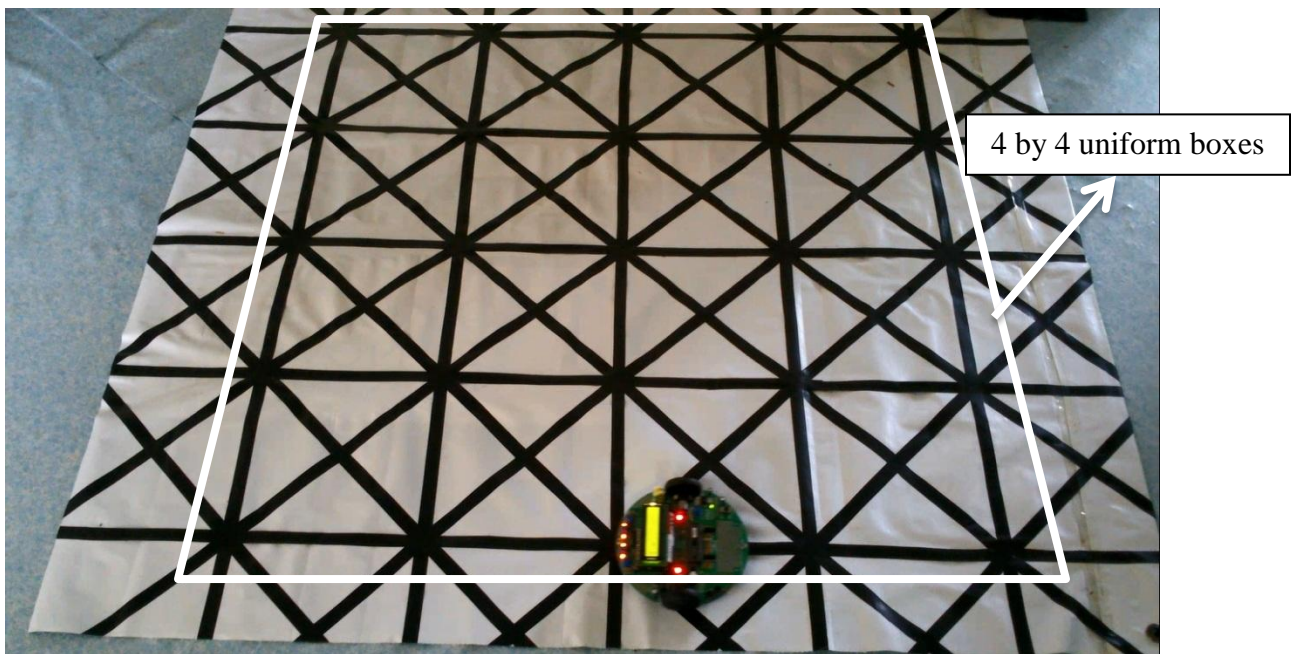


Figure 20: 45, 90 and 135 degree Grid Design

The grid design has about 4 by 4 uniform boxes. Each boxes has the same length and width which is equal to 22cm. Initially, most of the grid solution for mat is only for 90 degree grid. The solution is only suitable with the algorithm used to detect 90 degree junction and to turn 90 degree right or left. Thus, 45 degree junction is added and this directly cause the grid to have 135 degree junction. The grid is now suitable for testing and the effectiveness of the grid as well as the accuracy of the robot being tested.

Some pattern or types of route has been tested, below is the pseudo code of the main program and the figure of the routes:

1st test pattern/route for turtle robot:

1. Turtle robot at coodinate (0,0)
2. Increase in y-coordinate
3. Move to coordinate (0,4) //while (x,y) less than (0,4)
 - 3.1 Change coordinate variable
 - 3.2 If junction detected //at (0,4)
 - 3.3 Turn 90 degree right
 - 3.4 Change direction and variable
4. Move to coordinate (4,4) //while (x,y) less than (4,4)
 - 4.1 Change coordinate variable
 - 4.2 If junction detected //at (4,4)
 - 4.3 Turn 45 degree left
 - 4.4 Change direction variable
5. Move to coordinate (8,8) //while (x,y) less than (8,8)
 - 5.1 Change coordinate variable
 - 5.2 If junction detected //at (8,8)
 - 5.3 Turn 135 degree right
 - 5.4 Change direction variable
6. Move to coordinate (8,0) //while (x,y) not equal (8,0)
 - 6.1 Change coordinate variable
 - 6.2 If junction detected //at (8,0)
 - 6.3 Turn 90 degree right

- 6.4 Change direction variable
- 7. Move to coordinate (0,0) //while (x,y) not equal (0,0)
- 7.1 Change coordinate variable
- 7.2 If junction detected //at (0,0)
- 7.3 Turn 90 degree right
- 7.4 Change direction variable
- 8. While loop //do the main program again

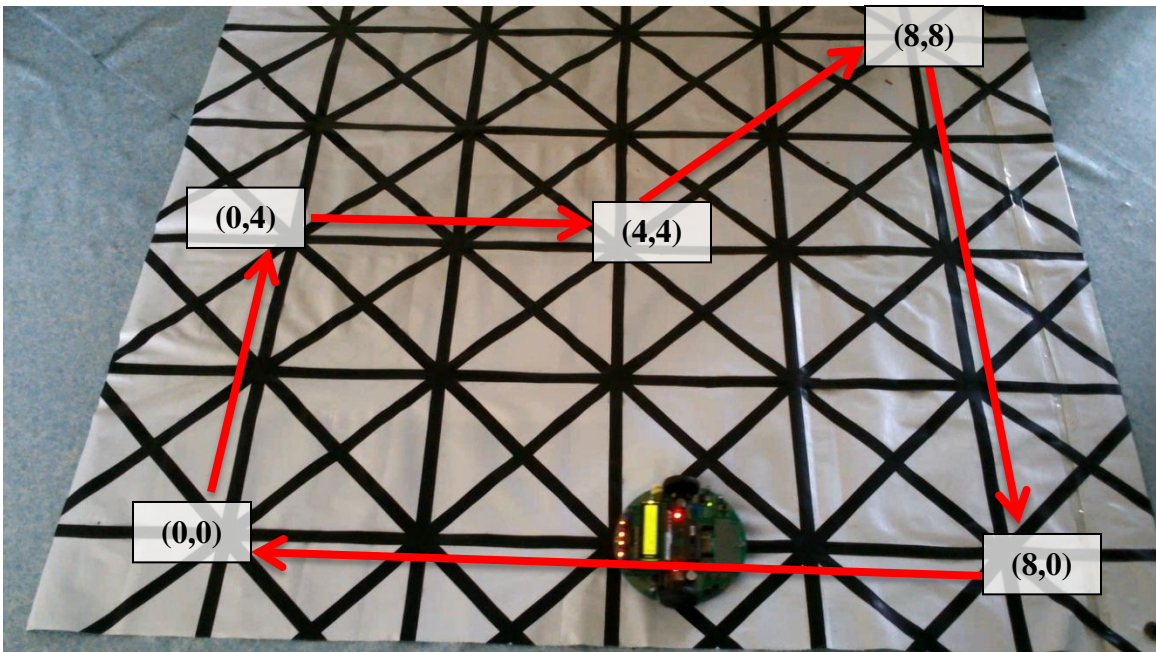


Figure 21: The 1st tested pattern/route

2nd test pattern/route for turtle robot:

- 1. Turtle robot at coordinate (0,0)
- 2. Increase in y-coordinate
- 3. Move to coordinate (0,6) //while (x,y) less than (0,6)
- 3.1 Change coordinate variable
- 3.2 If junction detected //at (0,6)
- 3.3 Turn 45 degree right
- 3.4 Change direction and variable
- 4. Move to coordinate (2,8) //while (x,y) less than (2,8)
- 4.1 Change coordinate variable

- 4.2 If junction detected //at (2,8)
- 4.3 Turn 45 degree right
- 4.4 Change direction variable
- 5. Move to coordinate (4,8) //while (x,y) less than (4,8)
 - 5.1 Change coordinate variable
 - 5.2 If junction detected //at (4,8)
 - 5.3 Turn 90 degree right
 - 5.4 Change direction variable
- 6. Move to coordinate (4,4) //while (x,y) not equal (4,4)
 - 6.1 Change coordinate variable
 - 6.2 If junction detected //at (4,4)
 - 6.3 Turn 90 degree left
 - 6.4 Change direction variable
- 7. Move to coordinate (8,4) //while (x,y) not equal (8,4)
 - 7.1 Change coordinate variable
 - 7.2 If junction detected //at (8,4)
 - 7.3 Turn 90 degree right
 - 7.4 Change direction variable
- 8. Move to coordinate (8,2) //while (x,y) not equal (8,2)
 - 8.1 Change coordinate variable
 - 8.2 If junction detected //at (8,2)
 - 8.3 Turn 45 degree right
 - 8.4 Change direction variable
- 9. Move to coordinate (6,0) //while (x,y) not equal (6,0)
 - 9.1 Change coordinate variable
 - 9.2 If junction detected //at (6,0)
 - 9.3 Turn 45 degree right
 - 9.4 Change direction variable
- 10. Move to coordinate (0,0) //while (x,y) not equal (0,0)
 - 10.1 Change coordinate variable
 - 10.2 If junction detected //at (0,0)

10.3 Turn 90 degree right

10.4 Change direction variable

11. While loop

//do the main program again

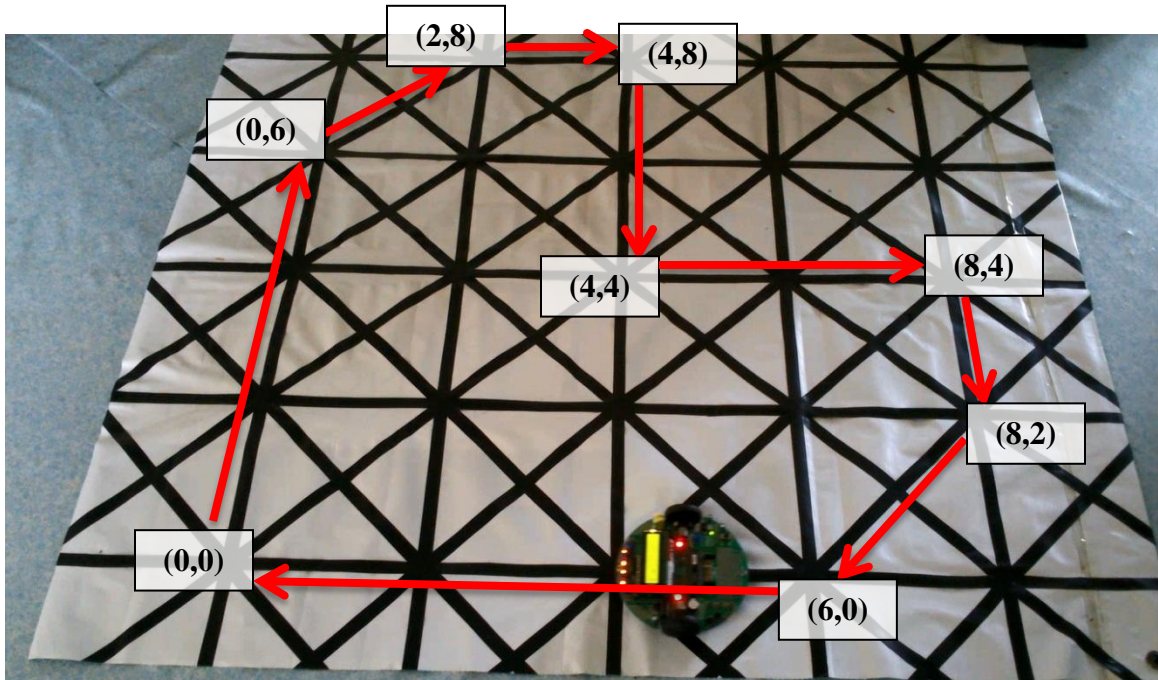


Figure 22: The 2nd tested pattern/route

From these two tested pattern, there are several things that need to be considered when the turtle robot is performing the main function/program which is following the preprogrammed route.

1. The accuracy of the turtle robot on the designated grid

The accuracy is very much depend on the grid design. In this case, our vinyl mat has uneven surface, causing a little bit disorientation of the turtle robot. This had also caused a little bit more time for the turtle robot to re-aligned its sensor to the line but the straight line is only 22cm long, with the speed; it is not enough for the robot to do that.

2. The effectiveness of the grid design

The problem above has come to the questioning of the effcetiveness of the grid design. Thus, new grid design need to be developed solving on 2 main problem which is the surface and increasing straight line.

CONCLUSION & RECOMMENDATION

Relevancy of the objectives is again evaluated after doing several project activities. The relevancy is confirmed when the project has the possibility to be solved according to the Gantt chart planned. Therefore, the project activities will be proceeding as usual according to the planner. It can be said that this project had achieved 90 percent of the objective in term of early development stage/phase.

However there are a lot of recommendations for this project. First, the uneven surface of the vinyl mat. Although vinyl mat can be rolled and portable, but the surface will remain uneven because of the heat and anything that had pressed accidentally in any condition. This will decrease the accuracy of the turtle robot's sensors to detect line as well as may cause a little bit disorientation while turtle robot is performing the main program/function.

It is recommended to build the grid on the white foam/polystyrene board to preserve the evenness of the surface. Another recommendation is to put another black line in between each junction of x (from west to east) and y (from north to south) coordinates. Our algorithm is to increase or decrease of x-coordinate (from west to east) and y-coordinate (from north to south) by 2 in each detected junction. Putting another black line in between will decrease the limitation of turtle robot movement and position.

Another recommendation is to include fail-safe program in the main program. This will be useful if there is event of power failure of disorientation/miss position of the turtle robot. It is suggested that if there is failure event (interruption), the robot may able to detect it and do the next instruction to save it from any fatal or damage. For example, turtle may stop and display 'start again' on the LCD if any of the failure happens.

The ultimate purpose of this project is for educational approach in teaching programming. Thus, it is recommended to add some physical interface on the turtle robot i.e. switch or push button to indicate the orientation and coordinate that can be directly programmed to the robot. This is called as block programming which uses simple interface to program a robot. Other suggestion is to make turtle robot more flexible by adding the turtle robot capability to move from any position on the grid.

REFERENCE

- [1] L. M. Surhone, *et al.*, *Turtle (Robot)*, Saarbrücken, Germany, VDM Publishing, 2010
- [2] R. Goldman, *et al.*, “Turtle Geometry in Computer Graphics and Computer Aided Design,” Dept. of Computer Science, Rice University, Houston, Texas
- [3] R. Goldman, “Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling,” Morgan Kaufmann, San Francisco, 2002
- [4] H. Abelson, *Apple Logo*, McGraw-Hill, 1892
- [5] T. Carroll, “ROS Meets Kinect Meets Create” in *Servo Magazine*, 2012
- [6] I. Dogan, “Microcontroller based applied digital control,” Hoboken, NJ, John Wiley, 2006
- [7] I. Dogan, “PIC Basic: Programming and Projects,” Oxford, Newenes, 2001
- [8] I. Dogan, “Advanced PIC microcontroller projects in C: from USB to ZIGBEE with 18F Series,” Amsterdam; Boston, Newnes/Elsevier, 2008
- [9] Carnegie Mellon Robotics Academy, “ROBOTC, Pseudo code & Flow Charts

C:\Users\mza\Desktop\project\fulldegrid.X\fulldegrid.c

```

=====
// Author      :MZA & CYTRON Tech
// Project     :PR23 Rev2.0
// Project description :DIY Project 23, Multifunction Mobile Robot
// Version     :v2.3
// IDE        :MPLAB X IDE v1.95
// Compiler    :HI-TECH PICC v9.83 or XC8 Compiler v1.21 (default XC)
// Date       :25 Nov 2013
// Example code is provided as "it is", Cytron Technologies do not take responsibility to
// verify, improve or explain the working of the code.
// If you have any inquiry, welcome to discuss in our technical forum:
// http://forum.cytron.com.my
// The other author, MZA will take full responsibility to the code.
=====

// include library files
=====
#ifdef __XC8)
#include <xc.h> //header file for hitech mid-range pic
#elif (HI_TECH_C)
#include <htc.h> //header file for hitech mid-range pic
#endif
// configuration
=====
#ifdef __XC) //if XC Compiler is use to compile this code
/*
 *if PIC18F452 is use as microcontroller
#pragma config OSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDT = OFF // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRT = ON // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOR = OFF // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRWD = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to b
#pragma config CPD = OFF // Flash Program Memory Code Protection bit (Code protection off)
*/

/*if PIC16F877/A, use below configuration word*/
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRT = ON // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to b
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

#elif (HI_TECH_C) //if HI-TECH C Compiler is use to compile this code
__CONFIG ( 0x3F32); //configuration bits value for
//High Speed oscillato
//Watchdog timer disable
//Power up Timer enable
//Low voltage programming disable

#endif
// define labels or constants
=====
#define _XTAL_FREQ 2000000 //Frequency of crystal oscillator, for delay, 20MHz
#define SW1 RE0 //SW1 push button is connected to RE0 of PIC
#define SW2 RE1 //SW2 push button is connected to RE1 of PIC
#define IR_L RA4 //IR01A medium range sensor for left side is connected to RA4 of PIC
#define IR_R RA5 //IR01A medium range sensor for right side is connected to RA5 of PIC

```

```

#define MOTOR_R1    RC0 //Control pin for motor, going through motor driver L293D, right motor
#define MOTOR_R2    RC3 //Control pin for motor, going through motor driver L293D, right motor
#define MOTOR_L1    RC4 //Control pin for motor, going through motor driver L293D, left motor
#define MOTOR_L2    RC5 //Control pin for motor, going through motor driver L293D, left motor

#define SEN_L        RB0 //Line Sensor, Left. Going through comparator. Dark high
#define SEN_ML       RB1 //Line Sensor, Middle Left. Going through comparator. Dark high
#define SEN_MR       RB2 //Line Sensor, Middle Right. Going through comparator. Dark high
#define SEN_R        RB3 //Line Sensor, Right. Going through comparator. Dark high

#define BUZZER       RE2 //BUZZER is connected to RE2 of PIC, active high. Is actually being share with
                        //LED
//2x16 character LCD
#define LCD_RS       RB7 //2x16 parallel LCD RS pin is connected to RB7 of PIC
#define LCD_E        RB6 //2x16 parallel LCD E pin is connected to RB6 of PIC
#define LCD_DATA     PORTD //2x16 parallel LCD Data pins are connected to PORTD of PIC
#define LCD_BLIGHT   RB5 //2x16 parallel LCD back light is connected to RB5 of PIC, active high
#define LINE1        0
#define LINE2        1

#define SPEEDL       CCP1L //PWM register for left motor, to control speed
#define SPEEDR       CCP2L //PWM register for right motor, to control speed

//label for Analog channel
#define CH0          0 // AN0 ( Ultrasonic LVEZ1 )
#define CH1          1 // AN1 ( Sharp Infrared Distance Sensor )

//direction for 90 degree capabilities mobile robot
#define N            1 //north direction
#define E            2 //east direction
#define S            3 //south direction
#define W            4 //west direction
#define NE           5 //northeast direction
#define SE           6 //southeast direction
#define SW           7 //southwest direction
#define NW           8 //northwest direction

// Global Variables
//=====
unsigned char data[6] = {0}; //general purpose array
const char diy_project [] = "Do It Yourself";
const char multifunction_robot [] = "Line&GridFunc MR"; //Multifunction Mobile Robot
const char pr23_rev [] = " PR23 Rev2.0";
const char cytron_tech [] = " Cytron Tech";
const char cytron_website [] = " cytron.com.my";
const char code_version [] = "Sample Code v2.3";
const char line [] = "1.Line Following";
//const char NDG[] = "2.90 Degree Grid";
const char FDG[] = "2.Full DeGrid";
//const char *mode_string [3] = {&line[0],&NDG[0],&FDG[0]}; //array of pointer to strings
const char *mode_string [2] = {&line[0],&FDG[0]}; //array of pointer to strings

unsigned int pulse_width = 0; //variable to store pulse width value from timer 1
//unsigned char memory = 0; //variable to memorize previous condition if sensor is out of line

//90 degree variables
int bot_dir; //variable to store robot direction (initially) = 1 (north)
int bot_x; //variable to store robot x coordinate (initially) = 0
int bot_y; //variable to store robot y coordinate (initially) = 0
int grid_x; //variable to store size of grid x = 8
int grid_y; //variable to store size of grid y = 8

// function prototype
//=====

```

```

void init(void);
void delay(unsigned long data);
void delay_ms(unsigned long data);
void beep(unsigned char count);

//LCD function prototype
void lcd_init(void); //initialize LCD
void lcd_config(unsigned char data); //lcd send config/command data
void lcd_char(unsigned char data); //lcd display single ASCII character
void e_pulse(void); //generate E pulse for lcd to read and process data
void lcd_goto(unsigned char data); //lcd, move cursor to lcd box, please refer to lcd address
void lcd_home(void); //lcd, move cursor to home (1st line, 1st column)
void lcd_2ndline(void); //lcd, move cursor to 2nd line
void lcd_clr(void); //clear lcd
void lcd_clr_line(unsigned char line); //clear single line on LCD
void lcd_string(const char* s); //lcd display string
void lcd_dis_num(unsigned char num_digit, unsigned int value); //lcd display number in decimal value

// modes
void display(void); //function for display direction and coordinate
void line_follow(void); //function for line following
//void ninety_degrid(void); //function for 90 degree grid function
void full_degrid(void); //function include 45, 90, 135 degree grid function

//mobile robot navigation control function
void forward(void);
void stop(void);
void backward(void);
void reverse(void);
void left(void);
void right(void);

//mobile robot movement function for 90 degree capabilities
void ahead(void); //function to move forward
void astern(void); //function to move backward
void starboard(void); //function to turn right
void port(void); //function to turn left
void halt(void); //function to stop
void rotate_right(void); //function to rotate right
void rotate_left(void); //function to rotate left
void overturn(void); //function to turn 180 degree to the right
void right_ninety(void); //function to turn 90 degree right
void left_ninety(void); //function to turn 90 degree left
void step(int *bot_x,int *bot_y); //function to move from/to each intersect
void right_45(void); //function to turn 45 degree right
void left_45(void); //function to turn 45 degree left
void right_135(void); //function to turn 135 degree right
void left_135(void); //function to turn 135 degree left
void follow_line(void); //function to follow any line existed

//ADC functions
void adc_init(void); //initialize ADC module
unsigned int read_adc(unsigned char channel); //read adc value and return

//timer1 functions
void timer1_init(void); //timer1 initialization
void interrupt_init(void); //interrupt initialization
void enable_global_int(void); //enable global interrupt
void disable_global_int(void); //disable global interrupt
unsigned int us_value(unsigned char mode); //function to read ultrasonic value from different input method
//ADC, PWM or UART(ASCII)

// interrupt service routine definition
//=====
void interrupt_isr(void)

```

```

{
static unsigned char i;
unsigned char receive_data = 0;
//RBIF set due to changes on RB4-RB7 pin, since only RB4 is input pin, RB4 changes
//To measure the pulse width of ultrasonic LVEZ1 PWM output
if(RBIF)
{
// _____
if(RB4 == 1) // RB4 is 1 mean is rising form 0 ___
{
TMR1H = 0; // clear timer 1 high byte
TMR1L = 0; // clear timer 1 low byte
TMR1ON = 1; // active timer 1
}
else
{
TMR1ON = 0; //deactive timer 1
pulse_width = TMR1H; // RB4 is 0 mean is falling edge, save the timer 1 register
pulse_width = (pulse_width << 8) + TMR1L; //combine the High byte and low byte of timer1
}
// _____ //
RBIF = 0; //reset the interrupt flag
}

if(RCIF) //for ultrasonic LVEZ1 UART output.
{
if(OERR == 1) //in case there is over run error
{
CREN = 0; //Reset the Receive engine
CREN = 1;
if(RCREG); //clear the RCREG
}
else
{
receive_data = RCREG; //store the received data
if (receive_data == 'R') data[i=0] = receive_data; // check if start byte of ASCII 'R', store 'R' to 1st byte
else if (data[0] == 'R') data [++i] = receive_data; // save the data in data array
}
}
}

// main function
//=====
void main(void)
{
unsigned char mode = 0, i = 0;
init(); // initiate cnfiguration and initial condition
lcd_init(); //initialize LCD
beep(2); // inditcate the program is running
LCD_BLIGHT = 1; //activate the LCD backlight
lcd_clr(); // clear the LCD screen
lcd_string(cytron_tech);
lcd_2ndline();
lcd_string(cytron_website);
delay_ms(800);

lcd_clr(); // clear the LCD screen
lcd_string(diy_project);
lcd_2ndline();
lcd_string(multifunction_robot);
delay_ms(800);

lcd_clr(); // clear the LCD screen

```



```

lcd_string(pr23_rev);
lcd_2ndline();
lcd_string(code_version);
delay_ms(800);

lcd_clr(); // clear the LCD screen
lcd_string(mode_string[mode]); // display string according to the mode
lcd_2ndline(); // move to 2nd line
lcd_string("SW1++ SW2->Run"); // display "select mode"
beep(1);

while(1) // infinite loop
{
    if(SW1 == 0) // if button SW1 is pressed
    {
        while(SW1 == 0); // wait for SW1 to be released
        beep(1);
        mode++; //increase mode value
        if (mode > 4) mode = 0; // if mode increased is more than 4, reset to zero
        lcd_clr_line(LINE1); //clear 1st line of LCD
        lcd_home(); // move LCD cursor back to home
        lcd_string(mode_string[mode]); // display string base on mode selected
    } //if(SW1 == 0)

    if(SW2 == 0) // if button SW2 is pressed
    {
        while(SW2 == 0); // wait until button is released
        beep(1);

        switch(mode) // check what is the current mode, execute the mode
        {
            case 0 :
                line_follow(); // mode 1 : line follow
                break;

            case 1 :
                //ninety_degrid(); // mode 2 : 90 degree grid
                //break;

            //case 2 :
                full_degrid(); // mode 3 : full degree grid
        } //switch case
    } //if(SW2 == 0)
} //while(1)
} //main()

/*****
//Delay function defination
void delay(unsigned long data) //delay function, the delay time
{
    for( ;data>0;data-=1); //depend on the given value
}

//delay in millisecond
void delay_ms(unsigned long data) //delay function, the delay time
{
    while(data -- > 0) //depend on the given value
        __delay_ms(1);
}

void beep(unsigned char count) //to sound buzzer
{
    while(count-- > 0)
    {

```

```

    BUZZER = 1;
    delay_ms(45);
    BUZZER = 0;
    delay_ms(30);
}
}
//=====
// LCD functions Definations
//=====
void lcd_init(void)
{
    LCD_E = 1;
    delay_ms(15); //delay 15ms for LCD to get ready
    lcd_config(0b00111000); //8-bit interface
    lcd_config(0b00000110); //entry mode-cursor increase 1
    lcd_config(0b00001100); //diplay on, cursor off and cursor blink off
    lcd_config(0b00000001); //clear display at lcd
    delay_ms(1);
}
void lcd_config(unsigned char data) //send lcd configuration
{
    LCD_RS = 0; //set lcd to config mode
    LCD_DATA = data; //lcd data port = data
    delay_ms(1);
    e_pulse(); //pulse e to confirm the data
}

void lcd_char(unsigned char data) //send lcd character
{
    LCD_RS = 1; //set lcd to display mode
    LCD_DATA = data; //lcd data port = data
    delay_ms(1);
    e_pulse(); //pulse e to confirm the data
}

void e_pulse(void) //pulse e to confirm the data
{
    LCD_E = 1;
    delay_ms(1);
    LCD_E = 0;
    delay_ms(1);
}

void lcd_goto(unsigned char data)//set the location of the lcd cursor
{
    lcd_config(0x80 + data);
    //address for LCD, in hexadecimal value
    // -----
    // | 00|01|02|03|04|05|06|07|08|09|0A|0B|0C|0D|0E|0F| |
    // | 40|41|42|43|44|45|46|47|48|49|4A|4B|4C|4D|4E|4F| |
    // -----
}
void lcd_home(void) //lcd, move cursor to home (1st line, 1st column)
{
    lcd_config(0x02);
}

void lcd_2ndline(void) //lcd, move cursor to 2nd line
{
    lcd_config(0x80 + 0x40);
}

void lcd_clr(void) //clear the lcd
{

```

```

    lcd_config(0x01);
    delay_ms(1);
}

void lcd_clr_line(unsigned char line) //clear single row on LCD
{
    unsigned char i = 0;
    if(line == LINE1)lcd_home(); //move cursor to 1st line, home
    else if(line == LINE2)lcd_2ndline(); //move cursor to 2nd line

    for(i = 16; i > 0; i--) lcd_char(' '); //display 16 x 'space' to clear single line on LCD
}
void lcd_string(const char* s) //send a string to display in the lcd
{
    while (s && *s)lcd_char (*s++);
}

void lcd_dis_num(unsigned char num_digit, unsigned int value)
{
    unsigned char digit[5] = {0}; //array to store digit value
    unsigned char i = 0, j = 0, non_zero = 0;
    unsigned int base = 10000;
    //loop to obtain 5 single digit from int value
    for(j = 4; j > 0; j--)
    {
        digit[j] = value / base;
        if(j == 1)
        {
            digit[0] = value % 10;
            continue;
        }
        value = value % base;
        base = base / 10;
    }

    //display the value on to LCD
    if(num_digit > 5) num_digit = 5;
    for(i = num_digit; i > 0; i--)
    {
        if(i == 1)
        {
            lcd_char(digit[i-1]+0x30);
        }
        else
        {
            if((digit[i-1] == 0) && (non_zero == 0))lcd_char(' '); //if zero display blank
            else
            {
                lcd_char(digit[i-1]+0x30);
                non_zero ++;
            }
        }
    }
} //for(i = num_digit; i > 0; i--)

// ADC functions definifation
void adc_init(void)//initialize ADC module
{
    // ADC configuration
    ADCON0 = 0b10000000; //conversion clock Fosc/32, channel 0, ADC is off
    ADCON1 = 0b10000100;//Configure RA0, RA1 and RA3 as Analog Input, right justified
    //The rest of AN pin is digital pin
}

```

```

//initialize timer1
void timer1_init(void)
{
    TMR1H = 0; //clear the timer 1 high byte value
    TMR1L = 0; //clear the timer 1 low byte value
    T1CON = 0b00100001; //prescaler of 1:4, internal clock source, timer 1 off
}

//function to initialize interrupt, basically to disable the interrupt used.
void interrupt_init(void)
{
    RCIE = 0; //disable UART receive interrupt
    RBIE = 0; //disable PORT B on change interrupt
    TMR1IE = 0; //disable Timer 1 overflow interrupt
}

//function to enable global and peripheral interrupt bits
void enable_global_int(void)
{
    GIE = 1; //enable global interrupt
    PEIE = 1; //enable peripheral interrupt
}

//function to disable global and peripheral interrupt bits
void disable_global_int(void)
{
    GIE = 0; //disable global interrupt
    PEIE = 0; //disable peripheral interrupt
}

//=====
// Initialization
// Description : Initialize the microcontroller
//=====
void init()
{
    PORTA = 0;
    PORTB = 0;
    PORTC = 0;
    PORTD = 0;
    PORTE = 0;

    // Tris configuration (input or output)
    TRISA = 0b00110011; //set RA0 and RA2 pin as input, other as output
                        //PR23 Rev2.0 has RA4 and RA5 as input for IR01A
    TRISB = 0b00011111; //set RB0-RB4 pin as input, other as output
    TRISC = 0b10000000; //set PORTC pin as output, RC7 is UART Receive pin (input)
    TRISD = 0b00000000; //set all PORTD pin as output
    TRISE = 0b00000011; //RE0 and RE1 as input (Switches) RE2 as output (LED)

    // initialize ADC module
    adc_init();
    interrupt_init(); //initialize interrupt

    // motor PWM configuration
    PR2 = 255; // set period register for PWM
    T2CON = 0b00000100; // Timer Control register, timer 2 ON, prescaler = 1:1
    CCP1CON = 0b00001100; // config for RC1 to generate PWM( for more detail refer datasheet section 'capture/compare/pwm')
    CCP2CON = 0b00001100; // config for RC2 to generate PWM
    SPEEDL = 0; //initial PWM is zero
    SPEEDR = 0; //initial PWM is zero

    disable_global_int(); //disable global interrupt
    lcd_init(); //initialize LCD
}

```

```

stop(); //motors are off
}

//=====
// Mode subroutine
//=====
// Mode 1 : line follow subroutine
// Description: Program for the mobile robot to follow line
// For more details about line follow concept please PR23 Detailed Description
//=====
void line_follow()
{
    unsigned char memory = 0; //variable to memorize previous condition if sensor is out of line

    lcd_clr(); // clear lcd screen
    lcd_string("Line Position"); // display "position" string

    //When sensor senses line (black) it will get logic 1 (5V or HIGH) at PIC pin.
    while(1) //infinite loop
    {
        forward(); //mobile robot will move forward
        if((SEN_L==1)&&(SEN_ML==0)&&(SEN_MR==0)&&(SEN_R==0)) // if only sensor left detected black line
        {
            SPEEDL = 0; //left motor stop
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 1; //1 = line is at left of mobile robot
            lcd_2ndline(); // lcd go to 2nd line 1st character
            lcd_string("right "); // display "right"mean the robot's position is on the right side of the line
        }
        else if((SEN_L==1)&&(SEN_ML==1)&&(SEN_MR==0)&&(SEN_R==0)) // if only sensor left detected black line
        {
            SPEEDL = 180; // left motor speed is 180
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 1; //1 = line is at left of mobile robot
            lcd_2ndline();
            lcd_string("m_right2");
        }
        else if((SEN_L==0)&&(SEN_ML==1)&&(SEN_MR==0)&&(SEN_R==0)) // if only sensor middle left detected black line
        {
            SPEEDL = 200; // left motor speed is 200
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 1; //1 = line is at left of mobile robot
            lcd_2ndline();
            lcd_string("m_right1 ");
        }
        else if((SEN_L==1)&&(SEN_ML==1)&&(SEN_MR==1)&&(SEN_R==0)) // if sensor middle left, middle right
            //and sensor left detected black line
        {
            SPEEDL = 200; // left motor speed is 200
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 1; //1 = line is at left of mobile robot
            lcd_2ndline();
            lcd_string("m_right1 ");
        }
        else if((SEN_L==0)&&(SEN_ML==1)&&(SEN_MR==1)&&(SEN_R==0)) // if sensor middle left and sensor middle right detecte
        {
            SPEEDL = 255; // left motor speed is 255(full speed)
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 2; //32 = line is at middle of mobile robot
            lcd_2ndline();
            lcd_string("middle ");
        }
        else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==1)&&(SEN_R==0)) // if only sensor middle right detected black line
        {

```

```

    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 200; // right motor speed is 200
    memory = 3; //3 = line is at right of mobile robot
    lcd_2ndline();
    lcd_string("m_left1 ");
}
else if((SEN_L==0)&&(SEN_ML==1)&&(SEN_MR==1)&&(SEN_R==1)) // if sensor middle left, sensor middle right and sensor r
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 200; // right motor speed is 200
    memory = 3; //3 = line is at right of mobile robot
    lcd_2ndline();
    lcd_string("m_left1 ");
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==1)&&(SEN_R==1)) // if sensor right and sensor middle right detected black
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 180; // right motor speed is 180
    memory = 3; //3 = line is at right of mobile robot
    lcd_2ndline();
    lcd_string("m_left2 ");
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==0)&&(SEN_R==1)) // if only sensor right detected black line
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 0; // right motor speed is 0
    memory = 3; //3 = line is at right of mobile robot
    lcd_2ndline();
    lcd_string("left ");
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==0)&&(SEN_R==0)) // if all sensor could not detected black line
{
    if(memory == 1 )
    {
        SPEEDL = 0; // left motor speed is 0
        SPEEDR = 255; // right motor speed is 255(full speed)
    }
    else if(memory == 3)
    {
        SPEEDL = 255; // left motor speed is 255(full speed)
        SPEEDR = 0; // right motor speed is 0
    }
}
}
} //while(1)
}

//=====
// Motor control function
// Description : subroutine to set the robot moving direction
//=====
void forward () //function to enable robot to move forward, do not change the speed
{
    MOTOR_R1 = 0;
    MOTOR_R2 = 1;
    MOTOR_L1 = 0;
    MOTOR_L2 = 1;
}

void backward () //function to enable robot to move backward, do not change the speed
{
    MOTOR_R1 = 1;
    MOTOR_R2 = 0;
    MOTOR_L1 = 1;
    MOTOR_L2 = 0;
}

```

```

}

void left() //function to enable robot to turn left, do not change the speed
{
    MOTOR_R1 = 0;
    MOTOR_R2 = 1;
    MOTOR_L1 = 1;
    MOTOR_L2 = 0;
}

void right() //function to enable robot to turn right, do not change the speed
{
    MOTOR_R1 = 1;
    MOTOR_R2 = 0;
    MOTOR_L1 = 0;
    MOTOR_L2 = 1;
}

void stop() //function to enable robot to stop, do not change the speed
{
    MOTOR_R1 = 0;
    MOTOR_R2 = 0;
    MOTOR_L1 = 0;
    MOTOR_L2 = 0;
}

//=====
// Mode 2 : 90 degree grid following subroutine
// Description: Program for the mobile robot to follow 90 degree grid
//=====

void full_degrid() //function to 'call' and perform 90 degree grid following ability
{
    lcd_clr();
    bot_dir=1;
    bot_y=0;
    bot_x=0;
    //grid_y=4; //90 degrid size
    //grid_x=4;
    grid_y=8; //full degrid size
    grid_x=8;

    /*****using below function to display LOVE symbol accordingly*****/
    /*
    while(1)
    {
        while(bot_y < 6) //while Y-coordinate less than 6, goto intesection to intersection
        {
            step(&bot_x,&bot_y);
            display();
        }
        if(bot_x==0 && bot_y==6) //if X and Y coordinate = (0,6) then turn 45 degree right
            right_45();
        while(bot_x < 2 && bot_y < 8) //while X-coordinate less than 2, Y-coordinate less than 8
        {
            step(&bot_x,&bot_y);
            display();
        }
        if(bot_x==2 && bot_y==8) //if X and Y coordinate = (2,8) then turn 45 degree right
            right_45();
        while(bot_x < 4 && bot_y==8) //while X-coordinate less than 4, goto intesection to intersection
        {
            step(&bot_x,&bot_y);
            display();
        }
    }
    */
}

```

```

}
if(bot_x==4 && bot_y==8) //if X and Y coordinate = (4,8) then turn 90 degree right
    right_ninety();
while(bot_x == 4 && bot_y > 4) //while Y-coordinate more than 4, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==4 && bot_y==4) //if X and Y coordinate = (4,4) then turn 90 degree left
    left_ninety();
while(bot_x < 8 && bot_y==4) //while X-coordinate less than 8, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==8 && bot_y==4) //if X and Y coordinate = (8,4) then turn 90 degree right
    right_ninety();
while(bot_x==8 && bot_y > 2) //while Y-coordinate less than 2, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==8 && bot_y==2) //if X and Y coordinate = (8,2) then turn 45 degree right
    right_45();
while(bot_x > 6 && bot_y > 0) //while X-coordinate more than 6, Y-coordinate more than 0
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==6 && bot_y==0) //if X and Y coordinate = (6,0) then turn 45 degree right
    right_45();
while(bot_x > 0 && bot_y==0) //while X-coordinate more than 0, Y-coordinate equal 0
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==0 && bot_y==0) //if X and Y coordinate = (0,0) then turn 90 degree right
    right_ninety();
}
*/
/*****

/*****using below function to test changing direction function*****/
/*
while(1)
{
    //right_45(); //test changing direction
    //left_45();
    //right_135();
    //left_135();
    //overturn();
}
*/
/*****

/*****using below function for 45/135 degree grid only function*****/
/*
while(1)
{
    while(bot_y < 4) //while Y-coordinate less than 4, goto intesection to intersection
    {
        step(&bot_x,&bot_y);
        display();
    }
}

```



```

if(bot_x==0 && bot_y==4) //if X and Y coordinate = (0,4) then turn 90 degree right
    right_ninety();
while(bot_x < 4) //while X-coordinate less than 4, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==4 && bot_y==4) //if X and Y coordinate = (4,4) then turn 45 degree left
    left_45();
while(bot_x < 8 && bot_y < 8) //while X & Y-coordinate less than 8, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==grid_x && bot_y==grid_y) //if X and Y coordinate = (8,8) then turn 135 degree right
    right_135();
while(bot_y > 0) //while Y-coordinate more than 0, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==grid_x && bot_y==0) //if X and Y coordinate = (8,0) then turn 90 degree right
    right_ninety();
while(bot_x > 0) //while X-coordinate more than 0, goto intesection to intersection
{
    step(&bot_x,&bot_y);
    display();
}
if(bot_x==0 && bot_y==0) //if X and Y coordinate = (0,0) then turn 90 degree right
    right_ninety();
} //while(1) loop
*/
/*****
/*****using below function for 90 degree grid only function*****/

while(1)
{
    while(bot_y < grid_y) //while Y-coordinate less than 4, goto intesection to intersection
    {
        step(&bot_x,&bot_y);
        display();
    }
    if(bot_x==0 && bot_y==grid_y) //if X and Y coordinate = (0,4) then turn 90 degree right
        right_ninety();
    while(bot_x < grid_x) //while X-coordinate less than 4, goto intesection to intersection
    {
        step(&bot_x,&bot_y);
        display();
    }
    if(bot_x==grid_x && bot_y==grid_y) //if X and Y coordinate = (4,4) then turn 90 degree right
        right_ninety();
    while(bot_y > 0) //while Y-coordinate more than 0, goto intesection to intersection
    {
        step(&bot_x,&bot_y);
        display();
    }
    if(bot_x==grid_x && bot_y==0) //if X and Y coordinate = (4,0) then turn 90 degree right
        right_ninety();
    while(bot_x > 0) //while X-coordinate more than 0, goto intesection to intersection
    {
        step(&bot_x,&bot_y);
        display();
    }
}

```

```

    if(bot_x==0 && bot_y==0)    //if X and Y coordinate = (0,0) then turn 90 degree right
        right_ninety();
} //while(1) loop

/*****

/*****using below function for testing and demonstration purposes*****/
/* while(1)
{
    step(&bot_x, &bot_y); //move from intersection to intersection
    //left_ninety();
    right_ninety();    //turn 90 degree right
    lcd_2ndline();    //clear 2nd line of LCD
    lcd_dis_num(1, bot_x); //display X-coordinate on LCD
    lcd_dis_num(3, bot_y); //display X-coordinate on LCD
    lcd_char(' ');
}
*/
/*****
}
//=====
//Function and subroutine
//=====

//function to display coordinate and direction
void display(void)
{
    lcd_2ndline();
    lcd_dis_num(1, bot_x); //display distance on LCD
    lcd_dis_num(3, bot_y);
    lcd_char(' ');
    return;
}

//functions to move the robot
void ahead()    //function to move forward
{
    forward();    //mobile robot will move forward
    SPEEDR = 255; //right motor speed is 255
    SPEEDL = 255; //left motor speed is 255
}
void astern()    //function to move backward
{
    backward();    //mobile robot will move backward
    SPEEDR = 255; //right motor speed is 255
    SPEEDL = 255; //left motor speed is 255
}
void starboard()    //function to move right
{
    right();    //mobile robot will turn right
    SPEEDR = 0; //right motor speed is 0
    SPEEDL = 255; //left motor speed is 230
}
void port()    //function to move left
{
    left();    //mobile robot will turn left
    SPEEDR = 255; //right motor speed is 230
    SPEEDL = 0; //left motor speed is 0
}
void rotate_right()    //function to rotate 90 degree right
{
    right();    //mobile robot will rotate right
    SPEEDR = 230; //right motor speed is 230
    SPEEDL = 230; //left motor speed is 230
}

```

```

}
void rotate_left() //function to rotate 90 degree left
{
    left(); //mobile robot will rotate left
    SPEEDR = 230; //right motor speed is 230
    SPEEDL = 230; //left motor speed is 230
}
void halt()
{
    stop(); //mobile robot will stop
    SPEEDR = 0; //right motor speed is 0
    SPEEDL = 0; //left motor speed is 0
}
//function for overturn 180 degree to the right
void overturn(void)
{
    //ahead(); //forward delay 400ms
    //delay_ms(400); // 90 degrid delay - before rotate
    delay_ms(550); //full degrid delay - before rotate
    rotate_right(); //rotate to the right
    delay_ms(1400); //delay time for rotating
    while(SEN_R==0) //while most right sensor still not detect line, do function above
        right(); //turn to the right a bit to align sensors and tyre to the line
    delay_ms(50); //delay for turning right
    halt(); //stop after detect line
    if(bot_dir==N) //if initial direction is North
    {
        bot_dir = S; //then change to South
        lcd_clr();
        lcd_home();
        lcd_string("S"); //display S
        return;
    }
    if(bot_dir==E) //if initial direction is East
    {
        bot_dir = W; //then change to West
        lcd_clr();
        lcd_home();
        lcd_string("W"); //display W
        return;
    }
    if(bot_dir==S) //if initial direction is South
    {
        bot_dir = N; //then change to North
        lcd_clr();
        lcd_home();
        lcd_string("N"); //display N
        return;
    }
    if(bot_dir==W) //if initial direction is West
    {
        bot_dir = E; //then change to East
        lcd_clr();
        lcd_home();
        lcd_string("E"); //display E
        return;
    }
    if(bot_dir==NE) //if initial direction is NorthEast
    {
        bot_dir = SW; //then change to SouthWest
        lcd_clr();
        lcd_home();
        lcd_string("SW"); //display SW
        return;
    }
}

```

```

}
if(bot_dir==SE) //if initial direction is SouthEast
{
  bot_dir = NW; //then change to NorthWest
  lcd_clr();
  lcd_home();
  lcd_string("NW"); //display NW
  return;
}
if(bot_dir==SW) //if initial direction is SouthWest
{
  bot_dir = NE; //then change to NorthEast
  lcd_clr();
  lcd_home();
  lcd_string("NE"); //display NE
  return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
  bot_dir = SE; //then change to SouthEast
  lcd_clr();
  lcd_home();
  lcd_string("SE"); //display SE
  return;
}
}

/*****These are subfunction for 90 degree turn right and left*****/
//function for turn 90 degree right

void right_ninety()
{
  ahead(); //move small forward before rotating
  //delay_ms(400); // 90 degrid delay - before rotate
  delay_ms(550); //full degrid delay - before rotate
  rotate_right(); //rotate the robot to the right
  delay_ms(650); //delay time for turning robot
  right(); //robot turn right abit to align sensor and tyres to the line
  delay_ms(50); //delay time for turning right
  while(SEN_MR==0) //while most right sensor still not detect line, do function above
    halt(); //stop after detect line
  if(bot_dir==N) //change robot's direction
  {
    //if initial direction is North
    bot_dir = E; //then change to East
    lcd_clr();
    lcd_home();
    lcd_string("E"); //display E
    return; //return to main function
  }
  if(bot_dir==E) //if initial direction is East
  {
    bot_dir = S; //then change to South
    lcd_clr();
    lcd_home();
    lcd_string("S"); //display S
    return;
  }
  if(bot_dir==S) //if initial direction is South
  {
    bot_dir = W; //then change to West
    lcd_clr();
    lcd_home();
    lcd_string("W"); //display W
    return;
  }
}

```

```

}
if(bot_dir==W) //if initial direction is West
{
  bot_dir = N; //then change to North
  lcd_clr();
  lcd_home();
  lcd_string("N"); //display N
  return;
}
if(bot_dir==NE) //change robot's direction
{
  //if initial direction is NorthEast
  bot_dir = SE; //then change to SouthEast
  lcd_clr();
  lcd_home();
  lcd_string("SE"); //display SE
  return; //return to main function
}
if(bot_dir==SE) //if initial direction is SouthEast
{
  bot_dir = SW; //then change to SouthWest
  lcd_clr();
  lcd_home();
  lcd_string("SW"); //display SW
  return;
}
if(bot_dir==SW) //if initial direction is SouthWest
{
  bot_dir = NW; //then change to NorthWest
  lcd_clr();
  lcd_home();
  lcd_string("NW"); //display NW
  return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
  bot_dir = NE; //then change to NorthEast
  lcd_clr();
  lcd_home();
  lcd_string("NE"); //display NE
  return;
}
}

//function for turn 90 degree left
void left_ninety()
{
  ahead(); //move small forward before rotating
  //delay_ms(400); // 90 degrid delay - before rotate
  delay_ms(550); //full degrid delay - before rotate
  rotate_left(); //rotate the robot to the left
  delay_ms(650); //delay time for turning robot
  left(); //robot turn left abit to detect sensor
  delay_ms(50); //delay time for turning left
  while(SEN_ML==0) //while most left sensor still not detect line, do function above
  {
    halt(); //stop after detect line
  }
  if(bot_dir==N) //change robot's direction
  {
    //if initial direction is North
    bot_dir = W; //then change to West
    lcd_clr();
    lcd_home();
    lcd_string("W"); //display W
    return; //return to main function
  }
}
if(bot_dir==E) //if initial direction is East

```

```

{
    bot_dir = N; //then change to North
    lcd_clr();
    lcd_home();
    lcd_string("N"); //display N
    return;
}
if(bot_dir==S) //if initial direction is South
{
    bot_dir = E; //then change to East
    lcd_clr();
    lcd_home();
    lcd_string("E"); //display E
    return;
}
if(bot_dir==W) //if initial direction is West
{
    bot_dir = S; //then change to South
    lcd_clr();
    lcd_home();
    lcd_string("S"); //display S
    return;
}
if(bot_dir==NE) //change robot's direction
{
    //if initial direction is NorthEast
    bot_dir = NW; //then change to NorthWest
    lcd_clr();
    lcd_home();
    lcd_string("NW"); //display NW
    return; //return to main function
}
if(bot_dir==SE) //if initial direction is SouthEast
{
    bot_dir = NE; //then change to NorthEast
    lcd_clr();
    lcd_home();
    lcd_string("NE"); //display NE
    return;
}
if(bot_dir==SW) //if initial direction is SouthWest
{
    bot_dir = SE; //then change to SouthEast
    lcd_clr();
    lcd_home();
    lcd_string("SE"); //display SE
    return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
    bot_dir = SW; //then change to SouthWest
    lcd_clr();
    lcd_home();
    lcd_string("SW"); //display SW
    return;
}
}
//function for movement from intersection to intersection
void step(int *bot_x,int *bot_y)
{
    ahead(); //move ahead to avoid sensors fluctuation
    //delay_ms(150); //90 degrid delay - step fluctuate
    delay_ms(400); //full degrid delay - step fluctuate
    while(1) //keep following line until condition below satisfied
    {

```

```

follow_line(); //if detected either one, do the follow_line function
if (SEN_L==1 && SEN_R==1) //if either sensor is detected, then junction detected
break; //break the while(1) loop
}
halt(); //stop and increase robot coordinate accordingly as below
if(bot_dir==N) //if the direction of movement is to the North
{
//++*bot_y; //then increase y coordinate by 1 (90degrid)
*bot_y+=2; //then increase y by 2 (fulldgrid)
}

if(bot_dir==S) //if the direction of movement is to the South
{
/--*bot_y; //then decrease y coordinate by 1 (90degrid)
*bot_y-=2; //then decrease y by 2 (fulldgrid)
}

if(bot_dir==E) //if the direction of movement is to the East
{
//++*bot_x; //then increase x coordinate by 1 (90degrid)
*bot_x+=2; //then increase x by 2 (fulldgrid)
}

if(bot_dir==W) //if the direction of movement is to the West
{
/--*bot_x; //then decrease x coordinate by 1 (90degrid)
*bot_x-=2; //then decrease y by 2 (fulldgrid)
}

if(bot_dir==NE) //if the direction of movement is to the NorthEast
{
++*bot_x; //then increase x coordinate by 1 (45/135degrid)
++*bot_y; //then increase y coordinate by 1 (45/135degrid)
//*bot_y+=2; //then increase y by 2 (fulldgrid)
}

if(bot_dir==SE) //if the direction of movement is to the SouthEast
{
++*bot_x; //then increase x coordinate by 1 (45/135degrid)
--*bot_y; //then decrease y coordinate by 1 (45/135degrid)
//*bot_y-=2; //then decrease y by 2 (fulldgrid)
}

if(bot_dir==SW) //if the direction of movement is to the SouthWest
{
--*bot_x; //then increase x coordinate by 1 (45/135degrid)
--*bot_y; //then increase y coordinate by 1 (45/135degrid)
//*bot_x+=2; //then increase y by 2 (fulldgrid)
}

if(bot_dir==NW) //if the direction of movement is to the NorthWest
{
--*bot_x; //then decrease x coordinate by 1 (45/135degrid)
++*bot_y; //then increase y coordinate by 1 (45/135degrid)
//*bot_x-=2; //then decrease y by 2 (fulldgrid)
}
}

/*****These are subfunction for 45 degree turn right and left*****/
//function for turn 45 degree right
void right_45()
{
ahead(); //move small forward before rotating
//delay_ms(400); // 90 degrid delay - before rotate
delay_ms(550); //full degrid delay - before rotate
rotate_right(); //rotate the robot to the right
delay_ms(325); //delay time for turning robot
//right(); //robot turn right abit to align sensor and tyres to the line
//delay_ms(50); //delay time for turning right
}

```

```

while(SEN_MR==0) //while most right sensor still not detect line, do function above
  halt(); //stop after detect line
if(bot_dir==N) //change robot's direction
{
  //if initial direction is North
  bot_dir = NE; //then change to NorthEast
  lcd_clr();
  lcd_home();
  lcd_string("NE"); //display NE
  return; //return to main function
}
if(bot_dir==E) //if initial direction is East
{
  bot_dir = SE; //then change to SouthEast
  lcd_clr();
  lcd_home();
  lcd_string("SE"); //display SE
  return;
}
if(bot_dir==S) //if initial direction is South
{
  bot_dir = SW; //then change to SouthWest
  lcd_clr();
  lcd_home();
  lcd_string("SW"); //display SW
  return;
}
if(bot_dir==W) //if initial direction is West
{
  bot_dir = NW; //then change to NorthWest
  lcd_clr();
  lcd_home();
  lcd_string("NW"); //display NW
  return;
}
if(bot_dir==NE) //if initial direction is NorthEast
{
  bot_dir = E; //then change to East
  lcd_clr();
  lcd_home();
  lcd_string("E"); //display E
  return; //return to main function
}
if(bot_dir==SE) //if initial direction is SouthEast
{
  bot_dir = S; //then change to South
  lcd_clr();
  lcd_home();
  lcd_string("S"); //display S
  return;
}
if(bot_dir==SW) //if initial direction is SouthWest
{
  bot_dir = W; //then change to West
  lcd_clr();
  lcd_home();
  lcd_string("W"); //display W
  return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
  bot_dir = N; //then change to North
  lcd_clr();
  lcd_home();
  lcd_string("N"); //display N
}

```



```

    return;
}
}
//function for turn 45 degree left
void left_45()
{
    ahead(); //move small forward before rotating
    //delay_ms(400); // 90 degrid delay - before rotate
    delay_ms(550); //full degrid delay - before rotate
    rotate_left(); //rotate the robot to the left
    delay_ms(325); //delay time for turning robot
    //left(); //robot turn left abit to detect sensor
    //delay_ms(50); //delay time for turning left
    while(SEN_ML==0) //while most left sensor still not detect line, do function above
        halt(); //stop after detect line
    if(bot_dir==N) //change robot's direction
    {
        //if initial direction is North
        bot_dir = NW; //then change to NorthWest
        lcd_clr();
        lcd_home();
        lcd_string("NW"); //display NW
        return; //return to main function
    }
    if(bot_dir==E) //if initial direction is East
    {
        bot_dir = NE; //then change to NorthEast
        lcd_clr();
        lcd_home();
        lcd_string("NE"); //display NE
        return;
    }
    if(bot_dir==S) //if initial direction is South
    {
        bot_dir = SE; //then change to SouthEast
        lcd_clr();
        lcd_home();
        lcd_string("SE"); //display SE
        return;
    }
    if(bot_dir==W) //if initial direction is West
    {
        bot_dir = SW; //then change to SouthWest
        lcd_clr();
        lcd_home();
        lcd_string("SW"); //display SW
        return;
    }
    if(bot_dir==NE) //change robot's direction
    {
        //if initial direction is NorthEast
        bot_dir = N; //then change to North
        lcd_clr();
        lcd_home();
        lcd_string("N"); //display N
        return; //return to main function
    }
    if(bot_dir==SE) //if initial direction is SouthEast
    {
        bot_dir = E; //then change to East
        lcd_clr();
        lcd_home();
        lcd_string("E"); //display E
        return;
    }
    if(bot_dir==SW) //if initial direction is SouthWest

```

```

{
  bot_dir = S; //then change to South
  lcd_clr();
  lcd_home();
  lcd_string("S"); //display S
  return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
  bot_dir = W; //then change to West
  lcd_clr();
  lcd_home();
  lcd_string("W"); //display W
  return;
}
}

/*****These are subfunction for 135 degree turn right and left*****/
//function for turn 135 degree right
void right_135()
{
  ahead(); //move small forward before rotating
  //delay_ms(400); // 90 degrid delay - before rotate
  delay_ms(550); //full degrid delay - before rotate
  rotate_right(); //rotate the robot to the right
  delay_ms(975); //delay time for turning robot
  right(); //robot turn right abit to align sensor and tyres to the line
  delay_ms(50); //delay time for turning right
  while(SEN_MR==0) //while most right sensor still not detect line, do function above
  {
    halt(); //stop after detect line
  }
  if(bot_dir==N) //change robot's direction
  {
    //if initial direction is North
    bot_dir = SE; //then change to SouthEast
    lcd_clr();
    lcd_home();
    lcd_string("SE"); //display SE
    return; //return to main function
  }
  if(bot_dir==E) //if initial direction is East
  {
    bot_dir = SW; //then change to SouthWest
    lcd_clr();
    lcd_home();
    lcd_string("SW"); //display SW
    return;
  }
  if(bot_dir==S) //if initial direction is South
  {
    bot_dir = NW; //then change to NorthWest
    lcd_clr();
    lcd_home();
    lcd_string("NW"); //display NW
    return;
  }
  if(bot_dir==W) //if initial direction is West
  {
    bot_dir = NE; //then change to NorthEast
    lcd_clr();
    lcd_home();
    lcd_string("NE"); //display NE
    return;
  }
  if(bot_dir==NE) //if initial direction is NorthEast
  {

```

```

    bot_dir = S; //then change to South
    lcd_clr();
    lcd_home();
    lcd_string("S"); //display S
    return; //return to main function
}
if(bot_dir==SE) //if initial direction is SouthEast
{
    bot_dir = W; //then change to West
    lcd_clr();
    lcd_home();
    lcd_string("W"); //display W
    return;
}
if(bot_dir==SW) //if initial direction is SouthWest
{
    bot_dir = N; //then change to North
    lcd_clr();
    lcd_home();
    lcd_string("N"); //display N
    return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
    bot_dir = E; //then change to East
    lcd_clr();
    lcd_home();
    lcd_string("E"); //display E
    return;
}
}
//function for turn 135 degree left
void left_135()
{
    ahead(); //move small forward before rotating
    //delay_ms(400); // 90 degrid delay - before rotate
    delay_ms(550); //full degrid delay - before rotate
    rotate_left(); //rotate the robot to the left
    delay_ms(975); //delay time for turning robot
    left(); //robot turn left abit to detect sensor
    delay_ms(50); //delay time for turning left
    while(SEN_ML==0) //while most left sensor still not detect line, do function above
        halt(); //stop after detect line
    if(bot_dir==N) //change robot's direction
    {
        //if initial direction is North
        bot_dir = SW; //then change to SouthWest
        lcd_clr();
        lcd_home();
        lcd_string("SW"); //display SW
        return; //return to main function
    }
    if(bot_dir==E) //if initial direction is East
    {
        bot_dir = NW; //then change to NorthWest
        lcd_clr();
        lcd_home();
        lcd_string("NW"); //display NW
        return;
    }
    if(bot_dir==S) //if initial direction is South
    {
        bot_dir = NE; //then change to NorthEast
        lcd_clr();
        lcd_home();
    }
}

```

```

    lcd_string("NE"); //display NE
    return;
}
if(bot_dir==W) //if initial direction is West
{
    bot_dir = SE; //then change to SouthEast
    lcd_clr();
    lcd_home();
    lcd_string("SE"); //display SE
    return;
}
if(bot_dir==NE) //change robot's direction
{
    //if initial direction is NorthEast
    bot_dir = W; //then change to West
    lcd_clr();
    lcd_home();
    lcd_string("W"); //display W
    return; //return to main function
}
if(bot_dir==SE) //if initial direction is SouthEast
{
    bot_dir = N; //then change to North
    lcd_clr();
    lcd_home();
    lcd_string("N"); //display N
    return;
}
if(bot_dir==SW) //if initial direction is SouthWest
{
    bot_dir = E; //then change to East
    lcd_clr();
    lcd_home();
    lcd_string("E"); //display E
    return;
}
if(bot_dir==NW) //if initial direction is NorthWest
{
    bot_dir = S; //then change to South
    lcd_clr();
    lcd_home();
    lcd_string("S"); //display S
    return;
}
}

// function for follow any line detected
void follow_line()
{
    unsigned char memory = 0; //variable to memorize previous condition if sensor is out of line
    {
        forward(); //mobile robot will move forward
        if((SEN_L==1)&&(SEN_ML==0)&&(SEN_MR==0)&&(SEN_R==0)) // if only sensor left detected black line
        {
            SPEEDL = 0; //left motor stop
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 1; //1 = line is at left of mobile robot
        }
        else if((SEN_L==1)&&(SEN_ML==1)&&(SEN_MR==0)&&(SEN_R==0)) // if only sensor left detected black line
        {
            SPEEDL = 180; // left motor speed is 180
            SPEEDR = 255; // right motor speed is 255(full speed)
            memory = 1; //1 = line is at left of mobile robot
        }
        else if((SEN_L==0)&&(SEN_ML==1)&&(SEN_MR==0)&&(SEN_R==0)) // if only sensor middle left detected black line
    }
}

```

```

{
    SPEEDL = 200; // left motor speed is 200
    SPEEDR = 255; // right motor speed is 255(full speed)
    memory = 1; //1 = line is at left of mobile robot
}
else if((SEN_L==1)&&(SEN_ML==1)&&(SEN_MR==1)&&(SEN_R==0)) // if sensor middle left, middle right and sensor left detect
{
    SPEEDL = 200; // left motor speed is 200
    SPEEDR = 255; // right motor speed is 255(full speed)
    memory = 1; //1 = line is at left of mobile robot
}
else if((SEN_L==0)&&(SEN_ML==1)&&(SEN_MR==1)&&(SEN_R==0)) // if sensor middle left and sensor middle right detecte
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 255; // right motor speed is 255(full speed)
    memory = 2; //2 = line is at middle of mobile robot
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==1)&&(SEN_R==0)) // if only sensor middle right detected black line
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 200; // right motor speed is 200
    memory = 3; //3 = line is at right of mobile robot
}
else if((SEN_L==0)&&(SEN_ML==1)&&(SEN_MR==1)&&(SEN_R==1)) // if sensor middle left, sensor middle right and sensor r
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 200; // right motor speed is 200
    memory = 3; //3 = line is at right of mobile robot
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==1)&&(SEN_R==1)) // if sensor right and sensor middle right detected black
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 180; // right motor speed is 180
    memory = 3; //3 = line is at right of mobile robot
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==0)&&(SEN_R==1)) // if only sensor right detected black line
{
    SPEEDL = 255; // left motor speed is 255(full speed)
    SPEEDR = 0; // right motor speed is 0
    memory = 3; //3 = line is at right of mobile robot
}
else if((SEN_L==0)&&(SEN_ML==0)&&(SEN_MR==0)&&(SEN_R==0)) // if all sensor could not detected black line
{
    if(memory == 1 )
    {
        SPEEDL = 0; // left motor speed is 0
        SPEEDR = 255; // right motor speed is 255(full speed)
    }
    else if(memory == 3)
    {
        SPEEDL = 255; // left motor speed is 255(full speed)
        SPEEDR = 0; // right motor speed is 0
    }
}
}
}
}
}

```