MAZE ROBOT: APPLYING AUTONOMOUS VEHICLE NAVIGATION ALGORITHM WITH EVENT-DRIVEN PROGRAMMING

By

NADHIRA BINTI ABDUL MALEK

DISSERTATION

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

MAZE ROBOT: APPLYING AUTONOMOUS VEHICLE NAVIGATION ALGORITHM WITH EVENT-DRIVEN PROGRAMMING

by

Nadhira binti Abdul Malek

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

[illegible]

Mr. Abu Bakar Sayuti Hj. Mohd Saman
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

May 2011

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 except 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.

[illegible]

Nadhira Abdul Malek

ABSTRACT

Autonomous navigation is an eminent feature in robotics as it provides mobile robot with the ability to traverse from one point to another point while avoiding any obstacles that lie within its path. To navigate through a maze with unpredictable routes would be a great challenge as it requires the assistance of an intelligent algorithm. The main objective of this project is to build and program a mini mobile robot that is able to autonomously navigate through a physical maze. The physical maze will comprise of several different configurations to measure the efficiency of the robot. Hardware and software co-design method is used to construct the mobile robot. The basic navigation algorithm was developed using finite state machine (FSM). Event-driven programming method was applied in producing the maze navigation algorithm for the robot.

ACKNOWLEDGEMENTS

Firstly, I would like to express many thanks to my supervisor, Mr. Abu Bakar Sayuti for the many guidance and help. And not forgetting, thank you also for being very patience with me throughout the whole process of completing this project.

I would also want to thank my greatest motivator, my family who has always been there with endless encouragements and positive remarks. I appreciate the financial support and the love that comes with it.

Not forgetting my friends who have made it easier to complete this project. My deepest gratitude goes to my fellow friends who have been patience and helpful. Those advices and tips have been of great help and of great use.

Thank you also to the staff and technicians of UTP for the kind and friendly help in getting some of the components and materials together. Lastly, I direct my final thanks to those who have indirectly helped me to finish this project. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

FSM	Finite State Machine
LED	Light Emitting Diode
CPU	Central Processing Unit
RAM	Random-Access Memory
ROM	Read-Only Memory
EEPROM	Electrically Erasable Programmable Read-only Memory
I/O	Input Output
PIC	Peripheral Interface Controller
PWM	Pulse Width Modulation
IDE	Integrated Development Environment
GUI	Graphic User Interface
IC	Integrated Circuit
A/D	Analogue Digital
IR	Infrared

CHAPTER 1
PROJECT BACKGROUND

1.1 Background of Study

Mobile robots are gaining momentum in the current research as it could be of great use to the presently growing technology. They are widely used for many different purposes in various fields. As technology expands, mobile robots are being studied further from merely able to detect physical properties into smarter robots with more advanced features such as navigation ability.

Autonomous navigation is normally tied to robots where it is basically the ability of a device to perform tasks without constant guidance from humans. There are many levels of autonomy which varies depending on the type of application it is used for. Several examples where autonomous navigation can be found are in fields such as space exploration, floor cleaning, lawn mowing, and waste water treatment. One of the most important elements in autonomous navigation robotics is the ability of the robot to interact with its environment.

Basically, Maze Robot is a Microchip's PIC microcontroller-based mobile robot which is able to navigate its way successfully through a physical maze. The outer part of the physical maze would have two openings which would act as the entrance and the exit for the robot. While inside the maze there would be some dividing walls to test out the robot intelligence in finding its way out. Therefore, the main objective is to have the robot go through the entrance and navigate through the many paths and junctions in the maze until it finds the exit point.

Apart from that, the PIC microcontroller plays the most important role in the project. It is loaded with appropriate coding which acts as the main essence for the robot functionality of finding its way out of the maze. This robot is equipped with proximity sensors. These sensors will receive signals reflected from the nature of the surroundings. The signals act as indicators for navigation options such as turning points or dead ends.

1.2 Problem Statement

A mobile robot has the ability of moving about in random manners. But provided with a set of obstacles within proximity, a mobile robot will have a difficulty in either avoiding them or finding its way back. In other words, once a mobile robot detects an obstacle, it will lose its intended track in attempt to avoid the obstacle. Thus, incorporating suitable sensors to the mobile robot and programmed with a suitable navigation algorithm, a mobile robot could overcome the said problems.

Aside from that, most existing robot designs do not include the feature which allows it to memorize the paths taken in order to create a smarter and faster navigation robot. With this feature, a mobile robot does not have to waste time on finding the route every time it goes through a set of maze after its first round.

1.3 Objectives

The primary objective of the project is to build a robot with the ability to navigate its way from the starting point to the exit point of a physical maze which can be rearranged in several configurations. But to have that achieved, the robot has to first respond to the proximity sensors accordingly.

1.4 Scope of Study

The project will focus on building the maze robot to meet its basic requirement which is to traverse through a physical maze. The study will be on two main elements of the project which are the robot and the maze. The former which is the main part will touch on the integration of PIC and the mobile robot in order to produce the Maze Robot. On the other hand, the latter which is the maze itself will cover the construction of the maze which includes the arrangement of barriers and openings, and the material used. An additional feature to be considered is to have the robot memorize the routes taken on the first round and determine which would be the best route to go through on the next round.

CHAPTER 2
LITERATURE REVIEW & THEORY

2.1 Autonomous Vehicle Navigation

Autonomous vehicle navigation covers both indoor and outdoor aspects. The study of autonomous mobile robots could serve many possible applications in the military, civilian or space fields. For the civilian field, autonomous mobile robotics concerns the logistics, airport surveillance, agriculture, cleaning industry, or intelligent driving. And for the military market, autonomous mobile robotics is beneficial as they are reducing manpower and increasing armed forces [1].

The accuracy to which a mobile robot needs to navigate acts as the scale of measurement for device navigation requirements, which vary with each type of application. Basically, there are three types of navigation requirement scales [2]:

- ➢ Global Navigation
  - ▪ Ability to determine one's position in map-referenced terms and also to be able to move to a desired point.
- ➢ Local Navigation
  - ▪ Ability to determine one's position relative to objects in surrounding and interact with them.
- ➢ Personal Navigation
  - ▪ Aware of the positioning of own various parts in relation to each other as well as in handling objects.

On the whole there are two types of maze where one is model-based maze and the other is sensor information-based maze. The former has its global models foreseen unlike the latter where its models are completely unknown. The sensor information-based maze uses sensors to collect information such as the size of the maze, the condition of branches in the maze, the dead-end and junctions as well as the current location of the robot in the maze [3].

2.2 Finite State Machine

Event-driven programming is a programming means which comprised of two sections; event selection and event handling. Events refer to the possible actions that a user might perform during an application usage. While an event handler manages a particular event after an event type has been identified.

The behavior of an event-driven system can be divided into a number of chunks. Event responses within each chunk depend on only the current event type but no longer on the sequence of past events. Basically, event-driven programming is used to produce efficient and maintainable software with well understood behavior without creating 'spaghetti code' which refers to the conventional and complex if-else programming structure.

Finite state machines are used for specifying and implementing event-driven systems. States refer to the "chunks of behavior" in an event-driven system. Whilst any changes of behavior are called state transition. This concept is handy as it reduces the number of execution paths through the code, simplifies the conditions tested at each branching point, as well as simplifying the transitions between different modes of execution [4].

Practically, to implement finite state machine, the process starts with behavior analysis where conditions and corresponding responses are identified. A finite number of states are determined along with the input combination which has its effect on the state transition. From this behavior analysis, a state transition table can be developed. The state transition table is then used to construct the state chart diagram. A simple theoretical example adapted from [5] is as the following:

Finite state machine example:

Take for example a simple programming where a microcontroller keeps track of the state of the LED. In this simple circuit there is an LED driver and a debounce button which is connected to the microcontroller. If the LED is in 'ON' state and the button is pressed, then the LED will be turned off. Similarly, if the LED is in 'OFF' state and the button is again pressed, then the LED will turn on. Therefore, we have three states, Start, LED 'On', and LED 'Off'. The FSM diagram is as shown in Figure 1:

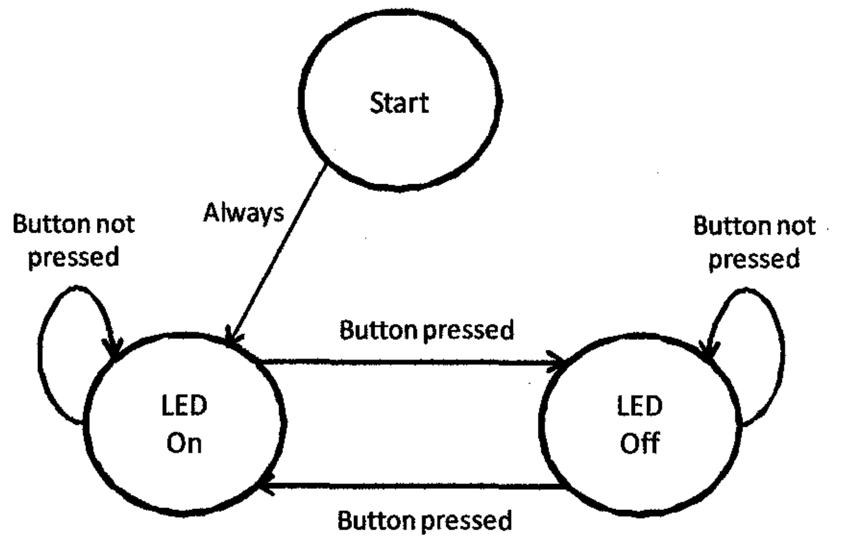


Figure 1: Example of FSM Diagram

This FSM diagram can then be translated into pseudocode before being programmed into real C program. A simple pseudocode of the FSM diagram for this example is as shown below:

```
Select state
    Case STARTstate
        Go to state 'LED On'

    Case LEDONstate
        If button is pressed, then
            Go to state 'LED Off'
        Else, remain in 'LED On' state

    Case LEDOFFstate
        If button is pressed, then
            Go to state 'LED On'
        Else, remain in 'LED Off' state

    Default case
        Go to 'Start' state
```

Thus, from the pseudocode above, a C program can be developed easily. This concept tests only the state variables compared to many variables in a conventional programming method. It eliminates a lot of conditional logic as it simplifies the switching between different states.

2.3 Microcontroller

Microcontroller is a low-cost single-chip computer which has its entire computer system within the confines of the integrated circuit. It has features and similarities to the standard personal computers where its primary feature is to store and run a program. It contains a central processing unit (CPU), random-access memory (RAM), read-only memory (ROM), electrically erasable programmable read-only memory (EEPROM), input/output (I/O) lines, serial and parallel ports, timers, and other built-in peripherals [6].

A microcontroller differs from microprocessor where the latter contains no RAM, no ROM, and no I/O ports on its chip. Even though this attribute of microprocessor provides versatility, it is a lot bulkier and more expensive. Thus this explains why microcontrollers are widely used as it is ideal for applications where cost and space are critical [7].

2.4 PIC Microcontroller

PIC basically refers to microcontrollers from Microchip Technology Corporation. PIC stands for Peripheral Interface Controller which has small amount of data RAM, a few hundred bytes of on-chip ROM, a timer, and a few pins for I/O ports, all on a single chip. PIC uses 8-bit processor which means that the CPU can control only 8 bits of data at a time. The PIC family ranges from many series from 10xxx up to 18xxx [7]. Concentrating on particularly PIC 16F6887 microcontroller, it has a program memory of 14KB, RAM memory of 368 bytes and EEPROM of 256 bytes. In addition, it has three kinds of timers and a set of Capture/Compare/PWM (CCP) module.

2.5 C Programming

C programming language is a widely used programming language for creating computer programs as it allows for easy implementation. It creates lists of instructions for a computer to follow in order to come up with a certain program. All C programming language is equipped with a standard library that contributes to efficiency and portability. Furthermore, its concise manner allows for powerful expressions to be coded as well as giving a direct access to many machine-level features that would otherwise be accessible only through the use of assembly language [8].

There are other languages other than C but its maximum control and efficiency has got the approval of most programmers around the world. C is a compiled language which means that once written, it must be run through a C compiler to turn the desired program into an executable that the computer can run. The C program is the human-readable form, while the executable that comes out of the compiler is the machine-readable and executable form [9]. Here are a few different C compilers designed for PIC programming:

i. Hi-Tech Compiler

   a. It can cater for different microcontroller series owned by Microchip.

ii. CCS Compiler

   a. Able to program many series of microcontrollers.

iii. C18 Compiler

   a. It is used to program the 18F series owned by Microchip.

2.6 PIC Programming

The microcontroller executes the program loaded in its flash memory which consists of binary code organized in 12-, 14-, or 16-bit wide words. These words are individually considered as executable instructions by the CPU when microcontroller is run [10]. All instructions that the microcontroller can recognize and execute are collectively known as the instruction set. The executable code is usually represented as a sequence of hexadecimal numbers called hex code which is a file format for conveying binary information. All programming languages supported by microcontroller generate a .HEX file which will be loaded to the microcontroller itself.

As for Microchip microcontrollers which are PICs, the general layout for any program is as follows:

a. Microcontroller header file which defines all the registers and peripherals.

b. Main configuration settings of PIC such as crystal frequency, watchdog status and others.

c. Main functions where the port initialization and input specification are set.

d. Rest of program which depends on user application.

e. Any use of peripherals or communication modules must be configured accordingly beforehand.

2.7 Integrated Development Environment (IDE)

IDE is a programming environment integrated into a software application that provides a graphic user interface (GUI) builder, a text or code editor, a compiler and/or interpreter and a debugger. It is the set of processes and programming tools used to create the program or software product which provides developers an orderly interface to and convenient view of the development process (or at least the processes of writing code, testing it, and packaging it for use).

2.8 Motors

There are many types of motor, but there are only two that will be put in discussion which is the basic DC motor and servo motor. Also included in this section is motor driver which acts as an intermediate between a motor and a chip that controls the motor.

2.8.1 DC Motor

DC motor is uses electricity and magnetic field to produce torque which turns the motor. Basically it requires two magnets of opposite polarity and an electric coil, which acts as an electromagnet. It uses the properties of magnets polarity to convert electricity into motion. The repellent and attractive electromagnetic forces of the magnets provide the torque that causes the DC motor to turn. The electromagnet switches the current flow as the motor turns, changing its polarity to keep the motor running. Its speed is controlled by pulse width modulation (PWM). It is a concept where the power level of the motor is controlled by strobing the power supply on and off [11].

2.8.2 Servo Motor

Servo motor is basically an assembly of a normal DC motor, a gear reduction unit, a position-sensing device such as potentiometer, and a control circuit. Servo acts as a receiver to a control signal that represents a desired output position of the servo shaft. It will then provide power to its DC motor until its shaft turns to that position. The potentiometer determines the rotational position of the shaft. Unlike DC motor which rotates freely round and round, a servo motor can only turn approximately 200 degrees back and forth [11]. Servo has its own internal drive electronics for running its built-in motors. A lightly loaded servo, therefore, does not consume much energy. It is extremely useful in robotics as the motors are small, with built-in control circuitry, and are extremely powerful for their size.

2.8.3 Motor Driver

Motor driver or also known as motor controller is used as an intermediate between a motor and a chip that controls the movement of the motor. Its function is to take a low-current control signal and turn it into a proportionally higher-current signal that is able to drive a motor [12]. It also provides the ability to control the motor movement from the direction of rotation to regulation of speed. Besides that, it also protects other ICs from electrical problems such as overloads and faults.

2.9 Proximity Sensors

Generally, sensor is a device that measures or detects a real-world condition, such as temperature, pressure, level, humidity, speed, motion, distance, light or the presence/absence of an object, and converts the condition into an analog or digital representation. The relevant type of sensor for this project is only the proximity sensor as it is able to detect the presence of nearby objects without any physical contact. There are many types of proximity sensors which serves various applications as discussed below [13].

2.9.1 Inductive Sensors

Inductive sensors utilize induced magnetic fields to respond to metallic objects. It consists of an oscillator circuit which acts as the sensing unit, and an output circuit which includes a switching device. An essential part of the oscillator circuit is the inductance coil in front of the sensing face which produces magnetic field.

2.9.2 Capacitive Sensors

Capacitive sensors respond to changes in the surrounding dielectric medium. It is often used in applications which cannot be solved with other sensing techniques due to its versatility of sensing almost any substance. The higher the dielectric constant, the more sensitive a capacitive sensor is to that target. However this sensor is less suitable for low density substances. Operation is based on an internal oscillator with two capacitive plate-electrodes, tuned to respond when a substance approaches the sensing face.

2.9.3 Ultrasonic Sensors

Ultrasonic sensors generate high frequency sound waves and make use of the wave reflection to detect parts or distances to the parts. It calculates the time interval between sending the signal and receiving the echo to determine the distance to an object. It is suitable for transparent targets.

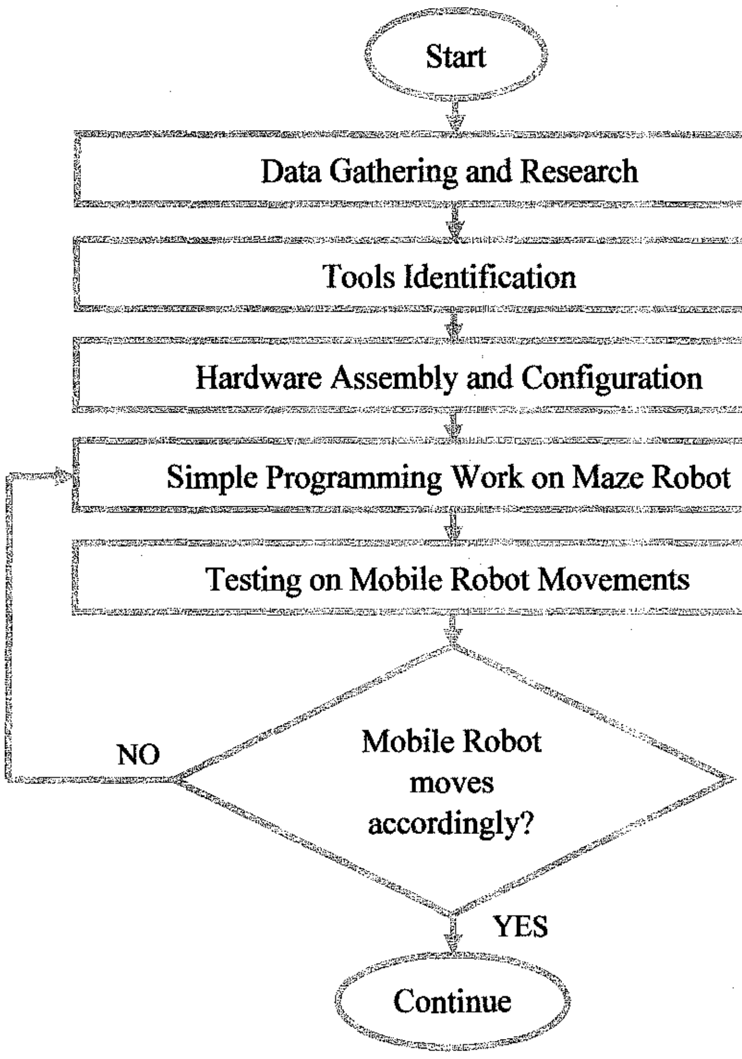
2.9.4 Photoelectric Sensors

Photoelectric sensors emit invisible infrared or visible red light and anticipate the light reflection to detect the presence of an object. The photoelectric sensor consists of a light emitting segment, a light receiving segment, an amplification circuit, an A/D converter, and a processing section. In the presence of an object, light reflected to the detection area will be converted into an electric signal. This electric signal is amplified and shaped in waveform to be converted into a digital value by the A/D converter. It compares the reflected light with a threshold value to determine the object presence.

CHAPTER 3
METHODOLOGY

3.1 Project Flowchart

This whole project was divided into two stages which was FYP 1 and FYP 2. The flow chart shown below is the summary of all activities done in FYP 1 and FYP 2.



Start
Data Gathering and Research
Tools Identification
Hardware Assembly and Configuration
Simple Programming Work on Maze Robot
Testing on Mobile Robot Movements
Mobile Robot moves accordingly?
NO
YES
Continue

Figure 2: Project Flowchart

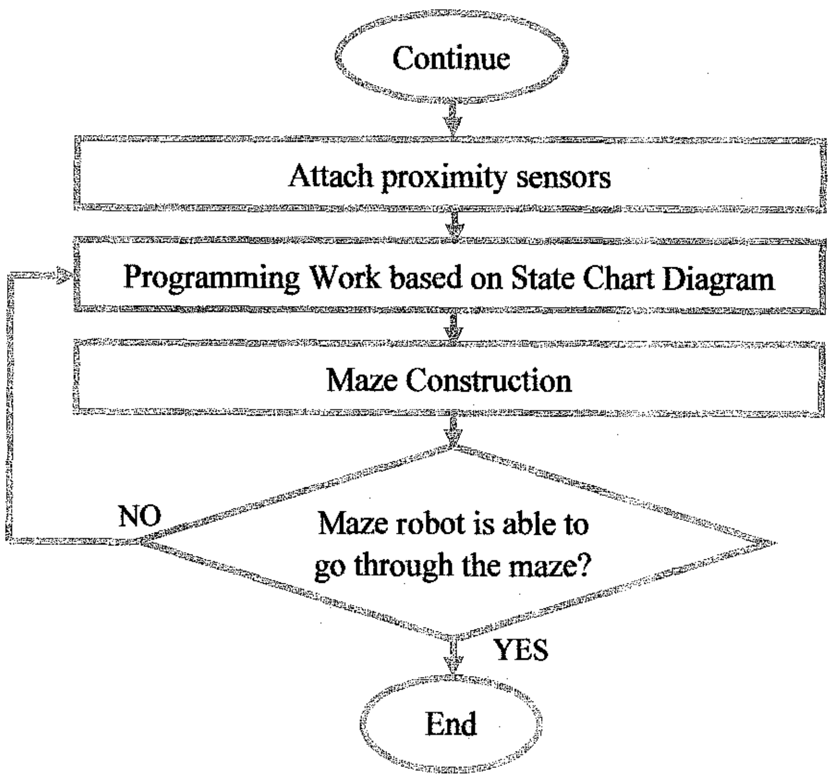

Continue
Attach proximity sensors
Programming Work based on State Chart Diagram
Maze Construction
Maze robot is able to go through the maze?
NO
YES
End

Figure 2: Project Flowchart

3.2 Project Activities

The project started with planning and research which comprised of the analysis stage where some researches were made to further understand and also to identify possible improvements that could be made on the project. Study was done on every aspect of the Maze Robot which ranges from the hardware design, software needs, and also the structure of the maze itself.

The initial part of prototype development was constructing the mobile robot controller. This stage is essential as it is more or less the foundation stage for completing this project. The main components of the mobile robot controller are the microcontroller, motor driver, and relevant sensors. In addition, components and tools identification took place to better understand the usage of each component. This stage started with an attempt to build a self-made circuit for the mobile robot controller. The datasheets and features of related components were studied to make sure they are compatible and are able to perform the functions required for this project.

However, due to time constraint, a ready-made mobile robot controller was used instead. This still involved a bit of studying and testing especially on the different microcontroller used, which was PIC 16F887 instead of previously used PIC 16F628A. The validity of the circuit was verified through a set of sample code prepared by the supplier. The mobile robot was ensured to be able to move forward, backward, turn left, turn right, and stop. Proximity sensors were also tested out and later on attached to the mobile robot controller.

This project adopts the Hardware and Software Co-design method where it emphasizes on simultaneous design along with simultaneous verification of both hardware and software so that it meets the requirement of a desired function. As a starter to the software development, a simple programming was constructed to incorporate the signals received by the sensors with the robot movement. The programming aimed for wall detection and robot change of direction.

After the initial programming went accordingly, the next step was to consider the speed of robot movement. This requires further study on pulse width modulation theory (PWM). The final stage of programming would be to come out with the optimal navigation algorithm for less time used to escape the maze.

The construction of the physical maze commenced after the navigation features are added to the robot. The material used for the physical maze wall was polystyrene. This was followed by robot refinement where all the components were affixed firmly and accordingly for troubleshooting ease.

The final stage was the implementation of the project which include troubleshooting and demonstration of project. These stages can be seen clearer in the Gantt charts planned for FYP 2 as per attached in Appendix A which also includes FYP 1 Gantt chart.

3.3 Tools

The followings are the required tools used to complete this project which comprises of three categories, hardware, software and additional materials.

3.3.1 Hardware

- PIC Microcontroller - 16F887 series
- MC40A Mini Mobile Robot Controller
- PIC Programmer
- IR Medium Range Sensors
- Motor driver – L293B
- DC Motors

3.3.2 Software

- Hi-Tech PICC Compiler – *C Compiler*
- MPLAB IDE – *Assembly and C Compiler*
- PIC Simulator IDE – *PIC Microcontroller Simulator*
- CCS PICC Compiler – *C Compiler*
- PICkit 2 – *PIC Programmer Software*

3.3.3 Additional Materials

- Polystyrene
- Construction tools

3.4 Mobile Robot Controller

A ready-made circuit for mobile robot controller was purchased in replacement of the circuit built in FYP 1. This change of plan was due to time constraint and insufficient energy supplied to the former circuit. The mobile robot controller used was MC40A mini mobile robot controller by Cytron Technologies Sdn. Bhd. The PIC microcontroller and motor driver used are PIC 16F887 and L293B respectively. For reference, the schematic circuit and manual for MC40A board are as per attached in Appendix B and Appendix C.

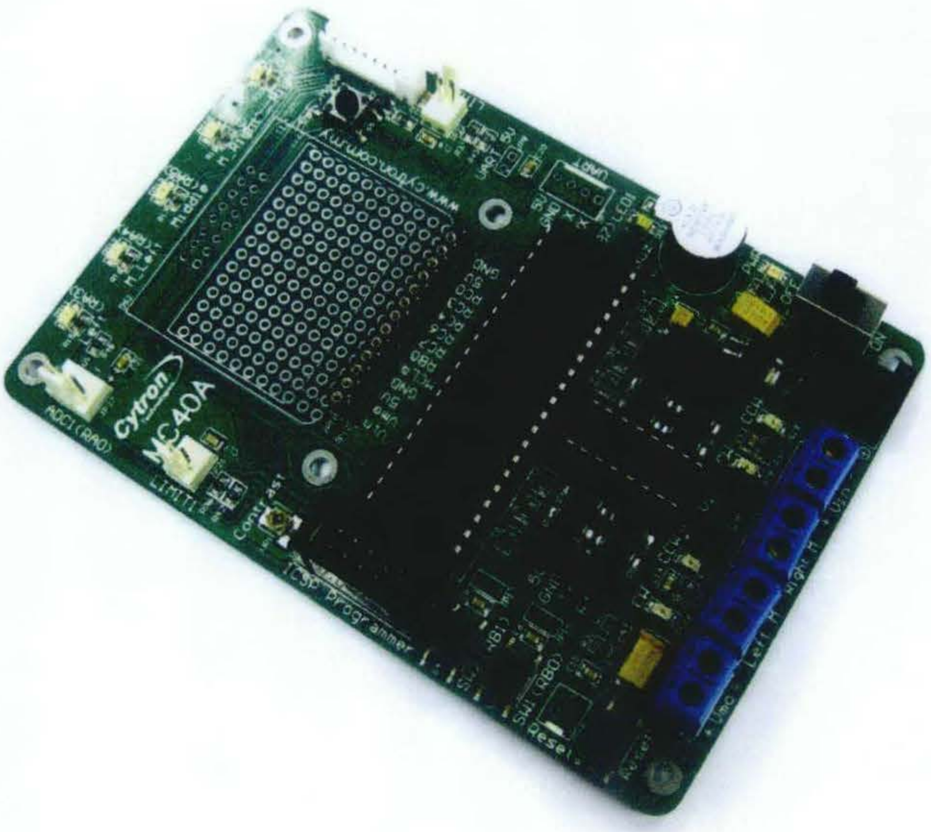


Figure 3: MC40A Mini Mobile Robot Controller
(Image courtesy of Cytron Technologies Sdn Bhd)

The initial step was testing out the MC40A board to implement only the necessary functions in order to ensure the mobile robot was able to move in different directions. This was done by using the MC40A board sample code provided by its supplier. Power supply used was 6V of four AA-size 1.5V batteries.

PIC 16F887 contained internal clock, therefore no crystal oscillator was needed to control the execution of instructions. This internal clock was triggered through the configuration settings in the PIC programming. The ports available for PIC 16F887 are port A, B, C, D, and E, unlike the previous microcontroller PIC 16F628A which had only two ports of port A and B. For this project, the main inputs which are the sensors were assigned to port C, while the outputs which are the motor driver inputs were assigned to port B. More details on PIC 16F887 could be retrieved from its datasheet as per attached in Appendix D.

The motor driver used in MC40A board was L293B. The motor enable pins are connected to the PIC which allows the rotation of both left and right wheels. The datasheet for L293B is as per attached in Appendix E.

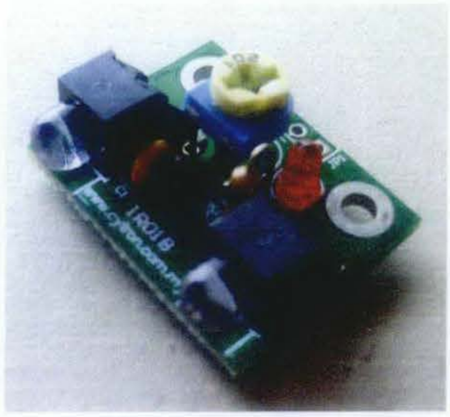


Figure 4: Medium Range Infrared Sensor

Three medium range infrared sensors were attached to the MC40A board with connection to the PIC 16F887 microcontroller. The datasheet for the sensors are as per attached in Appendix F. The three sensors were intended for three different positions of different directions on the mobile robot. All the sensors were assigned to detect obstacles that lie either in front, right or left of the mobile robot. The distance between sensor and obstacles that was able to be detected was only 2.5cm.

3.5 Initial Programming

The initial circuit was tested out using a simple source code to test out the effectiveness of the circuit connection. The programming aims to supply the motor with power via microcontroller. The motor was made sure to be able to move forward, backward, turn left, turn right, and stop.

Before the code was uploaded to the PIC, it was tested out using the PIC Simulator IDE. In the simulator, instead of having the motor driver as the output, LEDs were used to represent the movement of the motor. The LEDs were labeled from B0 to B7 where B0, B1, B2, and B3 in real-mode, are the pins connected to L293B which drive the left and right wheels. On the other hand, B4 and B5 are the L293B enable pins which were constantly logic 1 unless not used, and B6 and B7 are always left to logic 0 since they were not in use.

The source code for this test is as per attached in Appendix G. The followings are the results obtained from the PIC simulator after the implementation of the code.

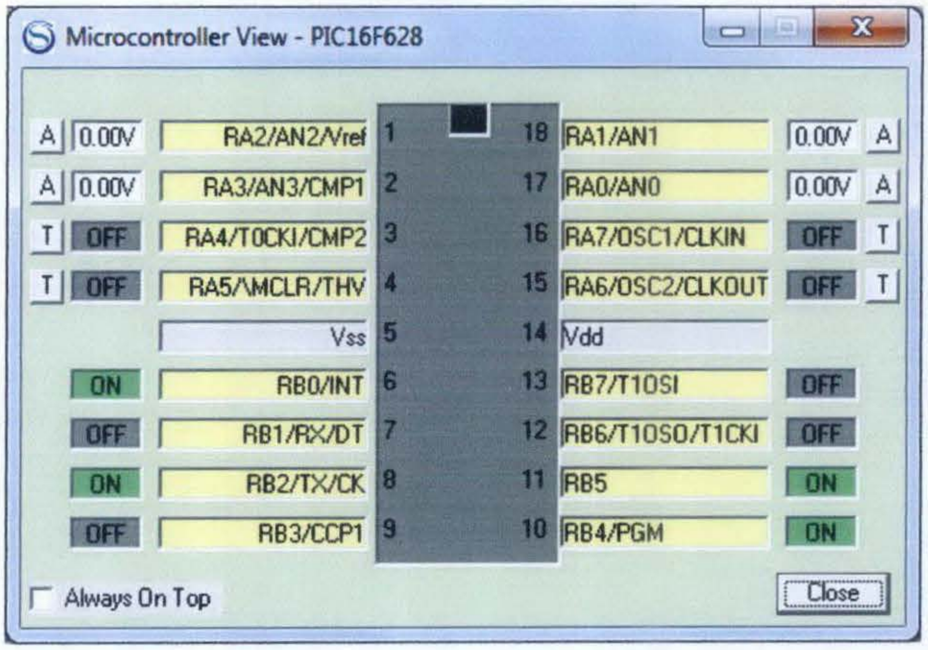


Figure 5: Microcontroller Layout during Code Execution

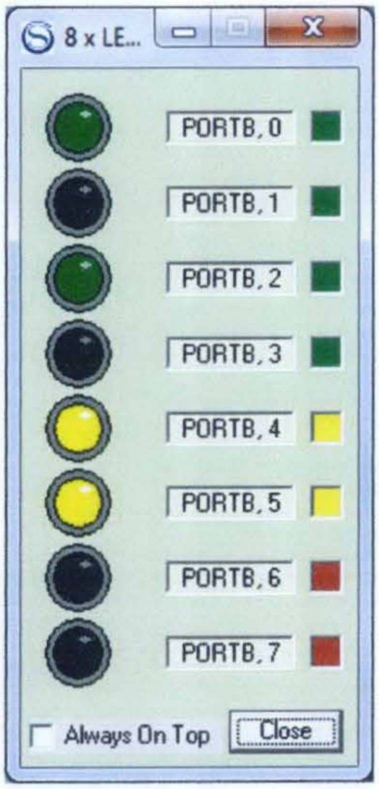


Figure 6: Output on LEDs for Moving Forward

As shown in Figure 4, this is the result from a portion of the program, when the mobile robot is meant to move forward. B4 and B5 are high to enable both left and right motors. This remains the same for all movements of mobile robot. B0 and B2 received logic 1 while B1 and B3 were grounded to indicate that in actual, both right and left wheels are moving forward in a straight direction.

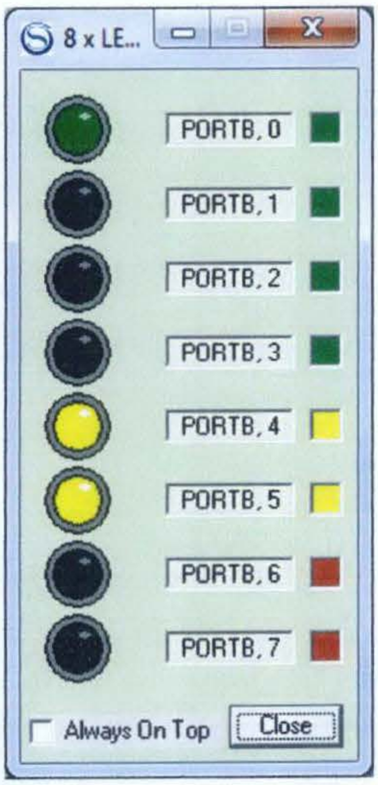


Figure 7: Output on LEDs for Turning Right

While in Figure 5, only LED for pin B0 lights up. This indicates that only the right motor is moving while the left motor which is controlled by pin B2 is left unmoved. Since in actual, only the right wheel moves, the mobile robot will make a right turn.

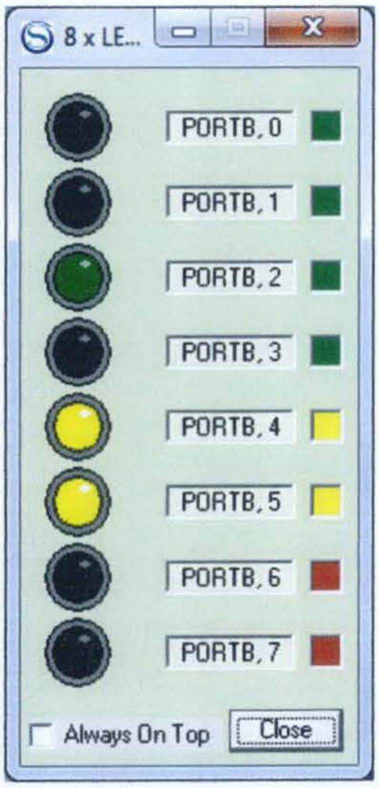


Figure 8: Output on LEDs for Turning Left

On the contrary, in Figure 6, only LED for pin B2 lights up. This indicates that only the left motor is moving while the right motor which is controlled by pin B0 lights off. Therefore, in actual the mobile robot will make a left turn as only the left wheel moves while the right wheel remains static.

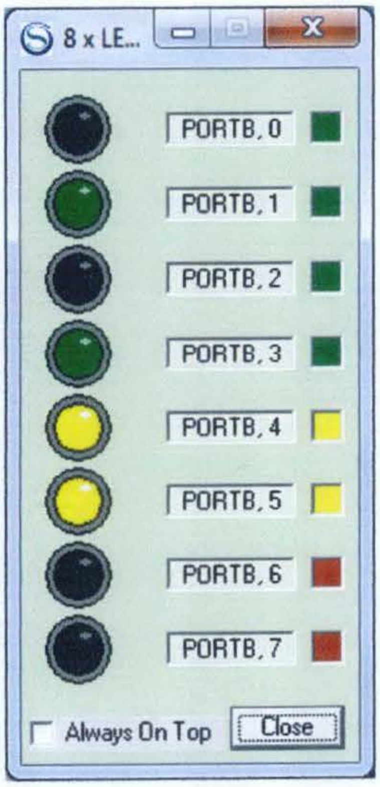


Figure 9: Output on LEDs for Moving Backward

Figure 7 is a reversal of Figure 4 where instead of pin B0 and B2 light up, only pin B1 and B3 light up. B0 and B2 are put to ground. This indicates that the mobile robot will move backward for this stage.

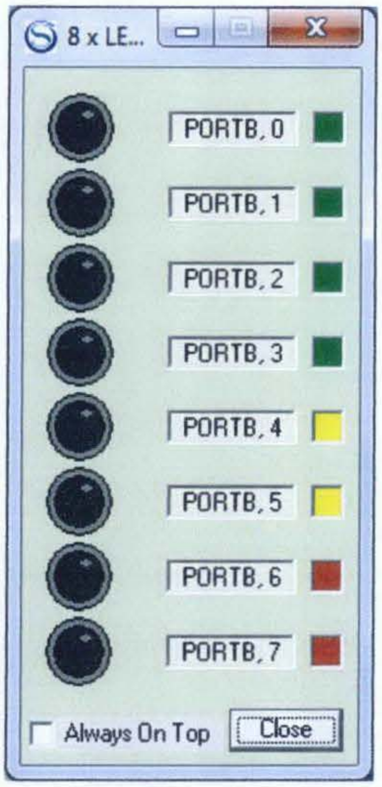


Figure 10: Output on LEDs for Stopping

Finally, Figure 8 shows no light is on since the enable pins are put to ground. This indicates that the motors are not running, thus the mobile robot will stop and remain static at this stage.

However, since the microcontroller used was replaced by PIC 16F887, the source code used for this experiment was still applicable but with a few minor alterations. The modified source code is as per attached in Appendix H.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 The Maze Robot

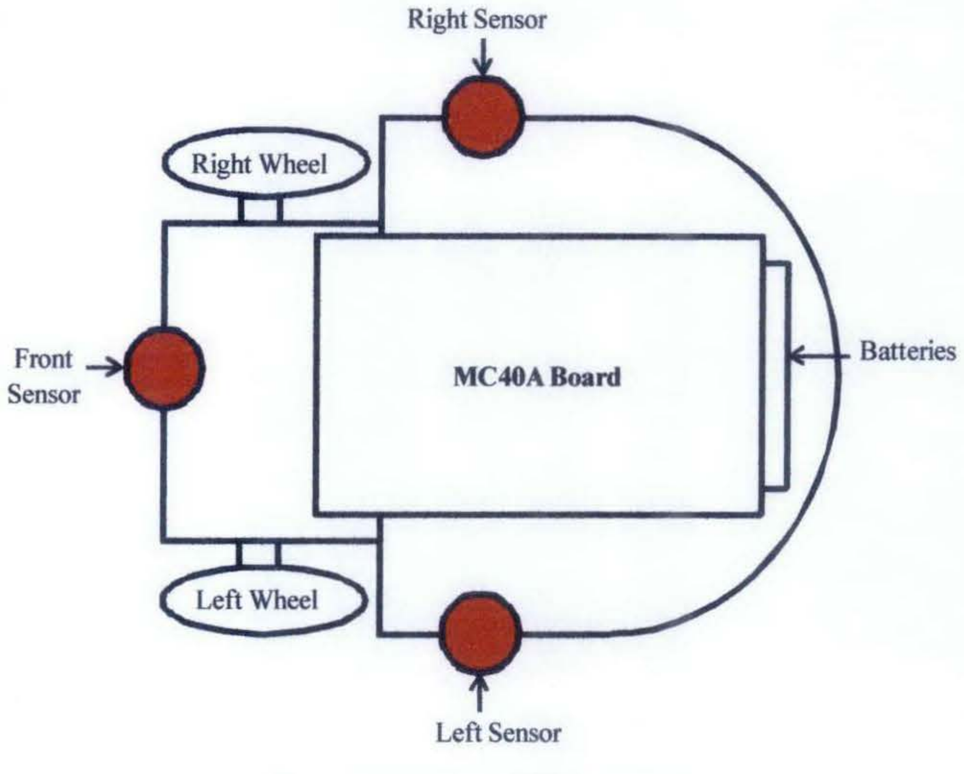


Figure 11: Maze Robot Design

The maze robot comprised of a plastic base which is attached to two DC motors to facilitate the left and right movement. The mobile robot controller, MC40A board is affixed on top of the robot base with the batteries fitted in between those two. Three IR medium range sensors are attached to the front, left and right of the robot base. These locations were chosen because the maze robot is designed to detect the presence of obstacles at the front, left, or right of the robot.

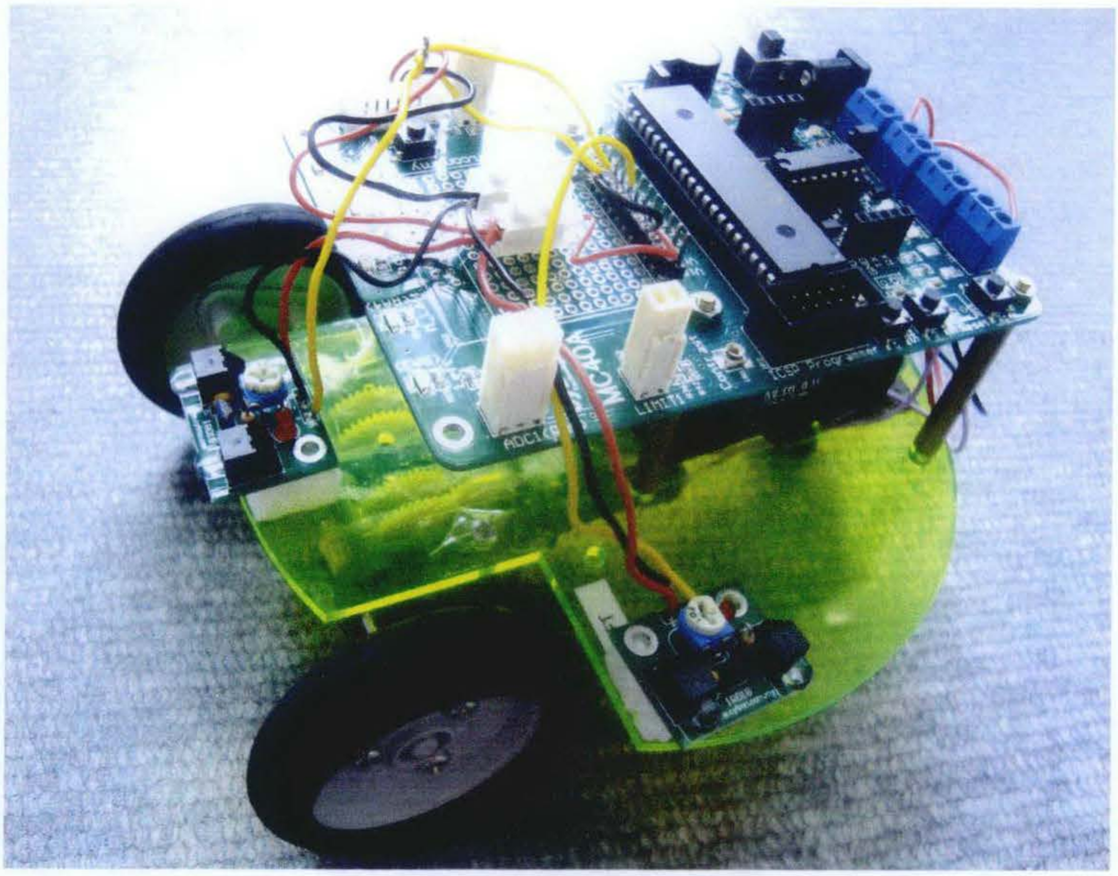


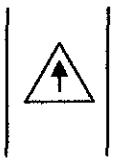
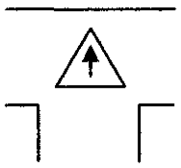
Figure 12: Actual Maze Robot

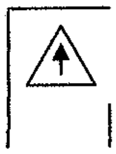
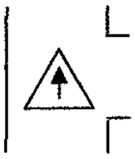
The MC40A board uses separate power supply from the DC motors where the board is powered up by 6V of batteries and the motors are powered up by 9V of batteries. On the contrary, the sensors are supplied with 5V from the PIC microcontroller. The signals from all three sensors are fed to three different input pins of the microcontroller. Both left and right DC motors are controlled by the motor driver, L293B.

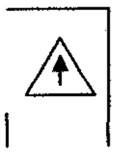
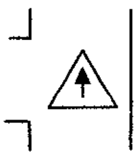
The paths to be taken through the maze are not known by the maze robot. Therefore, the maze robot finds its way out based on the information from the IR sensors. It solves the maze by following the right-hand wall from the entrance to the exit point. This solution was derived deliberately through event-driven programming method where the FSM diagram was constructed before the pseudocode can be developed.

4.2 Behavior Analysis

Following after the hardware assembly was to incorporate the signals from the IR sensors with motor movements. In order to do so, a state chart diagram was constructed since the event-driven programming method was used for this project. There are many conditions of the surrounding to be analyzed in order to determine the robot movements. Such conditions are the obstacles, or to be specific, the walls that would hinder the robot advancement. In order to obtain a smooth sailing navigation, a few cases on wall presence and robot response were made. Below is the breakdown of all the cases identified for this project. F, L, and R denote the front, left and right sensors which detect the front, left, and right wall respectively. 1 means obstacle is detected and 0 means no obstacle detected.

Case 1:  L = 1
F = 0
R = 1
Direction: Move Forward
	Case 5:  L = 0
	F = 1
	R = 0
	Direction: Turn Right

Case 2:  L = 1
F = 1
R = 0
Direction: Turn Right
	Case 6:  L = 1
	F = 0
	R = 0
	Direction: Turn Right

Case 3:  L = 0
F = 1
R = 1
Direction: Turn Left
	Case 7:  L = 0
	F = 0
	R = 1
	Direction: Move Forward

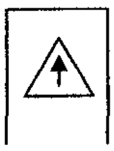
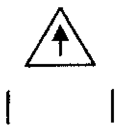
Case 4:  L = 1
F = 1
R = 1
Direction: Make a U-turn
	Case 8:  L = 0
	F = 0
	R = 0
	Direction: Idle/Stop

Table 1: Behavior Analysis

4.3 State Transition Table

A state transition table was built based on the case analysis mentioned in section 4.2. This table is the preparatory step in order to construct the state chart diagram. The next state depends on both the current state and its input combination which is the sensors.

Current State	Sensors			Next State
	Left	Front	Right	
Idle	0	0	1	Forward
	X	1	X	Idle
	0	0	X	Idle
	1	0	0	Idle
Forward	0	0	0	Idle
	0	1	1	Left
	1	X	0	Right
	0	1	0	Right
	X	0	1	Forward
	1	1	1	U-turn
Right	X	X	X	Forward
Left	X	X	X	Forward
U-turn	X	X	X	Forward

Table 2: State Transition Table

4.4 State Chart Diagram

The state chart diagram below was constructed based on the state transition table mentioned in section 4.3. This diagram is basically a more visual version of the state transition table. The initial step for this project is the 'Idle' state which will proceed to next state 'Move Forward', depending on the input combination that goes into the system at that time. The 3 bits of input were derived from three proximity sensor signals which are the left, front and right sensors respectively.

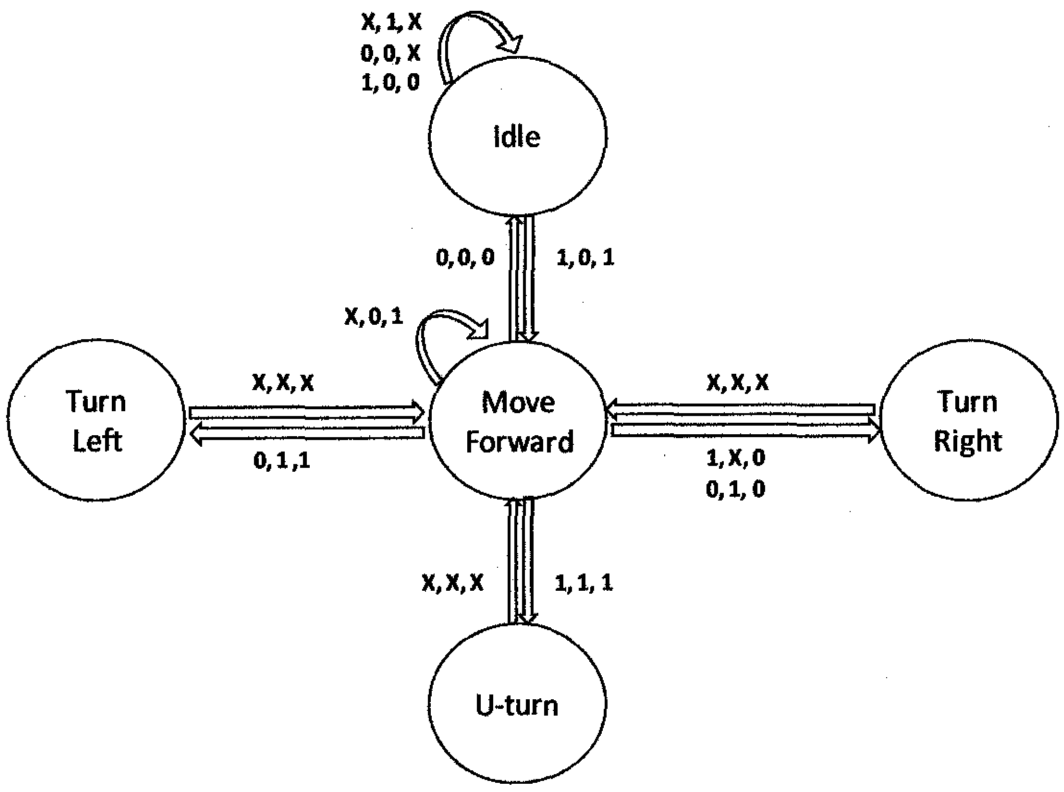


Figure 13: State Chart Diagram

The event-driven programming method was easily implemented with the aid of this state chart diagram. Aside from that, pulse width modulation was utilized in order to control the motor speed. The complete source code for the maze robot is as per Appendix I.

4.5 Algorithm

The navigation algorithm for this project aims to have the robot to follow the right-hand wall until it finds the exit of the maze. Therefore, the maze robot prioritize on turning to the right, given no obstacles are present on the right side. This can be seen in case 2, 5, and 6 of Table 1. Its least priority of direction is the left turn where the robot would only turn to the left if it was a left corner when obstacles are present on both the front and right side. This can be seen in case 3 of Table 1. In cases where the front and either sides of the robot have no obstacles as shown in case 6 and 7 of Table 1, the robot will choose to turn right for the former and move forward for the latter. This is because the robot was made to follow the right wall until it finds the exit.

The turning of the maze robot is a special case since the robot actual movement while turning would have it turn 90°, resulting in the robot to advance forward only a little but not enough to have its whole body completely entering the headed junction. This could affect the reading of the sensors if the robot does not enter the new junction completely. It will give a false sensor reading. Therefore, every turning movement was followed by moving forward one length. One length refers to the robot body length. This fix can be seen clearer on the images of the robot turning to the right as shown below.

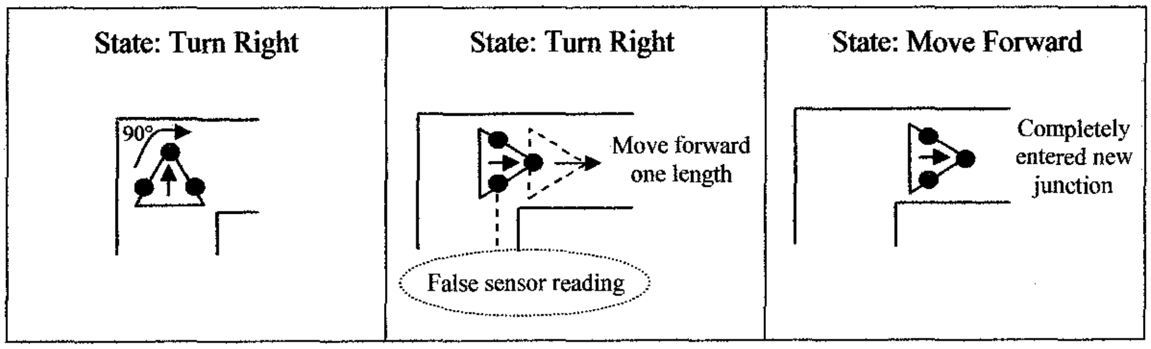


Figure 14: Special Case for Turning of Robot

The method used in applying this algorithm was the event-driven programming where finite state machine was utilized to show the flow of the maze robot movement from one state to another. The switch-case method was used to denote each state and its statement.

4.5.1 Pseudocode

The next step into programming the algorithm was to develop the pseudocode. Shown below is the pseudocode to the main program of the navigation algorithm which uses the switch-case method:

```
Select state
      Case IDLE
            Robot stops
            If L=1, F=0, R=1, then
                  State = FORWARD
            Else, state = IDLE

      Case FORWARD
            Robot moves forward
            If L=0, F=0, R=0, then
                  Move forward a bit for 2 seconds
                  If L=0, F=0, R=0, then
                        State = IDLE
            Else if L=0, F=0, R=1, then
                  State = FORWARD
            Else if L=0, F=1, R=0, then
                  State = RIGHT
            Else if L=0, F=1, R=1, then
                  State = LEFT
            Else if L=1, F=0, R=0, then
                  State = RIGHT
            Else if L=1, F=0, R=1, then
                  State = FORWARD
            Else if L=1, F=1, R=0, then
                  State = RIGHT
            Else if L=1, F=1, R=1, then
                  State = UTURN

      Case RIGHT
            Robot turns right 90 degrees
            State = FORWARD

      Case LEFT
            Robot turns left 90 degrees
            State = FORWARD

      Case UTURN
            Robot makes a u-turn
            State = FORWARD
```

4.5.2 *Source Code*

With the aid of the pseudocode from section 4.5.1 earlier, the source code can be constructed. The robot movements of moving forward, turning right, turning left, making a u-turn as well as staying idle were grouped into respective sub-functions. The source code was compiled using the CCS PICC Compiler. Shown below is an extract of the navigation portion from the main program. Since the sensors are output low, therefore 1 does not denote an obstacle presence as explained in theories explained earlier. Rather, for this source code, 0 denotes an obstacle presence, while 1 denotes that a path is cleared.

```
// Maze navigation program based on FSM
        switch(state)
        {
            case IDLE:
            {
                  stop();
                  if (L == 0 && F == 1 && R == 0)
                  {
                     state =FORWARD;
                     break;
                  }
                  else
                  { break;}
            }

            case FORWARD:
            {
                 gofwd();

                  // No walls: Forward/Stop
                  if (L == 1 && F == 1 && R == 1)
                  {
                       while(1)
                       {
                       delay_ms(200);

                        if(L == 1 && F == 1 && R == 1)
                        {
                           k++;

                        }
            else
            {
                   state=FORWARD;
                   break;
            }

               if (k>5)
               {
                   Buzz;
                   state = FIN;
                   break;
               }
            }
      }
        // Wall on Right only: Forward
      else if (L == 1 && F == 1 && R == 0)
      {
            state=FORWARD;
      }

      // Wall at Front only: Right
      else if (L == 1 && F == 0 && R == 1)
      {
            state=RIGHT;
      }

      // Walls on Right and Front: Left
      else if (L == 1 && F == 0 && R == 0)
      {
            state=LEFT;
      }

      // Wall on Left only: Right
      else if (L == 0 && F == 1 && R == 1)
      {
            state=RIGHT;
       }

      // Walls on Left and Right: Forward
      else if (L == 0 && F == 1 && R == 0)
      {
            state=FORWARD;
      }

      // Walls on Left and Front: Right
      else if (L == 0 && F == 0 && R == 1)
      {
            state=RIGHT;
      }
```

```
                // Walls on everyside: U-turn
                else if (L == 0 && F == 0 && R == 0)
                {
                        Buzz;       // Buzz while u-turning
                        state=UTURN;
                }

                else{}
                break;
        } // break from FORWARD
        case LEFT:
        {
                reverse();
                turnleft();
                state =FORWARD;
                break;
        }

        case RIGHT:
        {
                reverse();
                turnright();
                state =FORWARD;
                break;
        }

        case UTURN:
        {
                reverse();
                makeuturn();
                Buzz0;
                state =FORWARD;
                break;
        }

        case FIN:
        {
                delay_ms(500);
                Buzz0;
                stop();
                state=END;
                break;
        }
} // end of select-case
```

4.6 Maze Construction

The final step was the testing of the maze robot effectiveness by the use of a physical maze. Shown below is the maze design used to test out the effectiveness of the maze robot. The red dotted lines indicate the route that the maze robot should take based on the case analysis.

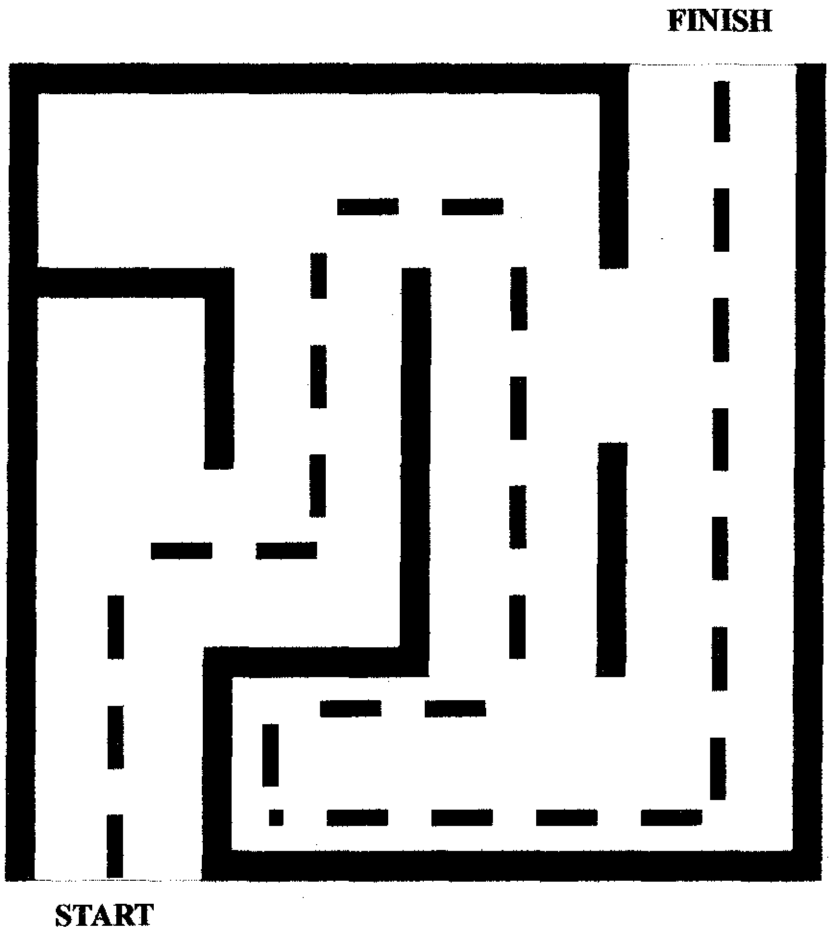


Figure 15: Maze Design

The design of the maze was based on the behavior analysis discussed in section 4.2 where all cases were taken into consideration. The maze shown in Figure 15 tests all possible cases to ensure the robot would react accordingly when certain situation occurs. This was essential in order to verify that all cases are valid.

The physical maze was built using polystyrene. Care was taken to make sure the widths of the alleys were not more than 19cm. This is because the width of the robot is approximately 15cm and a single sensor can detect an obstacle from a distance of approximately 2.5cm. Since there are left and right sensors to detect obstacles from both directions, therefore the width of the alley should not exceed 19cm. It is crucial to have the robot within the detection proximity to allow it to detect obstacles presence more efficiently based on the behavior analysis discussed in section 4.2.

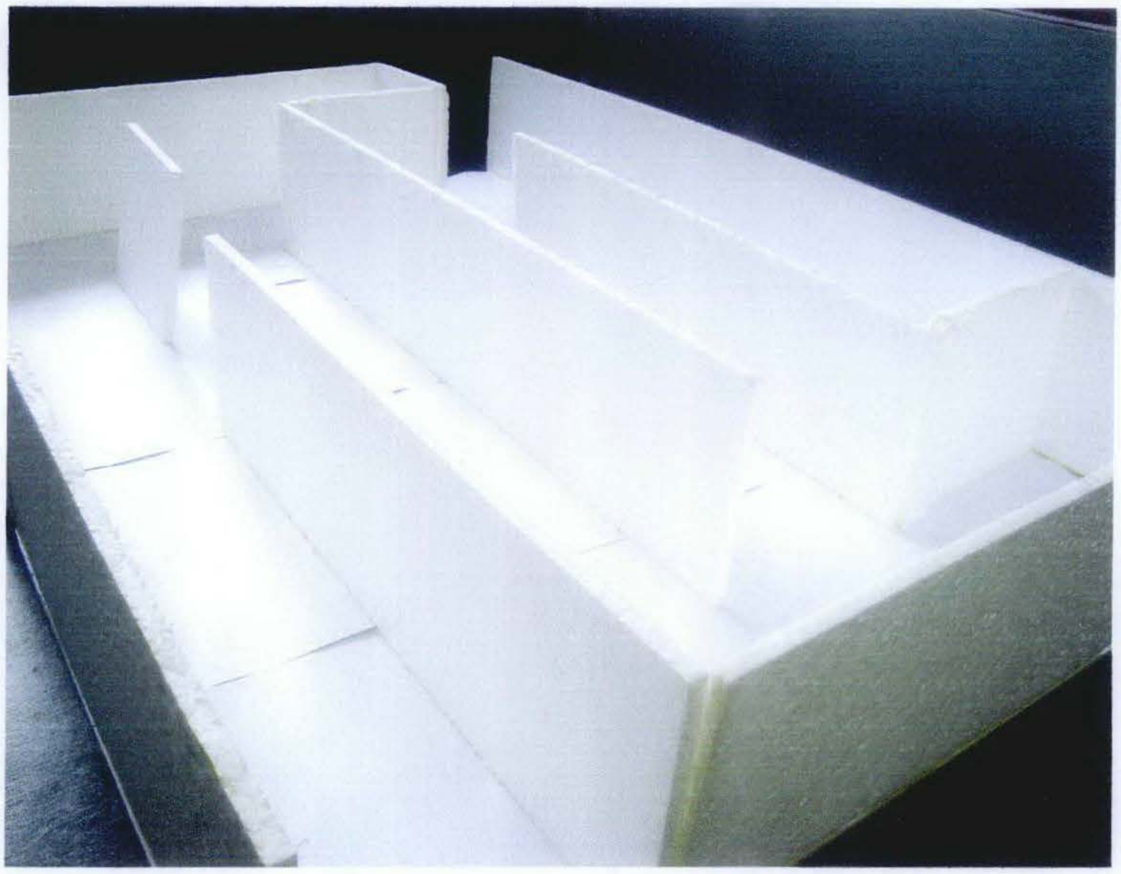


Figure 16: Actual Physical Maze

Figure 16 shows the actual physical maze which was constructed from polystyrene. Figure 17 shows the actual maze robot going through the physical maze with the LEDs of the IR sensors visibly switched on whenever walls were detected. The pictures of Figure 17 represent all the eight cases from Table 1.

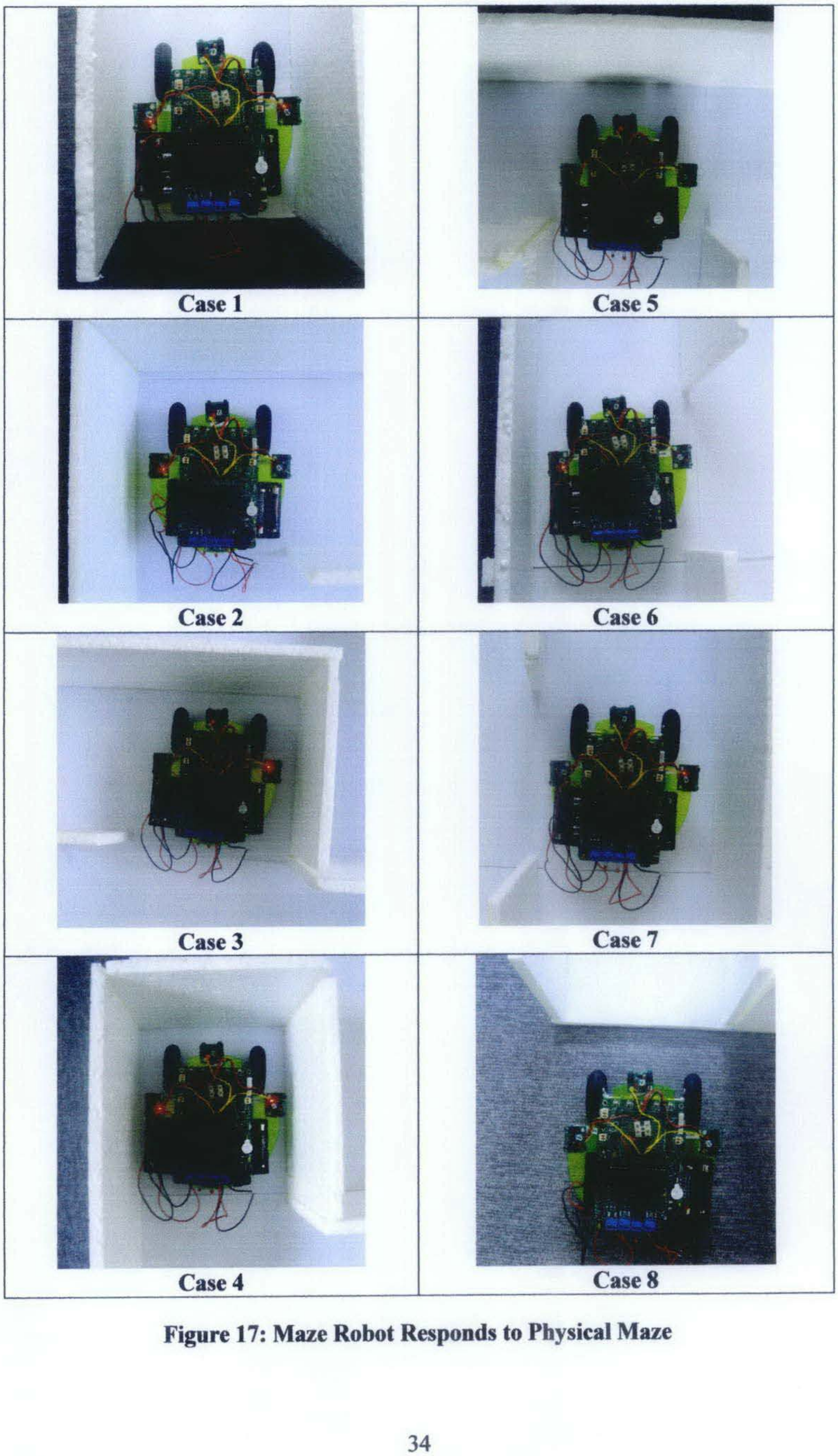


Figure 17: Maze Robot Responds to Physical Maze

CHAPTER 5
CONCLUSION & RECOMMENDATIONS

5.1 Conclusion

Maze Robot aims to contribute to the study of artificial intelligence especially in autonomous navigation. This project demonstrates the robotic function where the robot is able to find its way out through a set of physical maze with the help of relevant sensors. The programming method used in this project which was event-driven programming technique proved to be a better manageable means of coding which reduces the 'spaghetti code' occurrence. However, some features and improvements as mentioned in the following recommendation section would add more value to this study of robotics.

5.2 Recommendations

The maze robot built is only able to perform the basic requirements needed to find its way out a physical maze. However some improvement and enhanced features would increase the efficiency and functionality of the maze robot. The sensors used for this project are able to detect only a fair amount of distance which slows down the navigation process as care must be taken to avoid the robot from crashing into the walls. Therefore, it is advisable to use sensors which can detect obstacles from a larger distance. Other than that, memory should be added to the maze robot for it to be able to calculate the best route it can take for its next travel through the maze. Furthermore, the maze design can also be improved to make it more complicated to test out more complex routes such as wide path or island finishing point.

REFERENCES

[1] F. Carre, L. Gallo, B. Mazar, F. Megel, and B. Serra (1998), "Monai: An Autonomous Navigation System for Mobile Robots".

[2] J. Dixon & O. Henlich (1997), Mobile Robot Navigation, Retrieved August 2011, from http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/jmd/

[3] Jiang Hualong, Wang Honggi, and Tian Yonghong, "Design and Realization of a Maze Robot".

[4] Miro Samek (2009). Practical UML Statecharts in C/C++, Second Edition. USA: Newnes.

[5] Finite State Machine based LCD Controller, Retrieved July 2011, from http://www.engscope.com/pic-example-codes/finite-state-machine-based-lcd-controller/

[6] John Iovine (2004). PIC Microcontroller Project Book (pp. 1-11). USA: McGraw-Hill.

[7] Muhammad Ali Mazidi, Rolin D. Mckinlay, & Danny Causey (2008). PIC Microcontroller and Embedded Systems (pp. 23-35). New Jersey: Pearson Prentice Hall.

[8] Stas Bekman (2010), *C-C++ Frequently Asked Questions*, Retrieved February 2011, from http://stason.org/TULARC/webmaster/lang/c-cpp-faq/28-Why-are-C-and-C-so-popular-and-widely-used.html

[9] Marshall Brain (2000), *How C Programming Works*, Retrieved January 2011, from http://computer.howstuffworks.com/c1.htm

[10] Milan Verle (2010), *Programming Microcontrollers*, Retrieved February 2011, from http://www.mikroe.com/eng/chapters/view/75/pic-basic-book-chapter-2-programming-microcontrollers/

[11] The Handy Board, *What is the difference between a DC motor and servo motor?*, Retrieved January 2011, from http://handyboard.com/hb/faq/hardware-faqs/dc-vs-servo/

[12] Eric Seale (2003), Motor Drivers, Retrieved March 2011, from http://www.solarbotics.net/library/pieces/assy_driver.html

[13] Engineer's Handbook (2006), *Mechanical Components - Proximity Sensors*, Retrieved January 2011, from http://www.engineershandbook.com/Components/proximitysensors.htm

APPENDICES

APPENDIX A
GANTT CHARTS

Timeline for Final Year Project 1

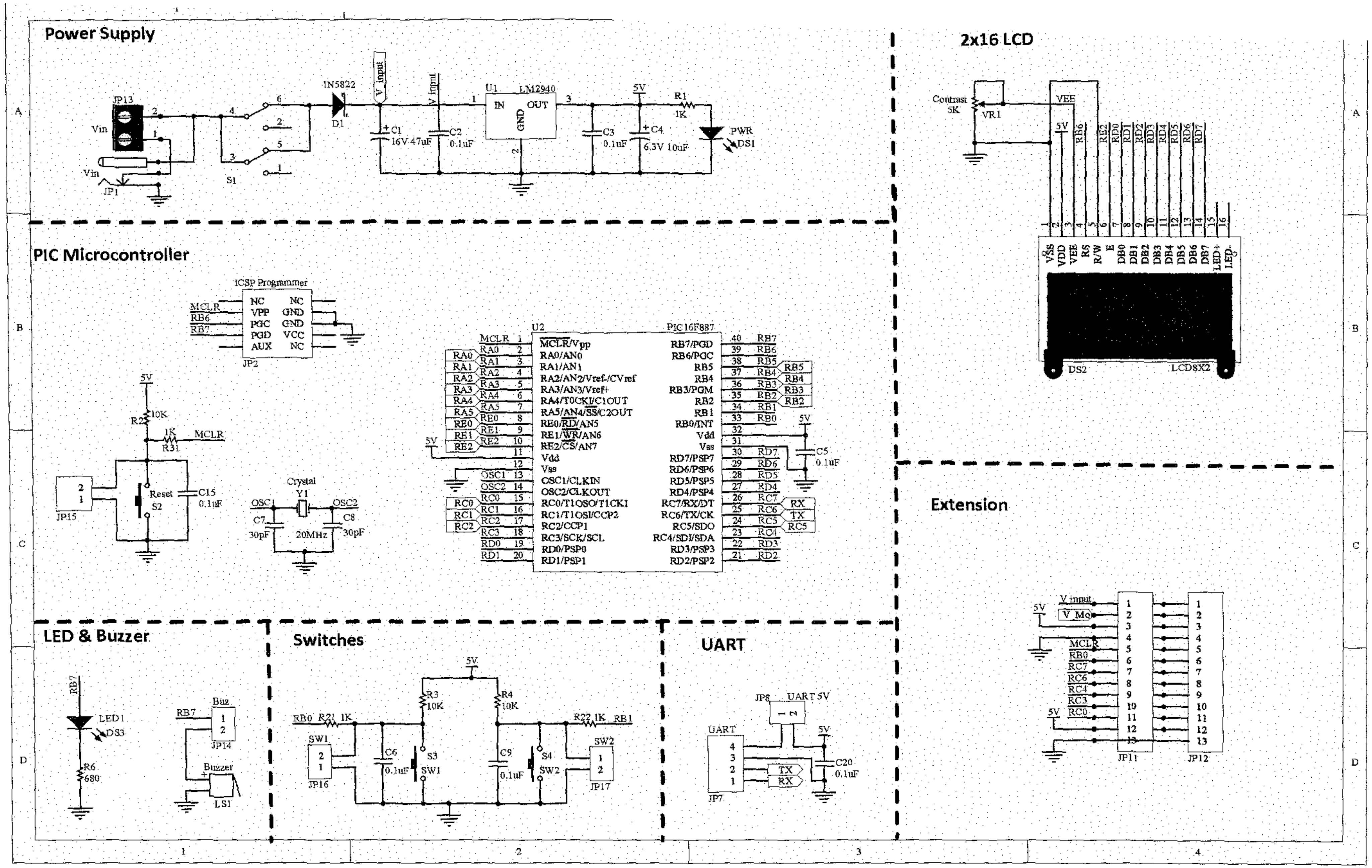
No.	Detail/ Week	1	2	3	4	5	6	7		8	9	10	11	12	13	14
1	Selection of Project Topic	■	■													
2	Preliminary Research Work		■	■	■											
3	Components and Tools Identification				■	■	■									
4	Submission of Extended Proposal						■	■								
5	Hardware Assembly							■		■						
6	Proposal Defense									■	■					
7	Continue on Hardware Assembly										■	■	■			
8	Study on PIC Programming												■	■	■	■
9	Submission of Interim Report													■	■	■

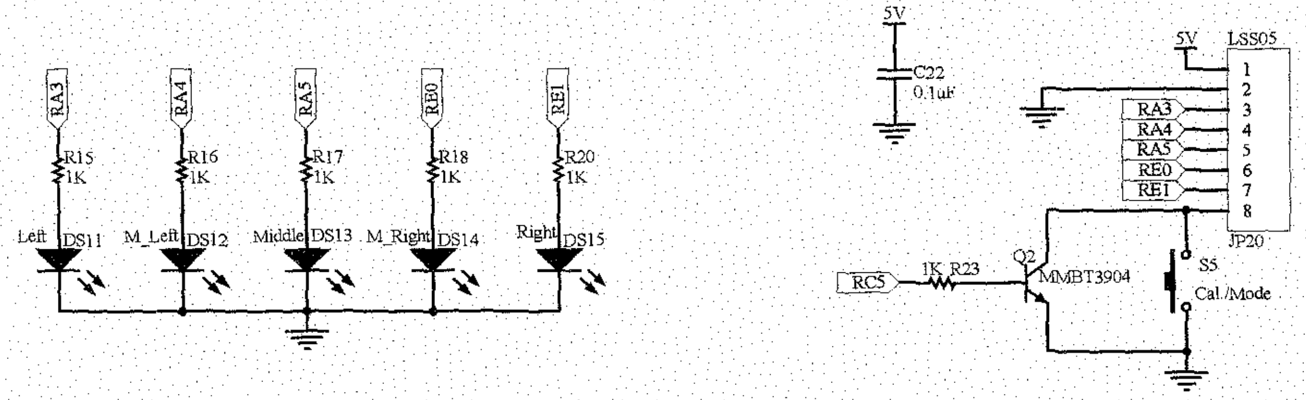
Timeline for Final Year Project 2

No.	Detail/ Week	1	2	3	4	5	6	7		8	9	10	11	12	13	14
1	PIC Programming	■	■	■	■	■	■	■		■						
2	Maze Construction									■	■	■				
3	Submission of Progress Report									■						
4	Prototype Troubleshooting										■	■	■			
5	Poster Exhibition												■			
6	Submission of Dissertation (Soft-bound)													■		
7	Oral Presentation														■	
8	Submission of Project Dissertation (Hard-bound)															■

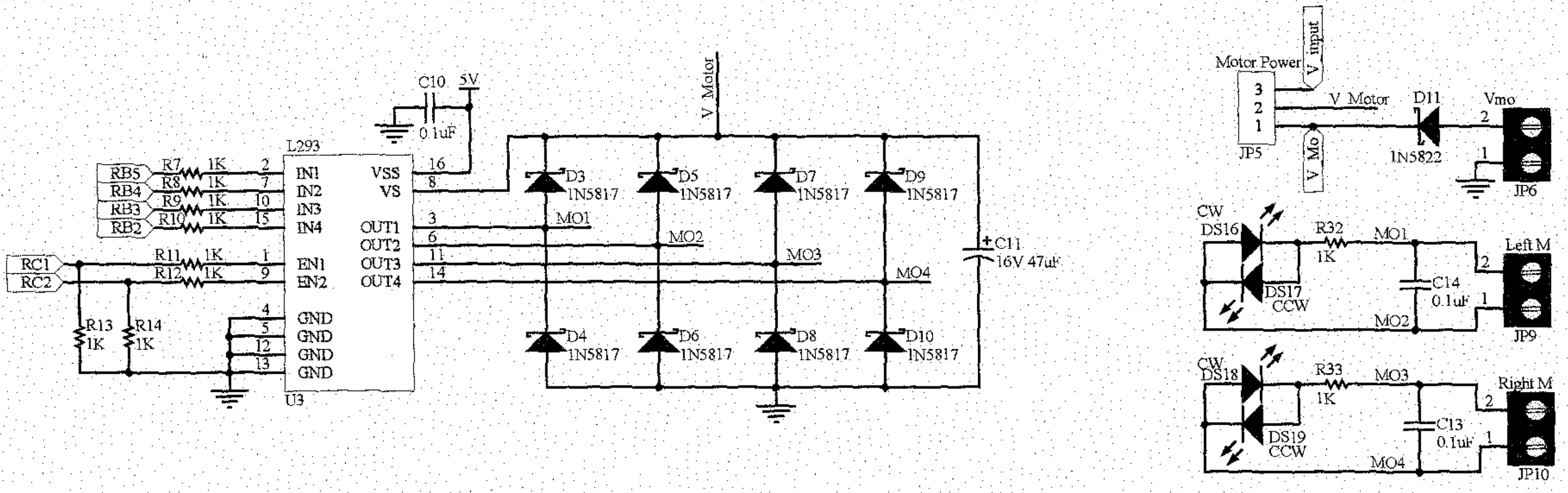
APPENDIX B

MC40A BOARD SCHEMATIC CIRCUIT


Power Supply
JP13
Vin
JP1
S1
1N5822
D1
V_input
C1 16V 47uF
C2 0.1uF
U1 LM2940
IN
OUT
GND
5V
C3 0.1uF
C4 6.3V 10uF
R1 1K
PWR
DS1
2x16 LCD
Contrast 5K
VR1
VEE
VSS VDD VEE RS R/W E DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 LED+ LED-
DS2
LCD8X2
PIC Microcontroller
ICSP Programmer
NC VPP PGC PGD AUX
NC GND GND VCC NC
MCLR
RB6
RB7
JP2
U2
PIC16F887
MCLR/Vpp
RA0/AN0
RA1/AN1
RA2/AN2/Vref-/CVref
RA3/AN3/Vref+
RA4/T0CKI/C1OUT
RA5/AN4/SS/C2OUT
RE0/RD/AN5
RE1/WR/AN6
RE2/CS/AN7
Vdd
Vss
OSC1/CLKIN
OSC2/CLKOUT
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RD0/PSP0
RD1/PSP1
RB7/PGD
RB6/PGC
RB5
RB4
RB3/PGM
RB2
RB1
RB0/INT
Vdd
Vss
RD7/PSP7
RD6/PSP6
RD5/PSP5
RD4/PSP4
RC7/RX/DT
RC6/TX/CK
RC5/SDO
RC4/SDI/SDA
RD3/PSP3
RD2/PSP2
C5 0.1uF
R2 10K
R31 1K
Reset
S2
C15 0.1uF
JP15
Crystal
Y1
20MHz
C7 30pF
C8 30pF
OSC1
OSC2
Extension
V_input
V_Mo
JP11
JP12
LED & Buzzer
LED1
DS3
R6 680
Buz
JP14
Buzzer
LS1
Switches
R3 10K
R4 10K
R21 1K
R22 1K
RB0
RB1
SW1
SW2
C6 0.1uF
C9 0.1uF
S3
S4
JP16
JP17
UART
JP8
UART 5V
JP7
TX
RX
C20 0.1uF



Motor Driver



APPENDIX C

MC40A BOARD USER MANUAL

1. INTRODUCTION AND OVERVIEW

MC40A is designed as mini mobile robot controller. With the rich features, it helps beginner to get start in building mobile robot, yet reserving the feature to expand the capabilities of the controller. Come with sample source code for PIC16F, user may start in no time with powering up the controller.

- Suitable for 40-pin 8-bit PIC Microcontroller
- Come ready with PIC16F887
- Sample source code to test the board
- Sample source code for Line following, reading ADC, displaying message on LCD, controlling DC brush motor, communication through UART
- Input power: 7V to 12V
- Support 2 DC brush motor as actuator for mobile robot tires
- Support 10-bit PWM of speed control, both channel of motor
- Motor power: 5V- 12V, selectable from Vmotor, or share from input power
- Support 2x8 parallel LCD (optional), operate in 4-bit interface.
- Support Cytron SK series including SKPS, SKXBee, SKKCA
- A Buzzer and LED as programmable output
- 2 programmable push button as digital input
- Ready with 2 connectors for limit switch
- Ready with ADC input for Infrared distance sensor, Ultrasonic distance sensor or other type of analog sensor.
- Ready with LSS05 (Auto-Calibrating Line Sensor) connector
- Ready with ICSP connector for UIC00A/B for loading program to PIC.
- Ready with connector for UC00A, USB to UART converter
- Free IO pins are extended out for further development.
- **Dimension: 12cm x 8 cm**

MC40A is suitable for u developing autonomous robot as it help to:

- Save the time in designing and developing interface between electronics component.
- Save the time purchasing and choosing the suitable components.
- Eliminate the frustration on soldering, testing and downloading program.
- Reduce unstable condition of self develop controller board.

This document explains the method to use MC40A.

3. PRODUCT SPECIFICATION AND LIMITATIONS

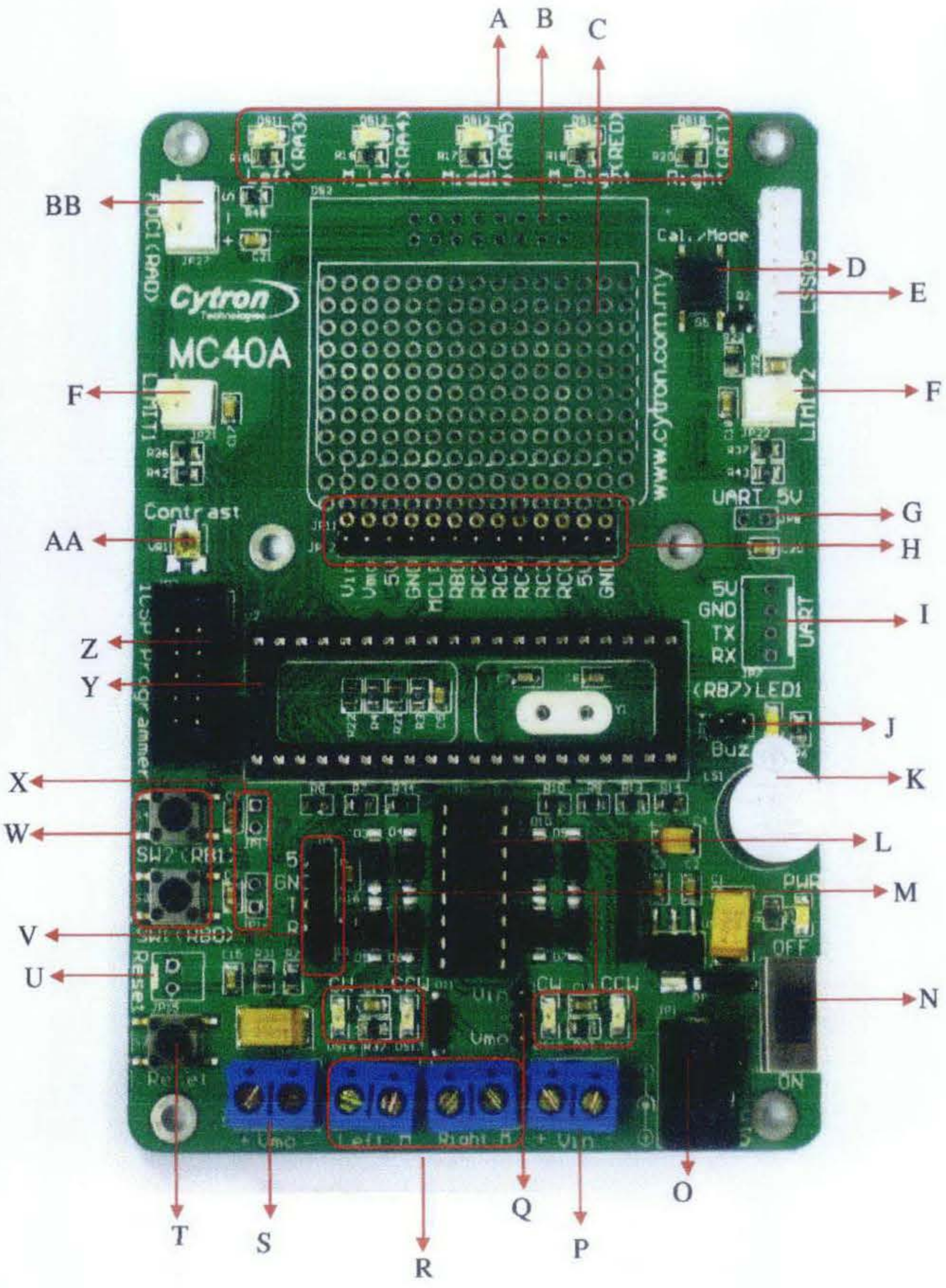
MC40A is designed to offer controller for mobile robot. The specification of the product should be referred.

Absolute Maximum Rating

Symbol	Parameter	Min	Max	Unit
V_{IN}	Operating voltage	6	12	V
V_{motor}	Motor supply voltage	5	12	V
I_{IN}	Input Current for 5V (circuit board)		500	mA
I_{motor}	Output current for Motor, per channel		800	mA

Note: DONOT supply more than 12V to Vin or Vmotor as it will burn the capacitor further cause explosion. DONOT use un-regulated power adapter as the output voltage is not same as stated.

4. BOARD LAYOUT



Label	Function	Label	Function
A	Status indicator LED	O	DC adaptor socket
B	Port reserved for 2x8 character LCD	P	Vin terminal block
C	Expansion/prototyping area	Q	Motor power selection
D	LSS05 calibration button	R	Motor terminal block, 2 channels
E	LSS05 connector	S	Vmotor terminal block
F	Limit switch connector	T	Reset button
G	UART VCC	U	Optional connector for reset buttons
H	Header and turn pin of I/O and power	V	Cytron's SK header socket
I	UART connector	W	Programmable push buttons
J	Buzzer activation jumper	X	Optional connector for buttons
K	Buzzer	Y	40 pin PIC MCU (16F887)
L	L293	Z	ICSP Programming socket
M	Motor direction indicator LED	AA	Contrast for LCD
N	Main power switch	BB	ADC connector

A – Status indicator LED for line following. The LED will turn on when detect bright line in the bright mode and detect dark line in the dark mode.

B – Reserved for 2x8 LCD. This is optional, 2x8 LCD is not included in packing list.

C – Reserved for expansion/prototyping area. User may use this area to do prototyping with other devices.

D – Calibration button used to enter mode for LSS05. Press once to enter the calibration mode. Press twice to set the line sensor bar into dark line following mode and press 3 times to set the line sensor bar into bright line mode. This button only applies if LSS05 is used.

E – LSS05 connector to connect MC40A with LSS05.

F – Connectors for adding limit switches. Optional usage.

G – Optional jumper to connect 5V of MC40A to external UART device.

H – Header and turn pin which expanded free I/O of PIC MCU and also power. Users are free to connect the used pin for further development.

I – Reserved for UART communication. Pin to pin compatible with UC00A.

J – Pin selection to activate buzzer. Connect this header pin with mini jumper to activate buzzer.

K – Buzzer.

L – L293 motor driver.

M – Status indicator LED for motor direction, CW or CCW.

N – Power switch to switch ON or OFF the MC40A.

O – DC power adaptor socket for user to plug in DC adaptor. The input voltage should be ranged from 7 to 12V. Typical is 12V.

P – Terminal block for Vin power supply. Besides use DC adaptor, user may use this terminal block to supply power to MC40A. Only 1 connector should be connected to power.

Note: DONOT supply more than 12V to Vin or Vmotor as it will burn the capacitor further cause explosion. DONOT use un-regulated power adapter as the output voltage is not same as stated.

Q – Power selection for motor. Use mini jumper to choose either V_{in} or V_{mo} uses to supply power to L293 motor driver and further to drive motor. If V_{in} is select, V_{in} will supply power to DC motor. If V_{mo} is select, DC motor will have its own power from the V_{mo} terminal.

R – Terminal block to connect left and right DC motor.

S – Terminal block to supply alternative power to DC motor. If user chooses V_{mo} at JP5, user needs to connect power to this terminal block.

Note: DONOT supply more than 12V to Vin or Vmotor as it will burn the capacitor further cause explosion. DONOT use un-regulated power adapter as the output voltage is not same as stated.

T – Reset button for MC40A.

U – Reset connector for user to extend reset button to other place.

V – Pin header for Cytron Starter Kit. User may use SKPS, SKXBEE to control MC40A.

W – 2 Programmable push buttons.

X – The header pin is providing for user to expand SW1 and SW2 to other place. To avoid confusion, the header pin of programmable push button is different with the connector of reset button.

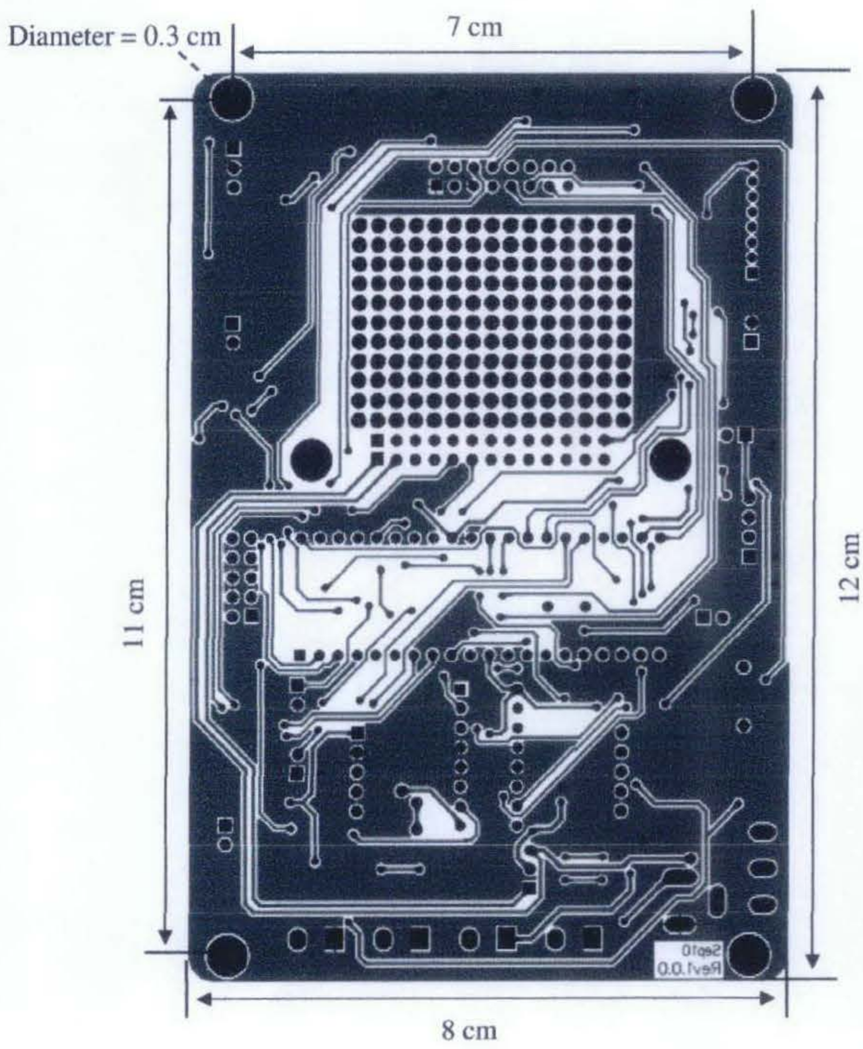
Y – 40 pin PIC Microcontroller.

Z – 2x5 box header for UIC00A/B, USB ICSP PIC Programmer.

AA – 5K of trimmer to set LCD contrast.

BB – This connector is provided for ADC. User may add analog sensor to MC40A using via this connector.

4.1 Dimension Drawing



5. INSTALLATION (HARDWARE)

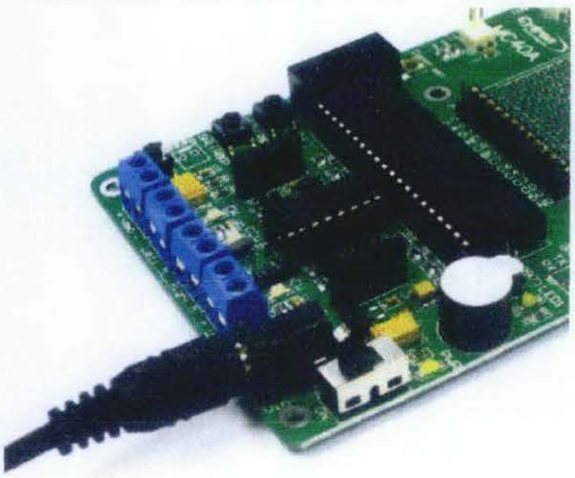
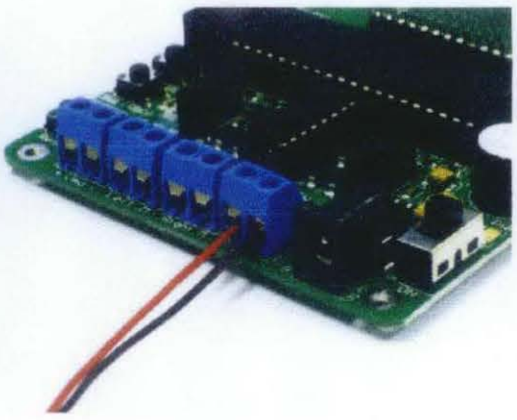
Though MC40A come with several features to help user get started, it will be good if we can show a few basic steps for beginner to get it "running". For full schematic of MC40A, please download it from MC40A product page.

5.1 Powering Up MC40A

There are 2 terminals for user to supply power to MC40A, this source will supply power to on board microcontroller, LCD (if installed) LSS05 (if installed), analog sensor, and other components. Power for motor can be connected to this source if user chosen to do so.

The 1st terminal for power input is DC adapter input. User may use the standard AC to DC adapter to supply the power into MC40A through this terminal. The voltage should be between DC 7V – 12V (maximum), typical value is DC12V.

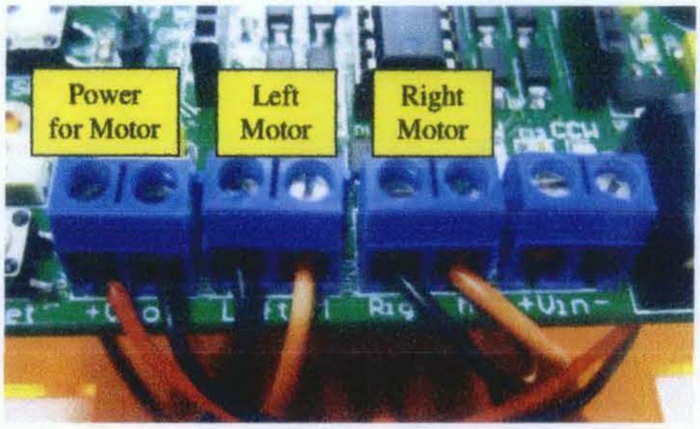
Alternatively, user may use wire terminal from battery to supply MC40A. Figure below shows the sample connection from battery to Blue color terminal. Do ensure the polarity of supply is correctly installed. The (+) sign terminal should be connected to positive (+) terminal of battery and vice versa for the (-) terminal. There should be **ONLY ONE** power source connected to MC40A, is either through DC adapter input socket or Blue terminal block. **DO NOT** connects both.

Turn on the main switch and PWR LED should light ON. User may start loading program and connect other devices such as sensor, limit switches, SKPS or brush motor for further development.

5.5 Brush Motor

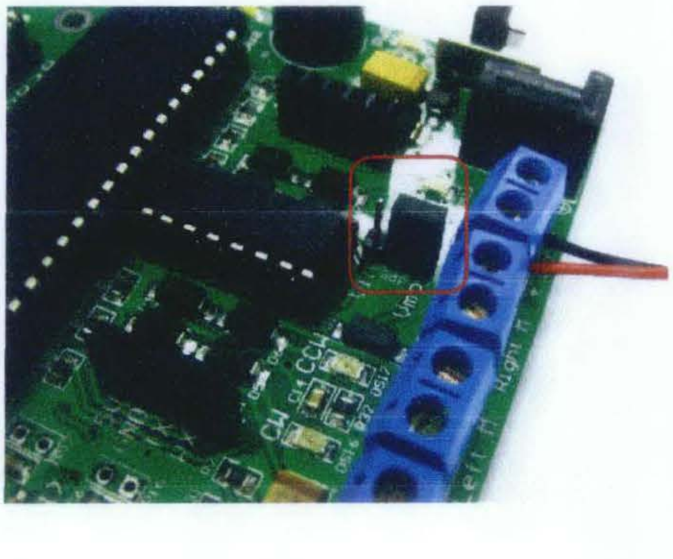
MC40A offers 2 channels to drive 2 brush motor bi-directionally with speed control. The power for motor can either be sharing the main input power source or separately from V_{mo}. The on board motor driver accept motor power source from 5V to 12V. User may supply this voltage into MC40A through the V_{mo} terminal. Though MC40A come with reverse polarity protection, it's always a good practice to connect the power accordingly to the marker on the terminal.



Example connection of motor supply and motors.

Note: DONOT supply more than 12V to Vin or Vmotor as it will burn the capacitor further cause explosion. DONOT use un-regulated power adapter as the output voltage is not same as stated.

On JP5, choose V_{mo} using mini jumper if user want to use power from the V_{mo} terminal for DC motor.



APPENDIX D

PIC 16F887 DATASHEET

PIC16F882/883/884/886/887

28/40/44-Pin Flash-Based, 8-Bit CMOS Microcontrollers with nanoWatt Technology

High-Performance RISC CPU:

- Only 35 Instructions to Learn:
  - All single-cycle instructions except branches
- Operating Speed:
  - DC – 20 MHz oscillator/clock input
  - DC – 200 ns instruction cycle
- Interrupt Capability
- 8-Level Deep Hardware Stack
- Direct, Indirect and Relative Addressing modes

Special Microcontroller Features:

- *Precision Internal Oscillator:*
  - Factory calibrated to ±1%
  - Software selectable frequency range of 8 MHz to 31 kHz
  - Software tunable
  - Two-Speed Start-up mode
  - Crystal fail detect for critical applications
  - Clock mode switching during operation for power savings
- Power-Saving Sleep mode
- Wide Operating Voltage Range (2.0V-5.5V)
- Industrial and Extended Temperature Range
- Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Reset (BOR) with Software Control Option
- Enhanced Low-Current Watchdog Timer (WDT) with On-Chip Oscillator (software selectable nominal 268 seconds with full prescaler) with software enable
- Multiplexed Master Clear with Pull-up/Input Pin
- Programmable Code Protection
- High Endurance Flash/EEPROM Cell:
  - 100,000 write Flash endurance
  - 1,000,000 write EEPROM endurance
  - Flash/Data EEPROM retention: > 40 years
- Program Memory Read/Write during run time
- In-Circuit Debugger (on board)

Low-Power Features:

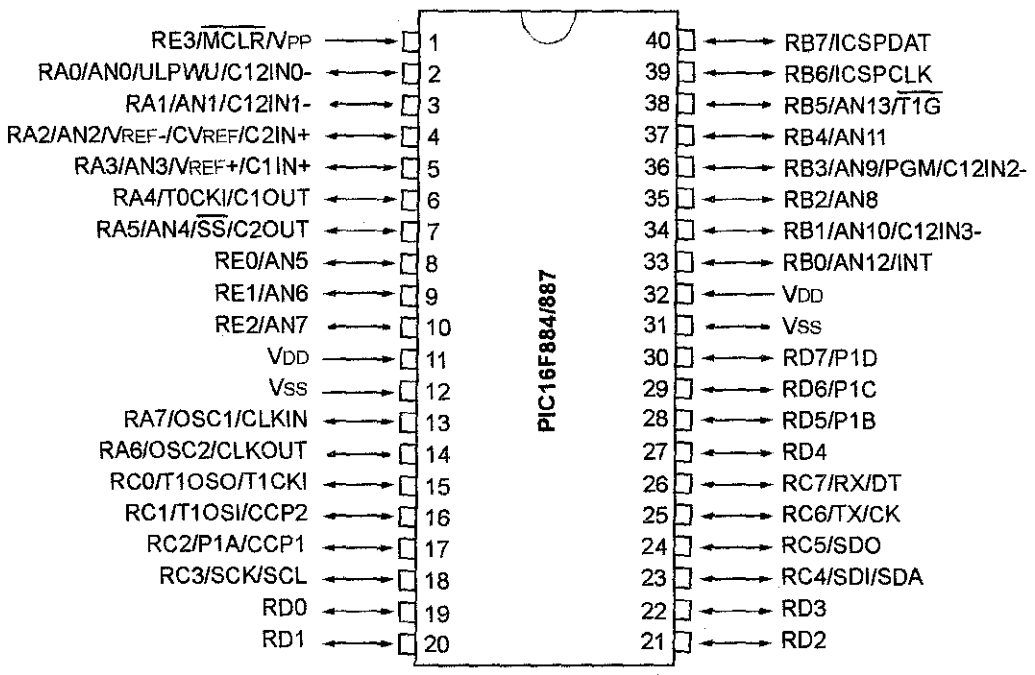
- Standby Current:
  - 50 nA @ 2.0V, typical
- Operating Current:
  - 11 μA @ 32 kHz, 2.0V, typical
  - 220 μA @ 4 MHz, 2.0V, typical
- Watchdog Timer Current:
  - 1 μA @ 2.0V, typical

Peripheral Features:

- 24/35 I/O Pins with Individual Direction Control:
  - High current source/sink for direct LED drive
  - Interrupt-on-Change pin
  - Individually programmable weak pull-ups
  - Ultra Low-Power Wake-up (ULPWU)
- Analog Comparator Module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (CVREF) module (% of VDD)
  - Fixed voltage reference (0.6V)
  - Comparator inputs and outputs externally accessible
  - SR Latch mode
  - External Timer1 Gate (count enable)
- A/D Converter:
  - 10-bit resolution and 11/14 channels
- Timer0: 8-bit Timer/Counter with 8-bit Programmable Prescaler
- Enhanced Timer1:
  - 16-bit timer/counter with prescaler
  - External Gate Input mode
  - Dedicated low-power 32 kHz oscillator
- Timer2: 8-bit Timer/Counter with 8-bit Period Register, Prescaler and Postscaler
- Enhanced Capture, Compare, PWM+ Module:
  - 16-bit Capture, max. resolution 12.5 ns
  - Compare, max. resolution 200 ns
  - 10-bit PWM with 1, 2 or 4 output channels, programmable "dead time", max. frequency 20 kHz
  - PWM output steering control
- Capture, Compare, PWM Module:
  - 16-bit Capture, max. resolution 12.5 ns
  - 16-bit Compare, max. resolution 200 ns
  - 10-bit PWM, max. frequency 20 kHz
- Enhanced USART Module:
  - Supports RS-485, RS-232, and LIN 2.0
  - Auto-Baud Detect
  - Auto-Wake-Up on Start bit
- In-Circuit Serial Programming™ (ICSP™) via Two Pins
- Master Synchronous Serial Port (MSSP) Module supporting 3-wire SPI (all 4 modes) and I^{2}C™ Master and Slave Modes with I^{2}C Address Mask

in Diagrams – PIC16F884/887, 40-Pin PDIP

40-pin PDIP



APPENDIX E

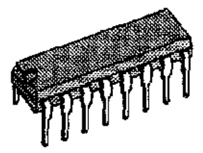
L293B MOTOR DRIVER DATASHEET

SGS-THOMSON
MICROELECTRONICS

L293B
L293E

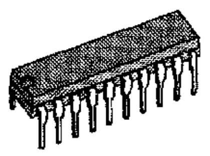
PUSH-PULL FOUR CHANNEL DRIVERS

- OUTPUT CURRENT 1A PER CHANNEL
- PEAK OUTPUT CURRENT 2A PER CHANNEL (non repetitive)
- INHIBIT FACILITY
- HIGH NOISE IMMUNITY
- SEPARATE LOGIC SUPPLY
- OVERTEMPERATURE PROTECTION



DIP16

ORDERING NUMBER : L293B



POWERDIP (16 + 2+ 2)

ORDERING NUMBER : L293E

DESCRIPTION

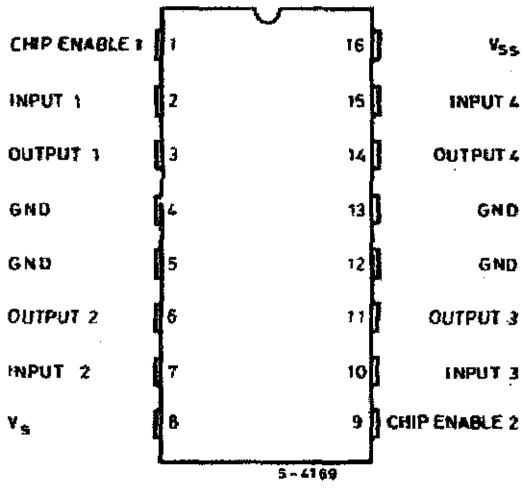
The L293B and L293E are quad push-pull drivers capable of delivering output currents to 1A per channel. Each channel is controlled by a TTL-compatible logic input and each pair of drivers (a full bridge) is equipped with an inhibit input which turns off all four transistors. A separate supply input is provided for the logic so that it may be run off a lower voltage to reduce dissipation.

Additionally, the L293E has external connection of sensing resistors, for switchmode control.

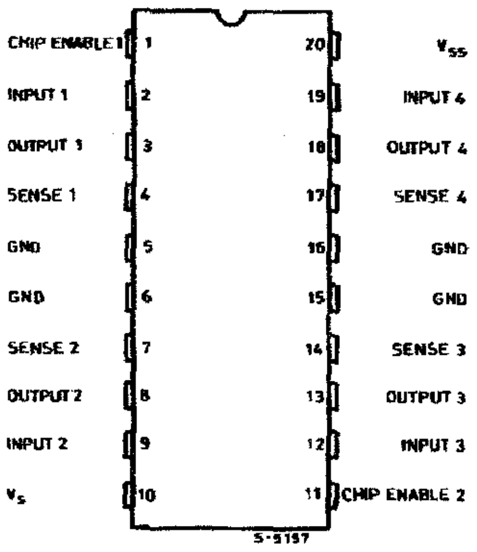
The L293B and L293E are package in 16 and 20-pin plastic DIPs respectively ; both use the four center pins to conduct heat to the printed circuit board.

PIN CONNECTIONS

DIP16 - L293B



POWERDIP (16+2+2) - L293E



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_s	Supply Voltage	36	V
V_{ss}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{inh}	Inhibit Voltage	7	V
I_{out}	Peak Output Current (non repetitive t = 5ms)	2	A
P_{tot}	Total Power Dissipation at $T_{ground-pins}$ = 80°C	5	W
T_{stg}, T_j	Storage and Junction Temperature	–40 to +150	°C

THERMAL DATA

Symbol	Parameter		Value	Unit
$R_{th\ j-case}$	Thermal Resistance Junction-case	Max.	14	°C/W
$R_{th\ j-amb}$	Thermal Resistance Junction-ambient	Max.	80	°C/W

ELECTRICAL CHARACTERISTICS

For each channel, V_S = 24V, V_{SS} = 5V, T_{amb} = 25°C, unless otherwise specified

Symbol	Parameter	Test Conditions	Min.	TYp.	Max.	Unit
V_s	Supply Voltage		V_{ss}		36	V
V_{ss}	Logic Supply Voltage		4.5		36	V
I_s	Total Quiescent Supply Current	V_i = L I_o = 0 V_{inh} = H V_i = H I_o = 0 V_{inh} = H V_{inh} = L		2 16	6 24 4	mA
I_{ss}	Total Quiescent Logic Supply Current	V_i = L I_o = 0 V_{inh} = H V_i = H I_o = 0 V_{inh} = H V_{inh} = L		44 16 16	60 22 24	mA
V_{iL}	Input Low Voltage		-03.		1.5	V
V_{iH}	Input High Voltage	V_{SS} ≤ 7V V_{ss} > 7V	2.3 2.3		V_{ss} 7	V
I_{iL}	Low Voltage Input Current	V_{il} = 1.5V			-10	µA
I_{iH}	High Voltage Input Current	2.3V ≤ V_{IH} ≤ V_{ss} - 0.6V		30	100	µA
V_{inhL}	Inhibit Low Voltage		-0.3		1.5	V
V_{inhH}	Inhibit High Voltage	V_{SS} ≤ 7V V_{ss} > 7V	2.3 2.3		V_{ss} 7	V
I_{inhL}	Low Voltage Inhibit Current	V_{inhL} = 1.5V		-30	-100	µA
I_{inhH}	High Voltage Inhibit Current	2.3V ≤ V_{inhH} ≤ V_{ss} - 0.6V			±10	µA
V_{CEsatH}	Source Output Saturation Voltage	I_o = -1A		1.4	1.8	V
V_{CEsatL}	Sink Output Saturation Voltage	I_o = 1A		1.2	1.8	V
V_{SENS}	Sensing Voltage (pins 4, 7, 14, 17) (**)				2	V
t_r	Rise Time	0.1 to 0.9 V_o (*)		250		ns
t_f	Fall Time	0.9 to 0.1 V_o (*)		250		ns
t_{on}	Turn-on Delay	0.5 V_i to 0.5 V_o (*)		750		ns
t_{off}	Turn-off Delay	0.5 V_i to 0.5 V_o (*)		200		ns

* See figure 1
** Referred to L293E

TRUTH TABLE

V_i (each channel)	V_o	V_{inh} (**)
H	H	H
L	L	H
H	X (*)	L
L	X (*)	L

(*) High output impedance
(**) Relative to the considerate channel

1. INTRODUCTION AND OVERVIEW

This Medium Range Infrared sensor offers simple, user friendly and fast obstacle detection using infrared; it is non contact detection. The implementations of modulated IR signal immune the sensor to the interferences caused by the normal light of a light bulb or the sun light. The sensing distance can be adjusted manually. The product features include:

- **5V powered**, low current consumption, **less than 10mA**.
- 3 pin interface which are **signal**, **GND** and **5V**.
- Small **LED as indicator** for detection status.
- Obstacle detection up to **10cm**.
- **Adjustable sensing range** (2cm – 10cm).
- Small size makes it easy to assembly.
- Single bit output.
- Compatible with all types of microcontrollers.
- **Dimension:** 2.6cm x 2cm

3. PRODUCT SPECIFICATION AND LIMITATIONS

3.1 Theory of Operation

IR01A uses special sensor to modulate IR signal emitted from 2 IR transmitters and detects the modulated IR signal reflected back from a nearby object. This sensor has a built-in IR LED driver to modulate the IR signal at 38KHz to match the built-in detector. The modulated IR signal immunes the sensor from the interferences caused by the normal light of a light bulb or the sun light. The module will output a HIGH if no object is detected and a LOW if an object is detected.

3.2 Pin Definitions and Ratings

Pin	Name	Function
+	VCC	Connects to Vcc (+4V to + 6V)
-	Ground	Connects to Ground
s	Output signal	Connects to an I/O pin of microcontroller which set to INPUT mode (or transistor/MOSFET).

Table 3.1

Absolute Maximum Rating

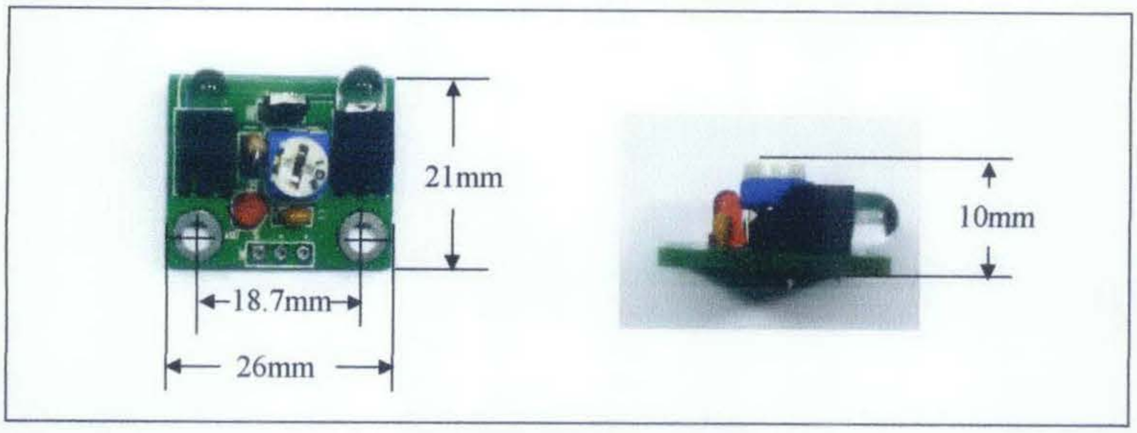
Parameter	Min	Max	Unit
Operating voltage	4	6	V
Sensing range	2	10	cm

3.3 Sensitivity

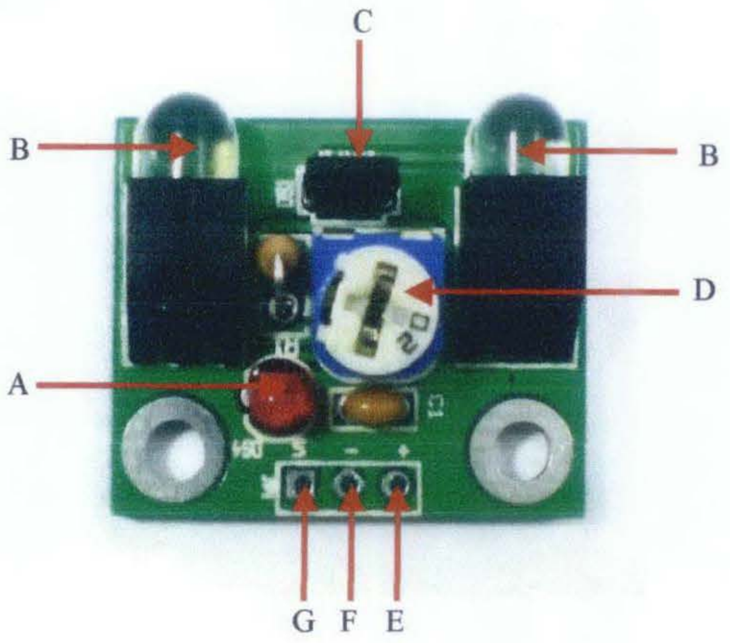
The Medium Range Infrared Sensor has a sensing range of approximately 2cm to 10cm. The sensitivity can vary with the reflectivity of the object and the ambient lighting. The modulated IR signal will reflect more on white surface and reflect less on black surface. The sensor is designed to adjustable sensing range. User may adjust sensing range by using the preset on IR01A for different application.

4. PRODUCT DIMENSIONS AND LAYOUT

4.1 Product Dimensions



4.2 Product Layout



Label	Function	Label	Function
A	Signal indicator LED	E	The hole to solder and connect VCC (+).
B	IR transmitter	F	The hole to solder and connect GND (-)
C	IR sensor	G	The hole to solder and connect output signal (s)
D	Preset		

A – is a signal indicators LED for IR01A. The LED will turn ON when signal is detected on IR01A.

B – are 2 IR transmitters, the output IR signal is modulated at 38Khz.

C – is IR sensor. This sensor modulates IR signal emitted from 2 IR transmitters and detects the modulated IR signal reflected back from a nearby object.

D – is a 1K Ohm preset for user to adjust the sensing range. The sensing range is 2cm – 10cm. (Performance of the sensor will vary with the reflectivity of the object and the ambient lighting.)

E – is a hole to solder and connect the power supply to IR01A. User may supply 4V-6V to IR01A, the typical voltage is 5V.

F – is a hole to solder and connect Ground to IR01A. User may connect the GND(-) of IR01A to the Ground (0V) of the control board.

G – is a hole to solder and connect the output signal from IR01A. User may connect the signal pin(s) from IR01A to an I/O pin of microcontroller which set to INPUT mode. The output signal of IR01A is LOW or 0V when an object detected.

APPENDIX G

INITIAL PROGRAMMING FOR TESTING MOVEMENT OF ROBOT WITH PIC 16F628A

```
//Nadhira Abdul Malek
// To test out the movement of the motor.
// Movements: Forward, Reverse, Turn right, Turn left.

#include <pic16f62xa.h>
#fuses INTRC_IO, NOWDT, NOPUT, PROTECT, NOBROWNOUT, MCLR

void delay_ms(unsigned int D);

void main(void)
{
        TRISB = 0x00; //make all bits of portB as outputs

        while(1)
        {
                //move forward
                PORTB = 0b00110101;
                delay_ms(200);

                //turn right
                PORTB = 0b00110001;
                delay_ms(200);

                //move forward
                PORTB = 0b00110101;
                delay_ms(200);

                //turn left
                PORTB = 0b00110100;
                delay_ms(200);

                //move forward
                PORTB = 0b00110101;
                delay_ms(200);

                //move backward
                PORTB = 0b00111010;
                delay_ms(200);

                //stop
                PORTB = 0x00;
                delay_ms(200);
        }
}


// Delay Function
void delay_ms(unsigned int D)
{
        unsigned int i, j;
        for( i=0; i<D; i++)
        for(j=0; j<1000; j++);
}
```

APPENDIX H

INITIAL PROGRAMMING FOR TESTING MOVEMENT OF ROBOT WITH PIC 16F887

```
// Nadhira Abdul Malek
// To test out the movement of the motor.
// Movements: Forward, Reverse, Turn right, Turn left.
// Compiler: Hitech-C

#include <stdio.h>
#include <htc.h>

// Configuration bit: use internal oscillator
__CONFIG(UNPROTECT & LVPDIS & BORDIS & MCLREN & PWRTDIS & WDTDIS & INTCLK );
#define _XTAL_FREQ  4000000

// Defining I/O Connections on MC40A
#define Led1              RB7          // General LED
#define Buzz              RB7          // Buzzer

#define EML               RC1          // Enable pin for left motor
#define EMR               RC2          // Enable pin for right motor
#define MR1               RB2          // Right motor
#define MR2               RB3          // Right motor
#define ML1               RB4          // Left motor
#define ML2               RB5          // Left motor

#define SenF              RC6          // Front sensor
#define SenL              RC7          // Left sensor
#define SenR              RC4          // Right sensor


// Declaring Sub-Functions
void delay(unsigned int D);            // delay function for 1 second
//void motor(unsigned char uc_left_motor_speed,unsigned char
uc_right_motor_speed);

// PWM functions
//void pwm_init(void);
//void set_pwmr(unsigned char uc_duty_cycle);
//void set_pwml(unsigned char uc_duty_cycle);

// Main Program
void main(void)
{

       TRISA = 0x00; // Make all bits of portA as outputs
       TRISB = 0x00;
       TRISC = 0xD0; // Set RC7, RC6, RC4 as inputs
       TRISD = 0x00;
       TRISE = 0x00;

       PORTA = 0;
       PORTB = 0;
       PORTC = 0;
       PORTD = 0;
       PORTE = 0;

       while(1)
       {
              //move forward
              MR1 = 1;
              ML1 = 1;
```

```
                delay(10);

                //turn right
                MR1 = 1;
                ML1 = 0;
                delay(5);

                //move forward
                MR1 = 1;
                ML1 = 1;
                delay(10);

                //turn left
                MR1 = 0;
                ML1 = 1;
                delay(5);

                //move forward
                MR1 = 1;
                ML1 = 1;
                delay(10);

                //move backward
                MR1 = 0;
                ML1 = 0;
                MR2 = 1;
                ML2 = 1;
                delay(5);

                //stop
                PORTC = 0x00;
                delay(2);
        }
}


// Delay Function
void delay(unsigned int D)
{
        unsigned int i, j;
        for( i=0; i<D; i++)    // # seconds
                for(j=0; j<10; j++)
                        _delay_ms(100);
}
```

APPENDIX I

MAZE ROBOT NAVIGATION SOURCE CODE

```
/*
// Nadhira binti Abdul Malek
// Date: August 2011
// Title: Navigation Algorithm for Maze Robot
//        through Event-Driven Programming
//        (Final Year Project)
//        Universiti Teknologi PETRONAS
// Compiler: CCS C Compiler
*/


#include <16F887.h>
#device  ADC=10
#fuses XT,NOWDT,NOPROTECT,NOLVP,NOPUT,NOBROWNOUT
#use delay(clock = 4000000)
#include <string.h>

// Defining I/O Connections on MC40A
#define Buzz          output_high(PIN_B7)        // Buzzer ON
#define Buzz0         output_low(PIN_B7)         // Buzzer OFF

#define EML           output_high(PIN_C1)        // Enable pin for left motor
#define EMR           output_high(PIN_C2)        // Enable pin for right motor

#define MR1           output_high(PIN_B2),output_low(PIN_B3)       // Right motor
#define  MR2             output_high(PIN_B3),output_low(PIN_B2)          // Right  motor
reverse
#define ML1           output_high(PIN_B4),output_low(PIN_B5)       // Left motor
#define  ML2             output_high(PIN_B5),output_low(PIN_B4)          // Left  motor
reverse
#define ML0           output_low(PIN_B4),output_low(PIN_B5)        // Left motor stop
#define MR0           output_low(PIN_B2),output_low(PIN_B3)        // Right motor stop

#define F             input(PIN_C6)        // Front sensor
#define L             input(PIN_C7)        // Left sensor
#define R             input(PIN_C4)        // Right sensor

#define dutyL         80     // duty cycle value = PR2 x %duty cycle
#define dutyR         90     // speed value => 0~102 where 51=> 50% and 102 =>100%

// State Definition
#define IDLE          1
#define FORWARD       2
#define RIGHT         3
#define LEFT          4
#define UTURN         5
#define FIN           6
#define END           7

// Declaring Sub-Functions of Robot Movement
void gofwd();
void turnleft();
void turnright();
void makeuturn();
void reverse();
void stop();

int k=0;


/****Main Program****/
void main(void)
{
   int state =IDLE;                // Initial state = IDLE

   setup_timer_2( T2_DIV_BY_1,102,5 );  //(PR2+1)*4*(1/Fosc)*prescaler = PWM period
   setup_ccp1(CCP_PWM);
   setup_ccp2(CCP_PWM);
```

```
// I/O Definition for all ports
set_tris_A(0x00);
set_tris_B(0x00);
set_tris_C(0xD0);   // Set RC7, RC6, RC4 as inputs (sensors)
set_tris_D(0x00);
set_tris_E(0x00);

output_A(0);       // RESET output
output_B(0);
output_C(208);     // Reset sensors F,L,R
output_D(0);
output_E(0);

while(1)
{
 if(F == 0)
 {Buzz;
 break;}
}
      // Speed of left and right motor based on PWM
      set_pwm1_duty(dutyR);
      set_pwm2_duty(dutyL);

   delay_ms(500);
   Buzz0;
   gofwd();

   while(state!=END)
   {
      output_C(208);

      // Maze navigation program based on FSM
      switch(state)
      {
           case IDLE:
           {
                  stop();
                  if (L == 0 && F == 1 && R == 0)
                  {
                     state =FORWARD;
                     break;
                  }
                  else
                  { break;}
           }

           case FORWARD:
           {
                 gofwd();

                  // No walls: Forward/Stop
                  if (L == 1 && F == 1 && R == 1)
                  {
                       while(1)
                       {
                       delay_ms(200);

                        if(L == 1 && F == 1 && R == 1)
                        {
                           k++;

                        }

                        else
                        {
                              state=FORWARD;
                              break;
                        }

                           if (k>5)
                           {
                              Buzz;
                              state = FIN;
                              break;
                           }
                        }
                  }
```

```
        // Wall on Right only: Forward
        else if (L == 1 && F == 1 && R == 0)
        {
              state=FORWARD;
        }

        // Wall at Front only: Right
        else if (L == 1 && F == 0 && R == 1)
        {
              state=RIGHT;
        }

        // Walls on Right and Front: Left
        else if (L == 1 && F == 0 && R == 0)
        {
              state=LEFT;
        }

        // Wall on Left only: Right
        else if (L == 0 && F == 1 && R == 1)
        {
              state=RIGHT;
         }

        // Walls on Left and Right: Forward
        else if (L == 0 && F == 1 && R == 0)
        {
              state=FORWARD;
        }

        // Walls on Left and Front: Right
        else if (L == 0 && F == 0 && R == 1)
        {
              state=RIGHT;
        }

        // Walls on everyside: U-turn
        else if (L == 0 && F == 0 && R == 0)
        {
              Buzz;           // Buzz while u-turning
              state=UTURN;
        }

        else{}
        break;    //to choose only either one of the cases, so we break
} // break from FORWARD

case LEFT:
{
        reverse();
        turnleft();
        state =FORWARD;
        break;
}

case RIGHT:
{
        reverse();
        turnright();
        state =FORWARD;
        break;
}

case UTURN:
{
        reverse();
        makeuturn();
        Buzz0;
        state =FORWARD;
        break;
}
```

```
              case FIN:
              {
                     delay_ms(500);
                     Buzz0;
                     stop();
                     state=END;
                     break;
              }
          } // end of select-case

         delay_ms(500);     //pause between cases
       //}  // end of main navigation program
   }


}     // end of Main Program


/***Subfunctions***/
void gofwd()
{
   MR1;
   ML1;
}

void turnleft()
{
   MR1;
   ML2;
   delay_ms(1300);
   MR1;
   ML1;
   delay_ms(300);
}

void turnright()
{
   MR2;
   ML1;
   delay_ms(1300);
   MR1;
   ML1;
   delay_ms(300);
}

void makeuturn()
{
   MR1;
   ML2;
   delay_ms(2500);
   MR1;
   ML1;
   delay_ms(1000);
}

void reverse()
{
   MR2;
   ML2;
   delay_ms(500);
}

void stop()
{
   MR0;
   ML0;
}
```