

CONTROL OF MOBILE ROBOT PLATFORM

By

STELLA LAU KUN SHII

FINAL REPORT

**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**

© Copyright 2011

by

Stella Lau Kun Shii, 2011

CERTIFICATION OF APPROVAL

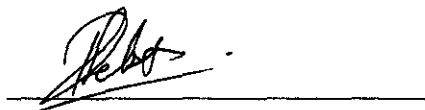
CONTROL OF MOBILE ROBOT PLATFORM

by

Stella Lau Kun Shii

A Final Report 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:



Mr Patrick Sebastian

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.



Stella Lau Kun Shii

ABSTRACT

This report is meant to report everything that was done in the whole of Final Year Project 1. First Chapter of the report states the detailed background study and problem statement for this project. It also includes the objectives and scope of study of this project. Second chapter of this report is literature review done for this project. This includes the basic study and theory of mobile robot and its important parts such as the DC motor. Third chapter records the methodology of this project including both Final Year Project 1 and Final Year Project 2. Chapter includes the hardware specifications and information of tools and software utilized in this project. Chapter Four shows the schematic designs of this project and some elaborations of the schematic designs. Chapter Five reports the hardware design of the project. This includes some important parts such as the rotary encoder. Chapter Six shows the firmware development of the project and some results from the firmware execution while Chapter Seven gives some details on the problem faced throughout the course of developing this mobile robots and solutions used to solve the problems. Lastly, Chapter Eight concludes the project and gives some recommendations for further study and improvements of the project.

ACKNOWLEDGEMENT

First and foremost, I would like to thank God for His grace and guidance through the whole course of this project development. His constant and unfailing presence at times when things seemed impossible has been a great comfort and encouragement.

Secondly, I would like to thank my supervisor, Mr Patrick Sebastian for all his advices. From the very choosing of project title till the end of FYP1 and now to the completion of FYP2, I thank him for the guidance that he had given. I am very grateful for the space he has given me to do this project on my pace but still give very valuable suggestions at times I got stuck on a certain stage.

Next, I would like to thank the FYP committee for their hard work in organizing the flow of both FYP1 and FYP2. Their constant reminder on e-learning has been very useful and the dates set between each due dates are comfortable.

Last but not least, I'm grateful to a few faithful friends who had been tirelessly helped me with the testing of my prototypes, taking videos of the testing over and over again into late nights.

TABLE OF CONTENTS

ABSTRACT	v
ACKNOWLEDGEMENT	vi
LIST OF TABLES	1
LIST OF FIGURES.....	2
CHAPTER 1 INTRODUCTION	3
1.1 Background of Study.....	3
1.2 Problem Statement	3
1.3 Objective	4
1.4 Final Year Project 1.....	4
1.5 Final Year Project 2.....	5
1.6 Scope of Study.....	5
1.6.1 To carry out literature review on building of mobile robot	5
1.6.2 To Analyze Suitable Parts, Gear and Interface Device for this Project	6
1.6.3 To Analyze Suitable Tools and Software for Project Design..	6
1.6.4 To Design a Main Circuit-Devices Connection.....	6
1.6.5 To Build the Most Suitable Hardware for This Mobile Robot	6
1.6.6 To Write a Suitable Firmware to Run Every Part of This Mobile Robot	6
CHAPTER 2 LITERATURE REVIEW	7
2.1 Basic Theory of Mobile Robots	7
2.2 Basic Theory of DC Motor.....	8
2.3 Basic Theory of Motor Driver.....	9
CHAPTER 3 METHODOLOGY	10
3.1 Procedure Identification	10
3.2 Hardware Specification	13
3.2.1 Tamiya Double Gearbox.....	13
3.2.2 L298 Motor Driver.....	13
3.2.3 Rotary Encoder	16

3.3 Tools And Equipments Used.....	17
3.3.1 Cytron’s USB ICSP PIC Programmer UIC00A	17
3.3.2 Cadsoft Eagle Freeware	18
3.3.3 Custom Computer Service (CCS) C Compiler	18
CHAPTER 4 SCHEMATIC DESIGN.....	19
4.1 Main Schematic Design.....	19
4.2 Main Circuit to Interface Device Design	20
CHAPTER 5 HARDWARE DESIGN.....	21
5.1 Voltage Regulator.....	21
5.2 Main Circuit	21
5.3 Rotary Encoder.....	22
CHAPTER 6 FIRMWARE DEVELOPMENT	24
6.1 Pin Declarations	24
6.2 DC Motor Control	25
6.3 LCD Input Prompt.....	26
6.4 Storing Input Data	27
6.5 Keypad Scanning.....	28
6.5.1 Row Scan	28
6.5.2 Column Scan.....	28
6.6 Firmware for Rotary Encoder (Going Straight)	30
6.7 Firmware for Rotary Encoder (Turning Left/Right)	33
6.8 Combined Firmware.....	35
6.8.1 Firmware Flowchart.....	35
6.8.2 Firmware Description	36
CHAPTER 7 PROBLEMS FACED AND SOLUTIONS	39
7.1 Friction on Course Surface.....	39
7.1.1 Mobile robot not moving in straight line	39
7.1.2 DC motor is exhausted.....	39
7.1.3 Turning is inaccurate	40
7.2 Insufficient Current Supply	40
7.3 Circuit Instability.....	41

CHAPTER 8 RECOMMENDATION AND CONCLUSION.....	42
8.1 Recommendations	42
8.1.1 Using Rotary Encoder with Higher Resolutions	42
8.1.2 Replacing Keypad Wires with Wireless Connections	42
8.1.3 Sensor to Avoid Obstacle	42
8.2 Conclusion.....	43
REFERENCES.....	44
APPENDICES.....	45
Appendix A FULL FIRMWARE.....	46
Appendix B Gantt chart(FYP1).....	63
Appendix C Gantt chart(FYP2).....	64

LIST OF TABLES

Table 1 Pin Functions.....	15
Table 2 DC motor control	16
Table 3 Rotary Encoder Part Description	17
Table 4 Keypad Pin Control.....	25

LIST OF FIGURES

Figure 1 Speed performance with PWM.....	8
Figure 2 Speed control with PWM.....	8
Figure 3 Tamiya Double Gearbox.....	13
Figure 4 L298 motor driver.....	13
Figure 5 L298's full bridge drivers and its connections to DC motor.....	14
Figure 6 L298 pin diagram.....	15
Figure 7 Rotary Encoder.....	16
Figure 8 Rotary Encoder Sensor Board.....	17
Figure 9 Cytron USBFigure 8 ICSP PIC Programmer UIC00A.....	17
Figure 10 PIC - Motor Driver schematic.....	19
Figure 11 PIC - Keypad and LCD Connections.....	20
Figure 12 Soldered Voltage Regulator Circuit.....	21
Figure 13 Assembled/Soldered main circuit.....	21
Figure 14 Rotary Encoder.....	22
Figure 15 Rotary Encoder 2.....	23
Figure 16 Firmware excerpt showing pins declarations.....	24
Figure 17 LCD Pin Initialization.....	25
Figure 18 DC Motor Control with PWM.....	26
Figure 19 LCD Input Prompt.....	26
Figure 20 Data Storing.....	27
Figure 21 LCD Display.....	27
Figure 22 Keypad Row Scanning.....	28
Figure 23 Keypad Column Scanning.....	28
Figure 24 Flowchart for keypad scanning.....	29
Figure 25 Firmware for rotary encoder (forward).....	30
Figure 26 Flowchart for forward routine with rotary encoder.....	31
Figure 27 Rotary encoder calibration.....	32
Figure 28 Firmware for turning left/right with rotary encoder.....	33
Figure 29 Flowchart for turn left/right routine.....	34
Figure 30 Full Firmware Flowchart.....	36

CHAPTER 1

INTRODUCTION

1.1 Background of Study

Looking at the toy industry today, one of the most commonly sold mobile robots are the wireless remote controlled cars. These remote controlled mobile robots simply respond the user through a control panel such as joystick, going in directions and avoiding obstacles as the user directs. Therefore, there is little chance for purchaser or user of controlling the mobile robots' navigation program wise.

However, in this new age of technology, it is important to instill structured thinking skills into children at young age, enhancing the development of problem solving ability in them. A variety of how mobile robots could be played may help to keep a child's mind challenged and wanting for more.

1.2 Problem Statement

Children today are the leader and hope of the future. It is important to shape their mind and character since young, to guide them to the right path and thinking. Especially in the world today where technology plays a big role in human's life, it is always good to get children to understand the fundamentals of technology. It is crucial also not to let children take for granted the easy life they have thanks to technology today.

A remote controlled mobile robot can be very fun, and very satisfying to have the car's navigation under control. However, it is not very building to a child's mind. How well they perform on a track depends basically on how fast and how finite the user's fingers can control it.

At other times, in competitions involving these remote controlled cars, such as the infamous RC Race, results depends heavily on how much money a person spends on his car. The quality of his car's gears and motors, the quality of the casing and the wheels. This may build an unhealthy mindset in a child to keep asking for money from parents for upgrades of their car, to keep up with the other players.

Therefore, an alternative has to be created to keep children playing it constantly challenged to achieve a higher level, yet at the same time teaches them some valuable principles in life besides developing their thinking skills.

1.3 Objective

The objective of this project is to build a controllable mobile robot platform. This mobile robot will be able to understand simple inputs of commands or instructions, and executes it later after the commands are confirmed by the user. Series of the instructions will be executed without human intervention. Results will tell if the user, which in this project we targets the children, was able to give the mobile robot sufficient or accurate instructions to perform required task or to get to the destination from a starting point.

1.4 Final Year Project 1

Final Year Project is separated into two parts, completed over two semesters. Final Year Project 1 is done on the first semester of final year study and Final Year Project 2 is to be completed in the second semester. Each semester has its own target to meet.

The basic and foundations of the project was covered in Final Year Project 1. Work done includes background of study, brief literature review on current technology and toy industry. Then, suitable parts and components, including motor type, microcontroller, appropriate rotary encoder and motor driver were selected. Basic hardware was then assembled and firmware was written to test the functionality of the hardware. Straightforward firmware was written to test the microcontroller's input/output pins and to find the right duty cycle of PWM(pulse width modulation) to get the motor running.

1.5 Final Year Project 2

Final year Project 2 includes drawing the schematic circuit to connect rotary encoder, keypad and lcd display to main circuit. Choose the most suitable pins on the microcontroller for the control of each application, and makes sure that sending or receiving of signals will not be confused later. Considerations are later done on the hardware of the project, on how the rotary encoder, keypad and lcd display should be connected to the mobile robot. Separate testing firmware is written for each different external device. Each rotary encoder, keypad and lcd display is tested with a straightforward firmware to see if they work. Firmware is later developed to suit the applications of this project.

1.6 Scope of Study

1.6.1 To carry out literature review on building of mobile robot

A full understanding on how a basic mobile robot works is crucial in designing a new one. Important parts such as the dc motor, the motor driver, the mobile robot's platform and the simplest algorithm of a mobile robot. Innovations can only begin after understanding on how the foundation of mobile robot's design works.

1.6.2 To Analyze Suitable Parts, Gear and Interface Device for this Project

For the mobile robot to work efficiently, suitable parts and gears have to be selected. To build a mobile robot with constant interface with user, suitable interface device is needed for it to be more user friendly, eliminating any chance of frustration when communicating with the mobile robot.

1.6.3 To Analyze Suitable Tools and Software for Project Design

For ease of designing, programming and debugging, the right software tools have to be chosen to avoid the need of trouble occurring from the design software itself.

1.6.4 To Design a Main Circuit-Devices Connection

Before having progress on the hardware, designs of pin connections for each device have to be done carefully to make sure each pin is connected well to the microcontroller and sending or receiving of control signal will not cause confusion to any other devices. Things like microcontroller's internal functions(ADC, PWM etc), and compiler's build in driver/functions are taken into considerations when choosing pins for certain devices.

1.6.5 To Build the Most Suitable Hardware for This Mobile Robot

The basic hardware, main circuit with the robot motor, motor driver, microcontroller and voltage regulator were built. Now, based on the further developed schematic design, other control devices such as the rotary encoder and keypad needs to be attached to the mobile robot, and it has to be carefully placed so they would not disturb the rotary of the motor or the navigation of the mobile robot.

1.6.6 To Write a Suitable Firmware to Run Every Part of This Mobile Robot

Referring to the schematic designed, pins are initialized accordingly and firmware is developed for the mobile robot to work as desired. A simple and straightforward one is first written and further developed after each function is tested to be working and successful.

CHAPTER 2

LITERATURE REVIEW

2.1 Basic Theory of Mobile Robots

Mobile robot is a type of robot with its own engine and power, enabling it to move around its environment independently. To make this possible, the robot needs to be able to travel or navigate in a range and accuracy depending on its navigational ability. Navigational ability will vary between different types of mobile robots depending on the size of the robot and the type of task it is meant to carry out. [1]

There are several types of mobile robot navigation. Below shows some of the example;

The Line Following Robot, is a type of mobile robot that follows a visual line painted or embedded in the floor. Most of these robots operate on a simple "keep the line in the center sensor" algorithm.

Autonomous Robot, randomized or guarded. Autonomously Randomized Robot bounce off walls, having simple algorithm of sense and turn into another direction, in a certain degree. Autonomously Guarded Robot will have a certain amount of information on where it should turn and has a certain goal to achieve.

Manual Remote or Tele-op and Guarded Tele-op is what we know more commonly as remote controlled car. A popular production in the toy industry. Manual remote is our usual robot or car that is totally under control of a driver with a joystick or other control device. Guarded Tele-op on the other hand has the ability to sense and avoid obstacles but otherwise navigate as driven. [2]

2.2 Basic Theory of DC Motor

A direct current (DC) motor is a straight forward and simple electric motor that uses electricity and a magnetic field to produce torque, which turns the motor. This means that it generates torque directly from DC power supplied to the motor by using internal commutation, stationary permanent magnets, and rotating electrical magnets.

The speed of the DC motor is directly proportional the supply voltage. This means that if the DC motor needs 9V to run at maximum speed, reducing the supply voltage to 4.5V will also halve the DC motor's speed. However, simply adjusting the supply voltage to the DC motor to control its speed is inefficient. A better way would be to switch the motor's supply on and off very quickly. If the switching is fast enough, the motor doesn't notice it, it only notices the average effect.

Therefore, best way to control the DC motor with the microcontroller will be with Pulse Width Modulation (PWM). Figure below shows the performance of DC motor's speed controlled by PWM.

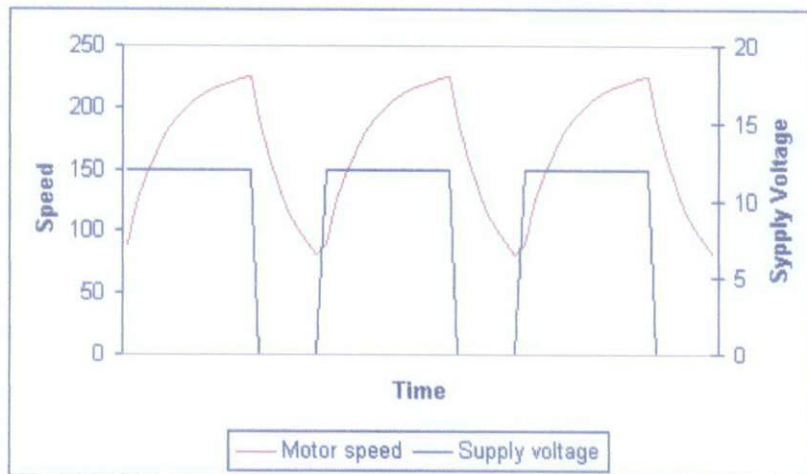


Figure 1 Speed performance with PWM

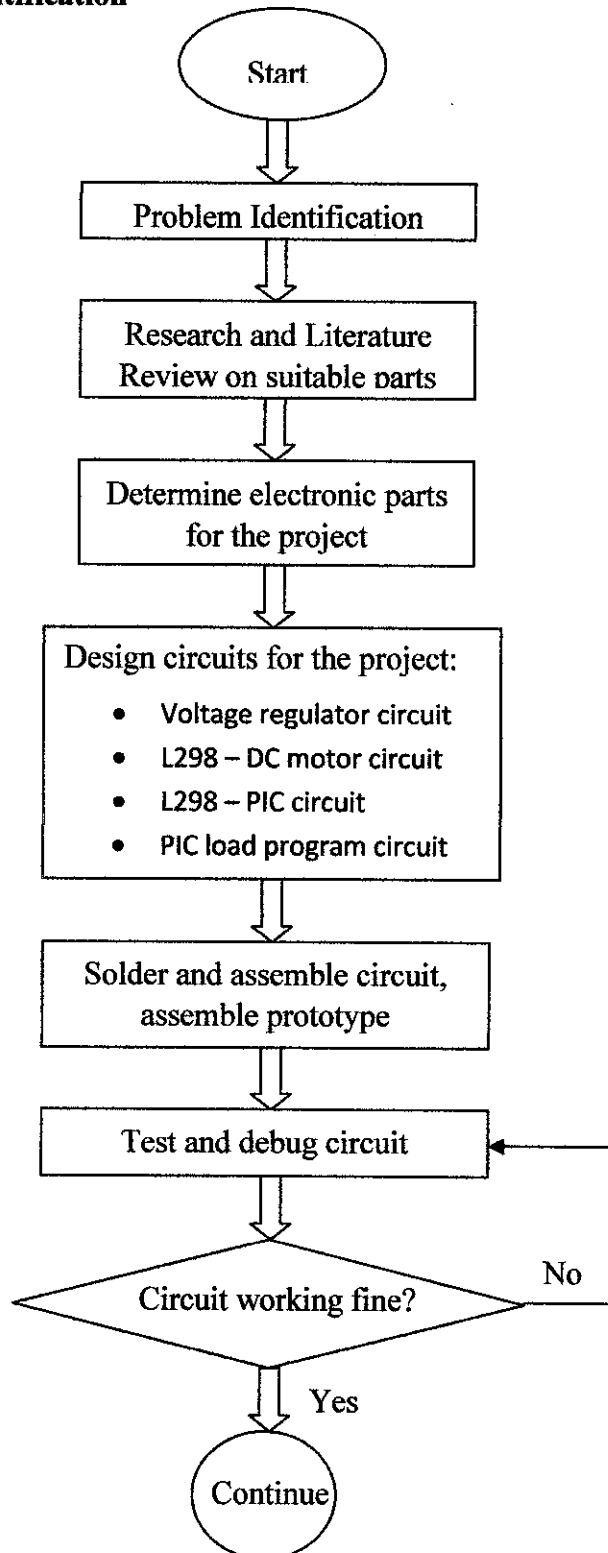
From Figure 1, we can see that the average speed is around 150, although it may vary a little. If the supply voltage is switched fast enough, it won't have time to change speed much, and the speed will be quite stable. To increase or decrease the average speed depends on the PWM's duty cycle. This is the principle of switch mode speed control. Thus the speed is set by PWM – Pulse Width Modulation. [3]

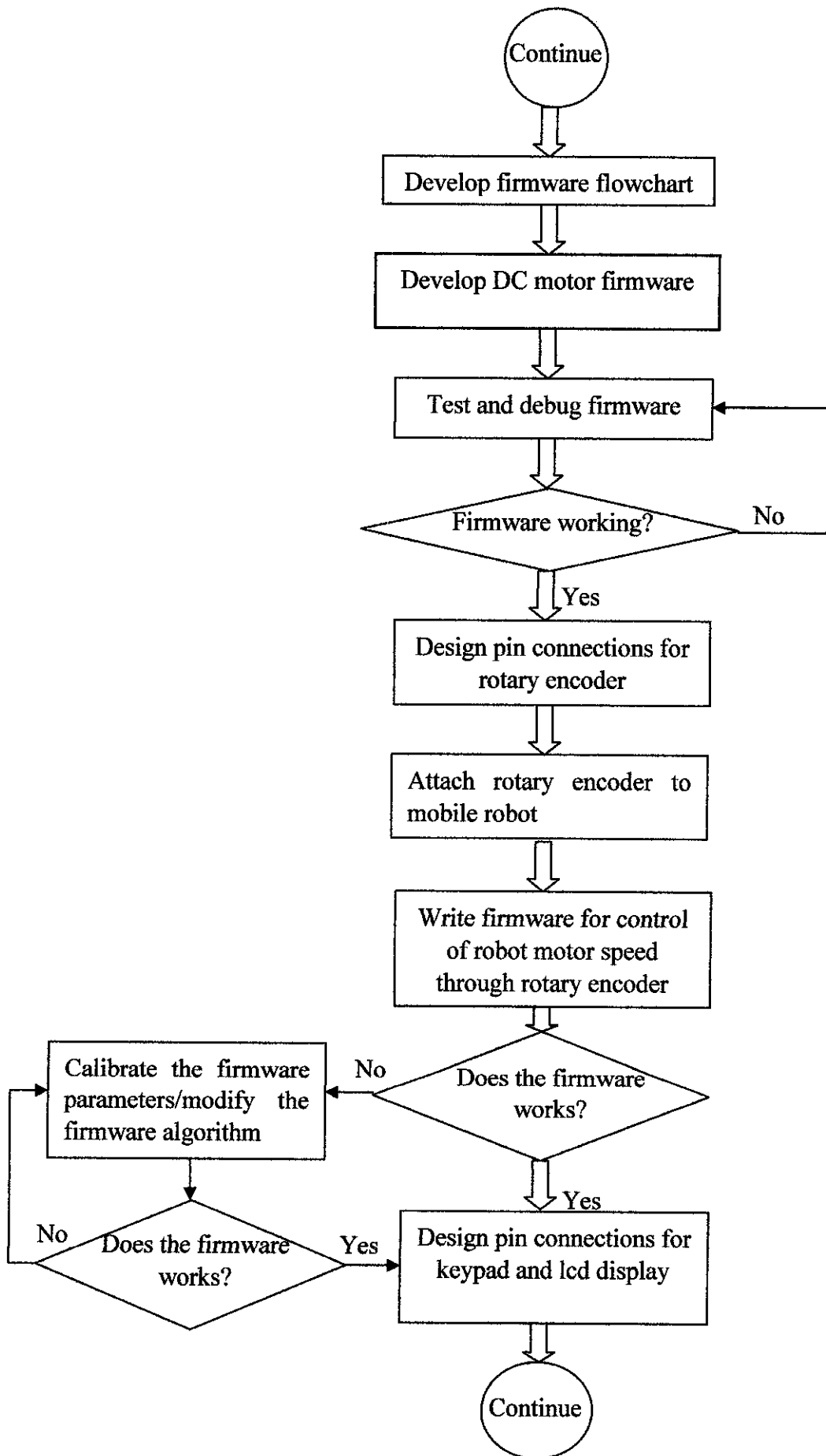
2.3 Basic Theory of Motor Driver

Motor drivers are essentially current amplifiers. Their function is to take a low-current control signal, and turn it into a proportionally higher-current signal that can drive a motor. Besides driving the motor, a motor driver may includes functions like manual or automatic means for starting and stopping the motor and selecting forward or reverse rotation. [4]

CHAPTER 3 METHODOLOGY

3.1 Procedure Identification





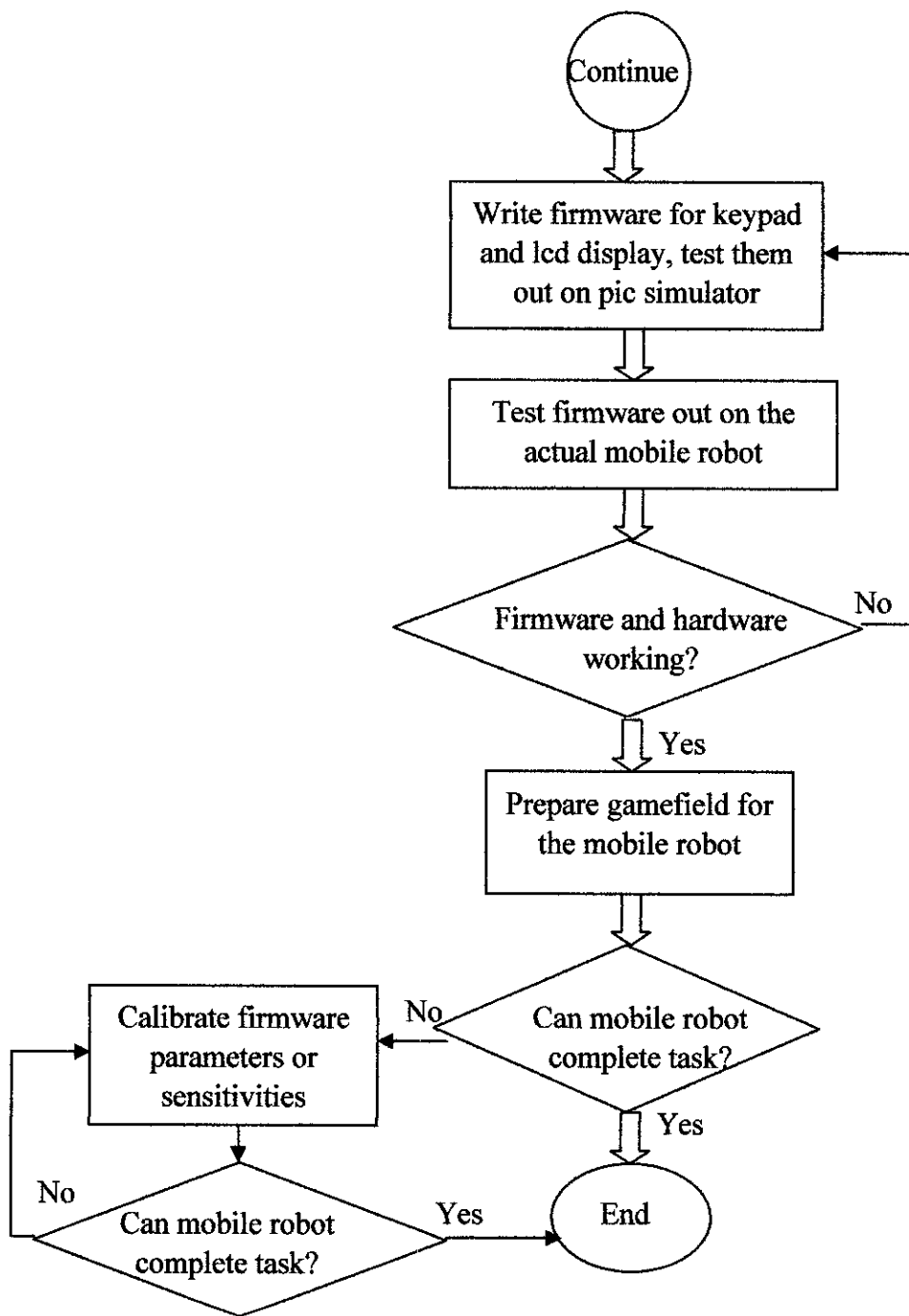


Figure 2 Project Flowchart

3.2 Hardware Specification

3.2.1 Tamiya Double Gearbox

DC motor that was chosen for this project is the Tamiya Double Gearbox set, shown in Figure below.

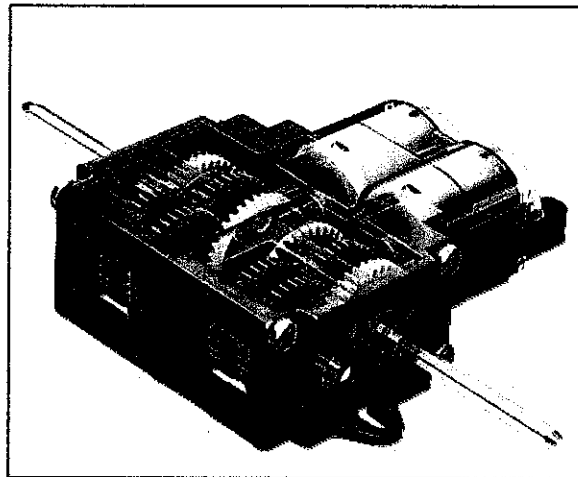


Figure 3 Tamiya Double Gearbox

Tamiya Double Gearbox is a compact unit with two independent motors and gear trains. The possible gear ratio configurations are 12.7:1, 38:1, 115:1, and 344:1. It works just like any DC motor as explained in the section before.

3.2.2 L298 Motor Driver

The motor driver chosen for this project is the L298 motor driver, shown in figure below.

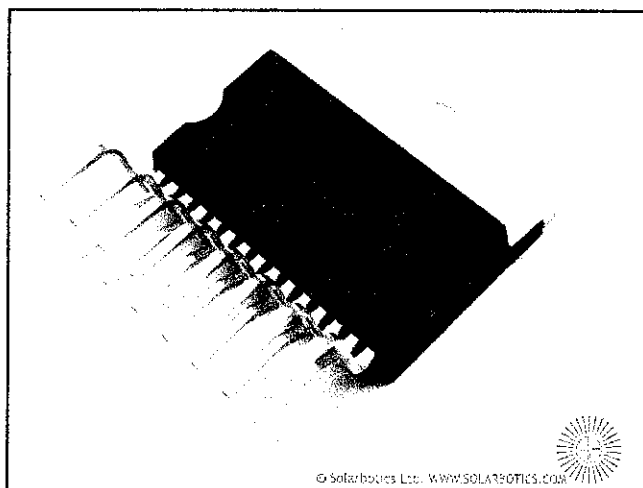


Figure 4 L298 motor driver

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and Power SO20 packages. For this project, the 15-lead Multiwatt L298 (as shown in Figure 1-2) is chosen.

It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. In this project, L298 is used to drive the bidirectional DC motor.

Two enable inputs are provided to enable or disable the device independently of the input signals. This means that L298 can controls two DC motor separately at the same time.

Diagram of L298's dual full bridge driver and its connections to DC motor is as shown below. Pin descriptions can be found on the next page.

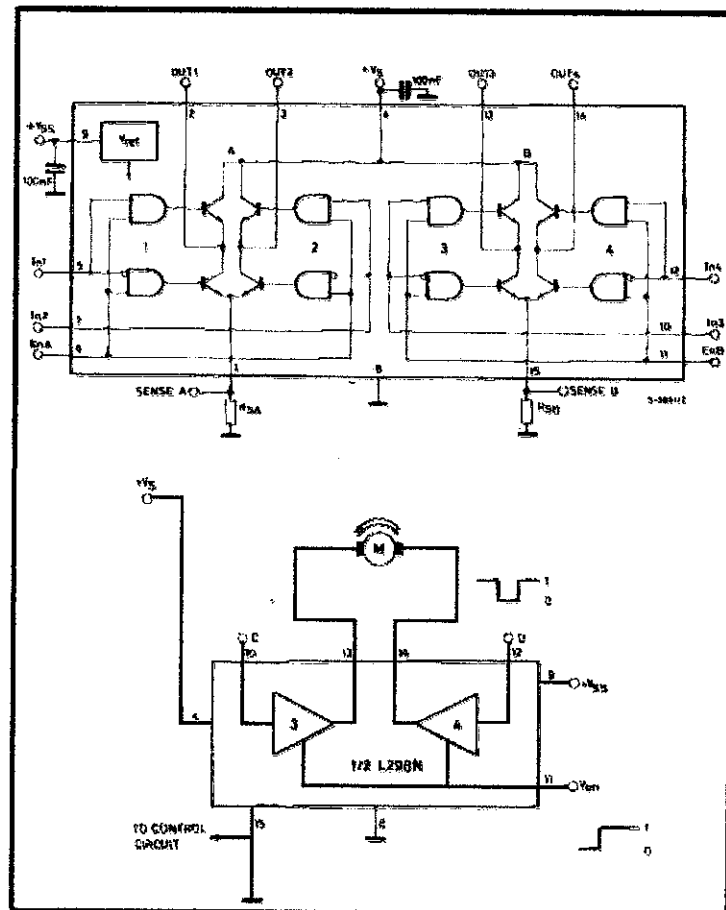


Figure 5 L298's full bridge drivers and its connections to DC motor

Pin Functions

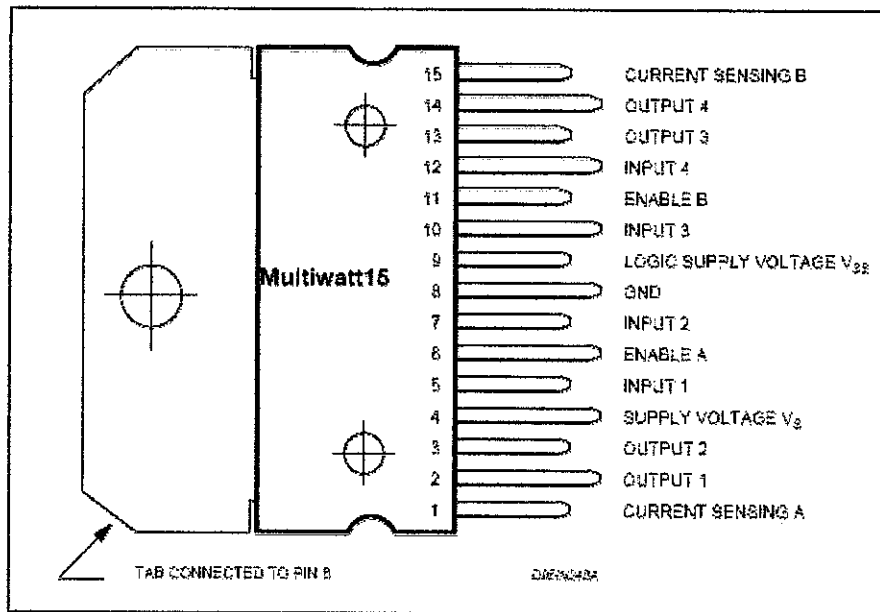


Figure 6 L298 pin diagram

As stated earlier, the L298 has two drivers that can drive two motors separately. Therefore, there are two sets of similar pins on for each driver which we may call driver A and driver B. To make matter more understandable, focus is made on one of the driver, driver A for explanation of pin functions.

Referring to Figure 6,

Table 1 Pin Functions

Pin Number	Pin Name	Functions
15	Current Sensing B	Control the current of the load
13,14	Output3, Output 4	Outputs of bridge A, connected to motor to drive the motor
4	Vs	Supply Voltage for Power Output Stages
10,12	Input 3, Input 4	TTL Compatible inputs to bridge A, input from PIC
8	GND	Ground
9	VSS	Supply Voltage for Logic Blocks
11	Enable B	Enable/ Disable bridge A (receives PWM from PIC)

Inputs of PWM into Enable A will determine the speed of the motor, while combinations of Input 1-Input 2 will determine rotation directions of the motor. How the direction is controlled is shown in the table below. [5]

Table 2 DC motor control

Inputs		Functions
Ven = H	1 = H; 2 = L	Forward
	1 = L; 2 = H	Reverse
	1 = 2	Fast Motor Stop
Ven = L	C = X; D = X	Free Running Motor Stop

H = High

L = Low

X = Don't care

1 = Input 1

2 = Input 2

3.2.3 Rotary Encoder

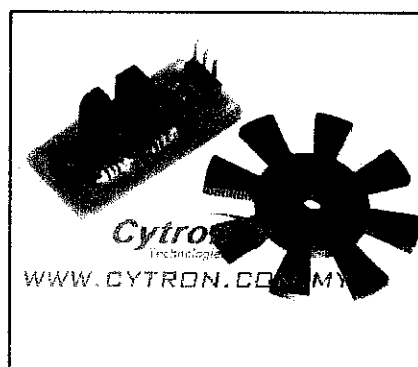


Figure 7 Rotary Encoder

This rotary encoder comes with a sensor board and a slotted disc of eight slots. Data of rotary motion is converted into a series of electrical pulses readable by the controller.

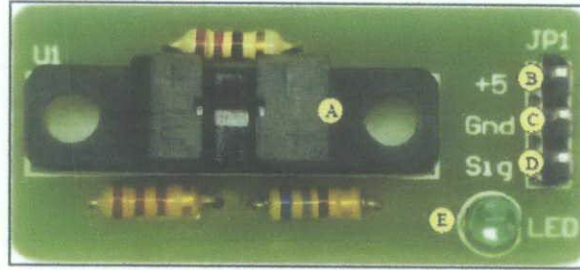


Figure 8 Rotary Encoder Sensor Board

Table 3 Rotary Encoder Part Description

Label	Part Name	Fuction
A	Optical Sensor	Detect missing slot of disc when the disc rotate, further generate pulses at signal pin
B	+5V input supply	+5V is should be connected to this pin
C	GND/ negative supply	Should be connected to negative terminal of supply
D	Signal Output/pulse output	Signal output of sensor board. This pin is internally pulled up to 5V, thus no extra component is needed for sensor to be connected to controller.
E	LED indicator	LED will on if the disc does not block the optical sensor

3.3 Tools And Equipments Used

3.3.1 Cytron's USB ICSP PIC Programmer UIC00A

Cytron USB ICSP PIC Programmer UIC00A is a PIC USB programmer able to program popular Flash PIC MCU which includes PIC12F, PIC16F and PIC18F family. It can also program 16bit PIC MCU.



Figure 9 Cytron USBFigure 8 ICSP PIC Programmer UIC00A

3.3.2 Cadsoft Eagle Freeware

Cadsoft Eagle is a tool used to design printed circuit board, PCB. It has three main modules.

Layout Editor

- maximum drawing area 1.6 x 1.6m (64 x 64 inch)
- resolution 1/10,000mm (0.1 micron)
- up to 16 signal layers
- conventional and SMT parts
- comes with a full set of part libraries
- easily create your own parts with the fully integrated library editor
- undo/redo function for ANY editing command, to any depth
- script files for batch command execution
- copper pouring
- cut and paste function for copying entire sections of a drawing
- design rule check
- interactive Follow-me Router (requires the Autorouter module)

Schematic Editor

- up to 999 sheets in one schematic
- electrical rule check
- gate and pinswap
- create a board from a schematic with a single command

Autorouter

- ripup&retry router
- up to 16 signal layers
- routing strategy driven by user definable cost factors

3.3.3 Custom Computer Service (CCS) C Compiler

CCS provides a complete integrated tool suite for developing and debugging embedded applications running on Microchip PIC[®] MCUs and dsPIC[®] DSCs. The heart of this development tools suite is the CCS intelligent code optimizing C compiler which gives freedom to developers to concentrate on design functionality instead of having to become an MCU architecture expert.

4.2 Main Circuit to Interface Device Design

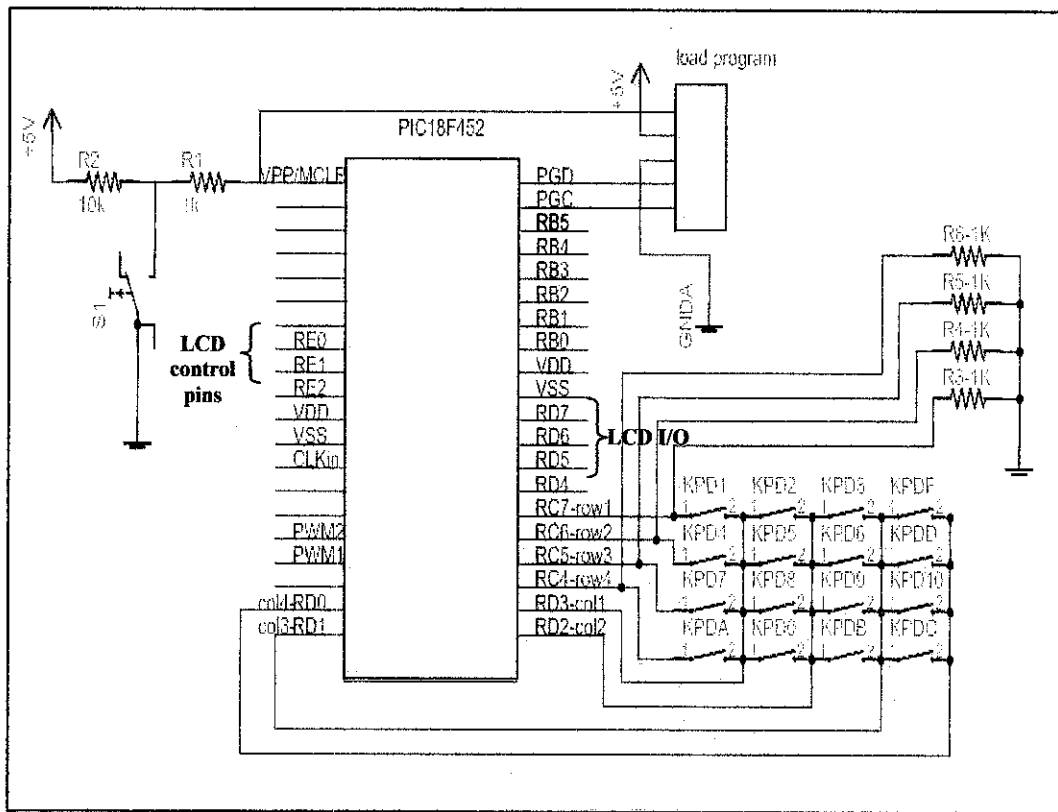


Figure 11 PIC - Keypad and LCD Connections

Figure above shows that the rows of the keypad are connected to pins RC4-RC7 while the columns are connected to pins RD0-RD3. The rows are connected to ground, giving high output alternatively one after another as means of scanning for inputs.

Pins RE0, RE1 and RE2 are connected to the LCD display's control pins while pins RD4-RD5 are connected as data output to LCD display.

CHAPTER 5

HARDWARE DESIGN

5.1 Voltage Regulator

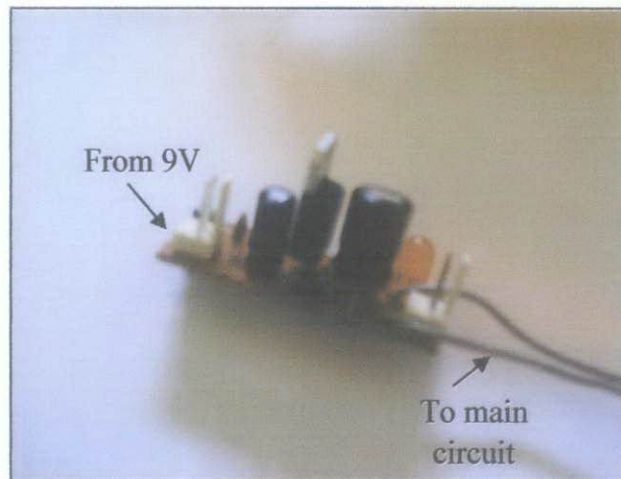


Figure 12 Soldered Voltage Regulator Circuit

One side of the VR has connector receiving 9V voltage from the main source or battery. Another side of VR has single core wire connected to the main circuit to provide main circuit and microcontroller with constant 5V.

5.2 Main Circuit

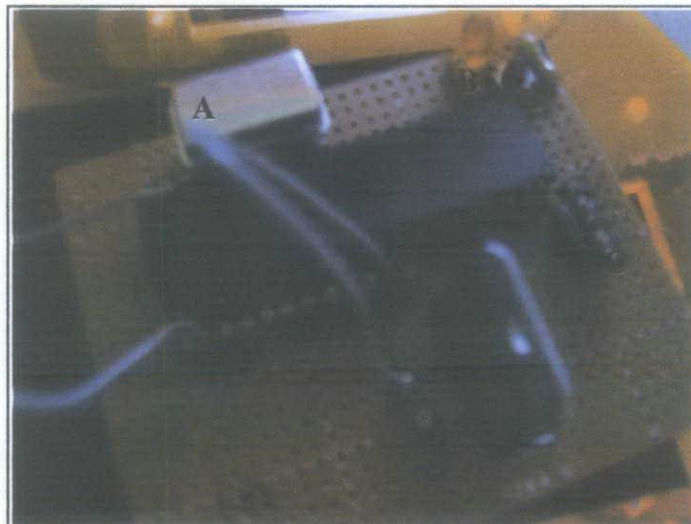


Figure 13 Assembled/Soldered main circuit

Referring to Figure 13;

A = crystal oscillator, 4MHz

B = master clear circuit

C = connector to programmer, USB ICSP PIC Programmer UIC00A

D = motor driver, ST L298

E = connector ports, connecting microcontroller's input/output ports to other devices or components via single core wire

F = microcontroller, PIC16F877A

Crystal oscillator determines the maximum speed the mobile robot can operate. Master clear circuit comes with a master clear switch for manual restart of program or the mobile robot itself at any time, even during execution. A connector to program is soldered directly on the main circuit board so program or firmware can be programmed directly to microcontroller without needing to take off and fix on the microcontroller repetitively. Some connections between microcontroller and other devices are currently temporal to allow changes made throughout the project development.

5.3 Rotary Encoder

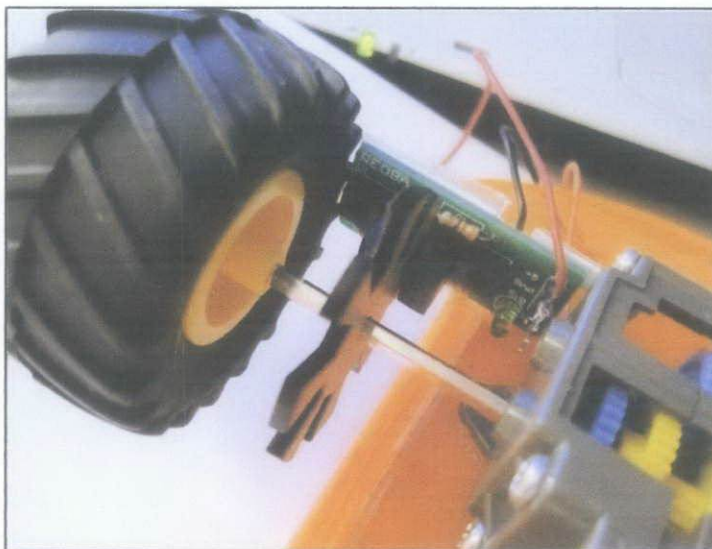


Figure 14 Rotary Encoder

The gear-plate used to trigger the sensor is placed on the shaft connecting the motor gear and the wheel. Therefore, it will move in parallel to the wheel and send signals to the microcontroller keeping track of wheel rotations.



Figure 15 Rotary Encoder 2

LED will light up when the sensor is blocked by the disc, and a low signal is sent to the PIC. In contrary, when sensor is not blocked, LED does not light up and a high signal is sent.

CHAPTER 6

FIRMWARE DEVELOPMENT

6.1 Pin Declaration

```
#include <18F452.h>
#fuses XT,NOBODT,NOPROTECT,NOLVP,NOPUT,NOBROWNOUT,NODEBUG
#use delay(clock = 4000000) //24MHz
#include "C:\Documents and Settings\User\My Documents\Microcontroller\lab4\Flexlcd2.c"
//#include <string.h>

//pin define for keypad
#define col1 PIN_D3
#define col2 PIN_D2
#define col3 PIN_D1
#define col4 PIN_D0
#define row1 PIN_C7
#define row2 PIN_C6
#define row3 PIN_C5
#define row4 PIN_C4

//pin define for motor
#define in1 PIN_B1 //set B6 to be output1
#define in2 PIN_B2 //set B7 to be output2
#define in3 PIN_B4 //set B4 to be output3
#define in4 PIN_B5 //set B5 to be output4

//pin define for rotary encoder
#define encoder1 PIN_B0
#define encoder2 PIN_B3
#define max 255
```

Figure 16 Firmware excerpt showing pins declarations

As shown in figure above, in1 and in2 determines direction of the motor rotary on the right while in3 and in4 determines the rotary direction of motor on the left. Encoder1 is for receiving encoder signals on the right while encoder2 is for receiving signals on the left.

Pin RD0 – RD3 are used to control the columns of the keypad while Pin RC4 – RD7 are for the control of keypad rows. Table below shows the keypad pin control.

Table 4 Keypad Pin Control

Row	Column	Key Represented	Row	Column	Key Represented
1	1	1	3	1	7
1	2	2	3	2	8
1	3	3	3	3	9
1	4	F	3	4	D
2	1	4	4	1	A
2	2	5	4	2	0
2	3	6	4	3	B
2	4	E	4	4	C

```
#define LCD_DB4 PIN_D4
#define LCD_DB5 PIN_D5
#define LCD_DB6 PIN_D6
#define LCD_DB7 PIN_D7

#define LCD_E PIN_E0
#define LCD_RS PIN_E1
#define LCD_RW PIN_E2
```

Figure 17 LCD Pin Initialization

Figure above shows the pin initialization for the LCD display. A built-in driver of CCS is used for the LCD display application these initializations are done on that separate source file.

6.2 DC Motor Control

Referring to Figure 15 again,

In1 = Input 1

In2 = Input 2

In3 = Input 3

In4 = Input 4

Combinations of In1/In2 and In3/In4 determines if that particular motor rotates forward or backward.

```

void forward(void)
{
    set_pwm2_duty(120);
    set_pwm1_duty(120);
    output_high(in1);
    output_low(in2);
    output_high(in3);
    output_low(in4);
    // encoder();
}

void reverse(void)
{
    set_pwm2_duty(100);
    set_pwm1_duty(100);
    output_low(in1);
    output_high(in2);
    output_low(in3);
    output_high(in4);
}

void turn_right(void)
{
    set_pwm2_duty(0);
    set_pwm1_duty(80);
    output_high(in1);
    output_low(in2);
    output_high(in3);
    output_low(in4);
}

void turn_left(void)
{
    set_pwm2_duty(80);
    set_pwm1_duty(0);
    output_high(in1);
    output_low(in2);
    output_high(in3);
    output_low(in4);
}

void stop(void)
{
    set_pwm2_duty(0);
    set_pwm1_duty(0);
    output_high(in1);
    output_high(in2);
    output_high(in3);
    output_high(in4);
}

```

Figure 18 DC Motor Control with PWM

When mobile robot is in forward or reverse mode, both PWM are set to the same duty cycle and changes in in1,in2 and in3,in4 will determine if the rotation of the wheels are forward or backward. When the mobile robot needs to turn left, PWM will have decreased or in this case, zero duty cycle on the left wheel. Bigger gap between voltage high periods lowers the voltage resulting in lower speed in that DC motor. Same thing applies to if mobile robot needs to turn to the right, PWM supplied to the right driver/DC motor will be zero.

6.3 LCD Input Prompt

```

for (k=0;k<200;k++)
{
    for (l=0;l<3;l++)
    {
        lcd_gotoxy(1,1);
        lcd_putc("Pls enter instr");
        lcd_gotoxy(1,2);
        lcd_putc("& steps desired");
        // delay_ms(3000);
    }
    lcd_gotoxy(1,2);
    lcd_putc("\f");
}

```

Figure 19 LCD Input Prompt

Figure 19 shows output that will be displayed on the LCD display upon power on, prompting for user to key in their desired instructions(forward/reverse/left/right) and the number of steps(1 step = 1 inch).

6.4 Storing Input Data

```
while(key != 13)
{
    key = 0;
    while (key == 0)
    {
        keypad();
    }
    if (key == 10 || key == 11 || key == 12 || key == 13 || key == 14 || key == 15 || key == 16)
    {
        store_instr[h]=key;
        h++;
    }
    else
    {
        store_step[i] = key;
        i++;
    }
    delay_ms(500);
    keypad_lcd();
    delay_ms(1500);
}
```

Figure 20 Data Storing

Figure 20 shows a loop for scanning user input. While (key==0) makes sure that the firmware will keep scanning for keypad input until a ‘press’ is detected. Keypad scanning is done by the keypad() function. Data received is then stored in the store[] array and function keypad_lcd() will display to users the instructions that had been keyed in. The receiving and storing of input data will continue until key 13(A) is pressed. Firmware will then continue to the next part.

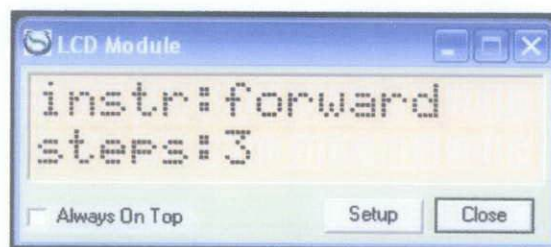


Figure 21 LCD Display

Figure 21 shows an example of how the LCD display will look like as user keys in input of desired instructions and steps.

6.5 Keypad Scanning

6.5.1 Row Scan

```
void row_scan(void)
{
    switch(k)
    {
        case 0: output_high(row1); break;
        case 1: output_high(row2); break;
        case 2: output_high(row3); break;
        case 3: output_high(row4); break;
    }
}
```

Figure 22 Keypad Row Scanning

Figure 22 shows a switch case to determine which row to scan at one time. The rows will be given high output alternatively, repeatedly and at the same time, the column will also be scanned.

6.5.2 Column Scan

```
for(k=0; k<4; k++)
{
    row_scan();

    if (k == 0)
    {
        if (input(col1))
        {
            // while (input(col1));
            key = 1;
        }

        if (input(col2))
        {
            // while (input(col2));
            key = 2;
        }

        if (input(col3))
        {
            // while (input(col3));
            key = 3;
        }

        if (input(col4))
        {
            // while (input(col4));
            key = 10;
        }
        delay_ms(100);
        output_low(row1);
    }
}
```

Figure 23 Keypad Column Scanning

Figure 23 shows a firmware routine of scanning for column pressed. Since only one row is given high output at a time, only the column in that row is given consideration at the time. For example, as shown in Figure 21, when $k=0$ and the first row is given high output, if any of the column in that row is pressed, that particular PIC pin connected to the keypad will receive a high input. The high input received will determine the key representation to be stored in the `store[]` array. At the end of the routine, the row's output will become low and the next row will be given a high output.

The commented whiles such as `while(input(col1))` is to give an option for the keypad press to be high-to-low edge triggered. This means that data will only be received by the PIC once the key is let go.

Figure 24 shows a flowchart of the keypad scanning routine. The routine will continue until key A (on keypad) or when `key=13` is detected in the main function.

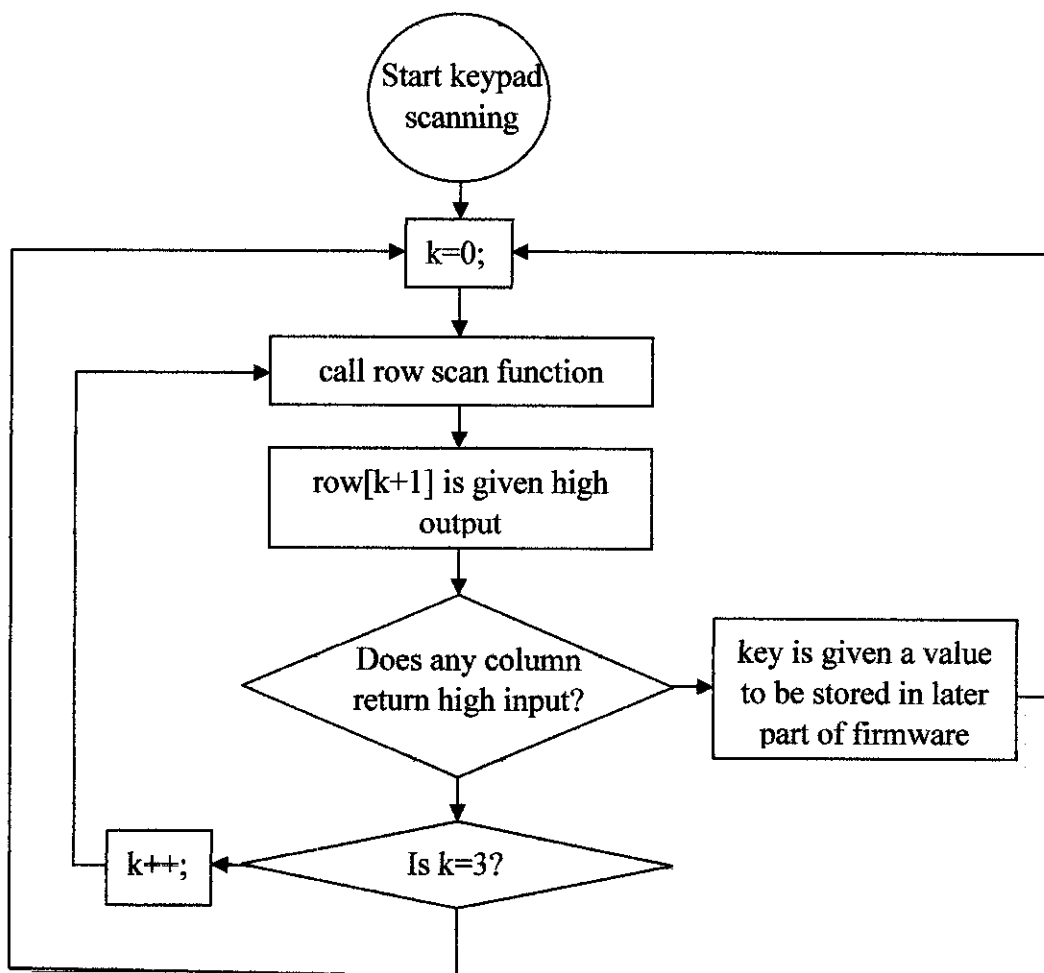


Figure 24 Flowchart for keypad scanning

6.6 Firmware for Rotary Encoder (Going Straight)

```
void encoder(void)
{
    keypad_lcd();
    forward();

    for(g=1;g<step+1;g++)
    {
        do
        {
            if(input(sensor))
                break;
            if(input(encoder1))
            {
                while(input(encoder1));
                count1 = count1 + 1;
                keyEn1 = count1;
                encoder1_lcd();
            }
            if(input(encoder2))
            {
                while(input(encoder2));
                count2 = count2 + 1;
                keyEn2 = count2;
                encoder2_lcd();
            }

            if(count1 > count2)
            {
                slight_right();
                delay_ms(5);
            }
            else if(count2 > count1)
            {
                slight_left();
                delay_ms(5);
            }
            else
            {
                forward();
            }
        }
        while((count1 < 9) && (count2 < 9));
        if(count1 < count2)
        {
            count1 = 1;
            count2 = 0;
            keyEn1 = count1;
            encoder1_lcd();
            keyEn2 = count2;
            encoder2_lcd();
        }
        else if(count2 < count1)
        {
            count1 = 0;
            count2 = 1;
            keyEn1 = count1;
            encoder1_lcd();
            keyEn2 = count2;
            encoder2_lcd();
        }
    }
}
```

Figure 25 Firmware for rotary encoder (forward)

Figure 25 shows a firmware for the rotary encoder. Encoder1 represents encoder on the right while encoder2 represents encoder on the left. Whenever one side rotates faster than the other by 1 count, the faster one will slow down for the slower one to catch up. The turning used for going-forward calibrating is different than the fulls top on one side turning because the abrupt stop on one side may cause the mobile robot to off track on the other side. Therefore, functions `slight_right()` and `slight_left()` is used instead. These functions can be found in the full firmware in APPENDIX A.

Flowchart on the next page shows how the forward routine in the firmware above works, followed by some elaborations.

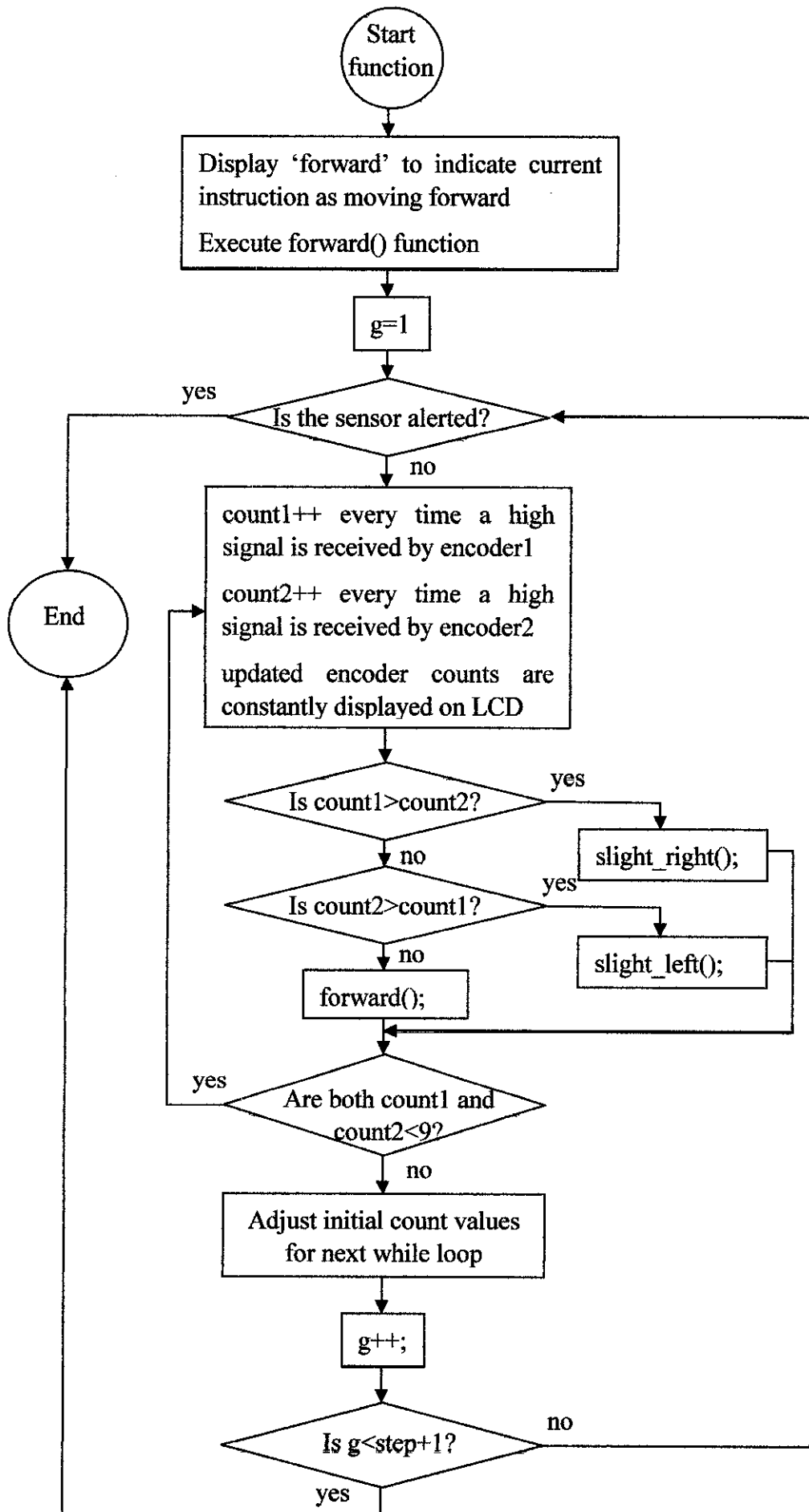


Figure 26 Flowchart for forward routine with rotary encoder

A full rotary gives 8 counts which is also equivalent to the distance of a step defined for this mobile robot. As shown in the firmware in Figure 24, the DC motor on each side will continue to rotate until both signal counts has reached 9. When the counts reach a maximum of 60000, both counts will start from 0, to prevent any confusion caused by register flow over.

The `if(input(sensor))` is used to check for signals from sensor if mobile robot happens to run into any obstacles. If obstacle is detected (sensor is triggered), then the mobile robot will come to a stop, to prevent forcing forward against the obstacle which may cause damage to the mobile robot.



Figure 27 Rotary encoder calibration

Figure 27 shows how the rotary encoder is supposed to work. If the wheel rotation on one side is too much faster than the other that it caused the mobile robot to sway off line, the signals given by the rotary encoder is supposed to make the firmware to self correct and the mobile robot is supposed to turn back into the line.

6.7 Firmware for Rotary Encoder (Turning Left/Right)

```

for (j=0; j<h; j++)
{
    key = store_instr[j];
    motor_lcd();
    step = store_step[j];

    if (key == 16)
    {
        for (g=1;g<step+1;g++)
        {
            do
            {
                if(input(encoder2))
                {
                    while(input(encoder2));
                    count2 = count2 + 1;
                }
                keyEn2 = count2;
                encoder2_lcd();
            }
            while(count2 < 9);
            count2 = 0;
        }
    }

    if (key == 12)
    {
        for (g=1;g<step+1;g++)
        {
            do
            {
                if(input(encoder1))
                {
                    while(input(encoder1));
                    count1 = count1 + 1;
                }
                keyEn1 = count1;
                encoder1_lcd();
            }
            while(count1 < 8);
            count1 = 0;
        }
    }
}

```

Figure 28 Firmware for turning left/right with rotary encoder

Firmware in Figure 28 shows how the turning of the mobile robot is calibrated. It has been tested that for the rotary encoder to send 8 signals, wheel rotation is sufficient for the mobile robot to make a 45degree turn. Therefore, every one step keyed in by the user represents a 45degree turn, and if the user desire for the mobile robot to make a 90degree turn, at step: (2) has to be keyed in. Please refer to flowchart in Figure 29 for better understanding.

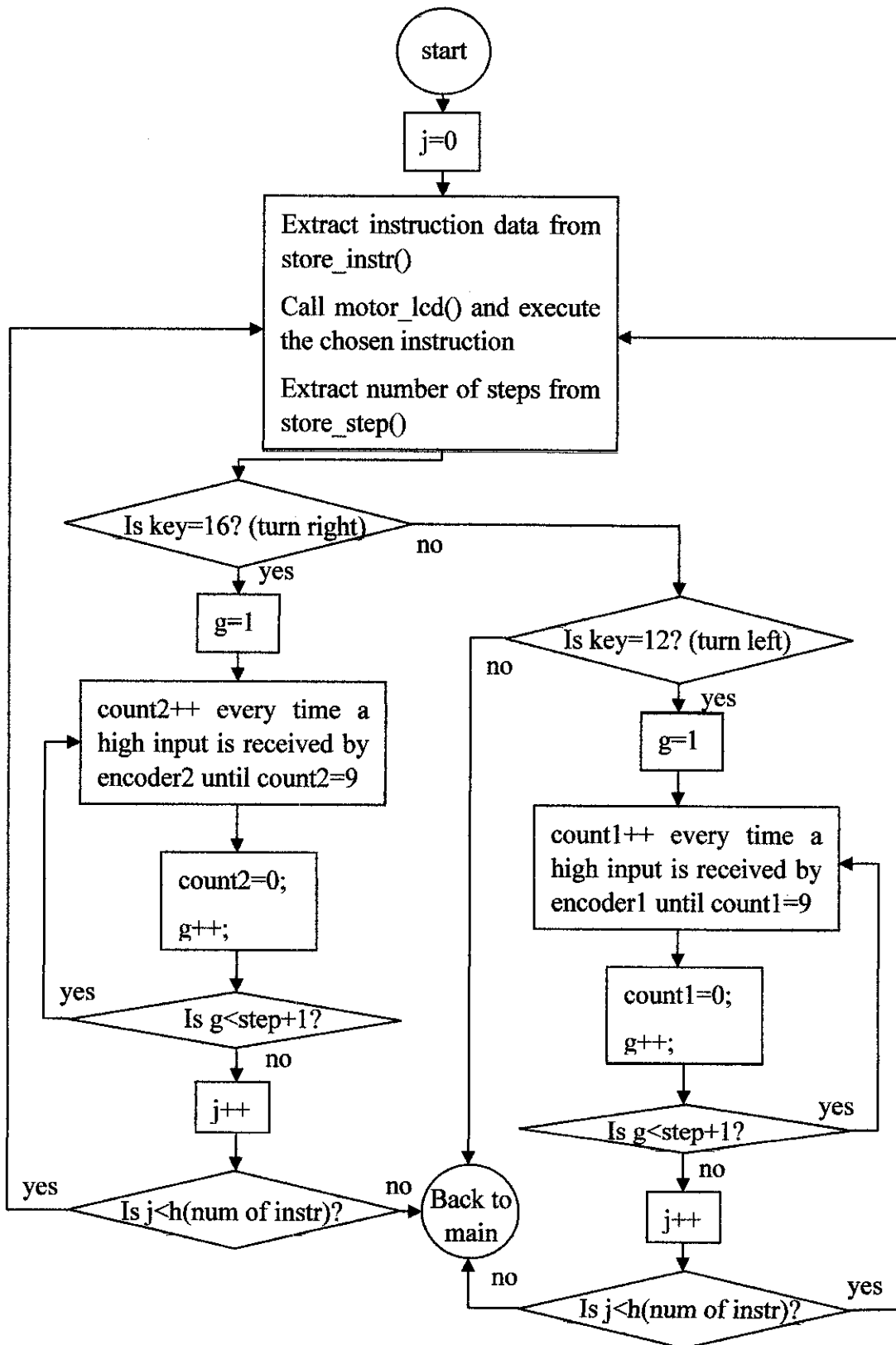
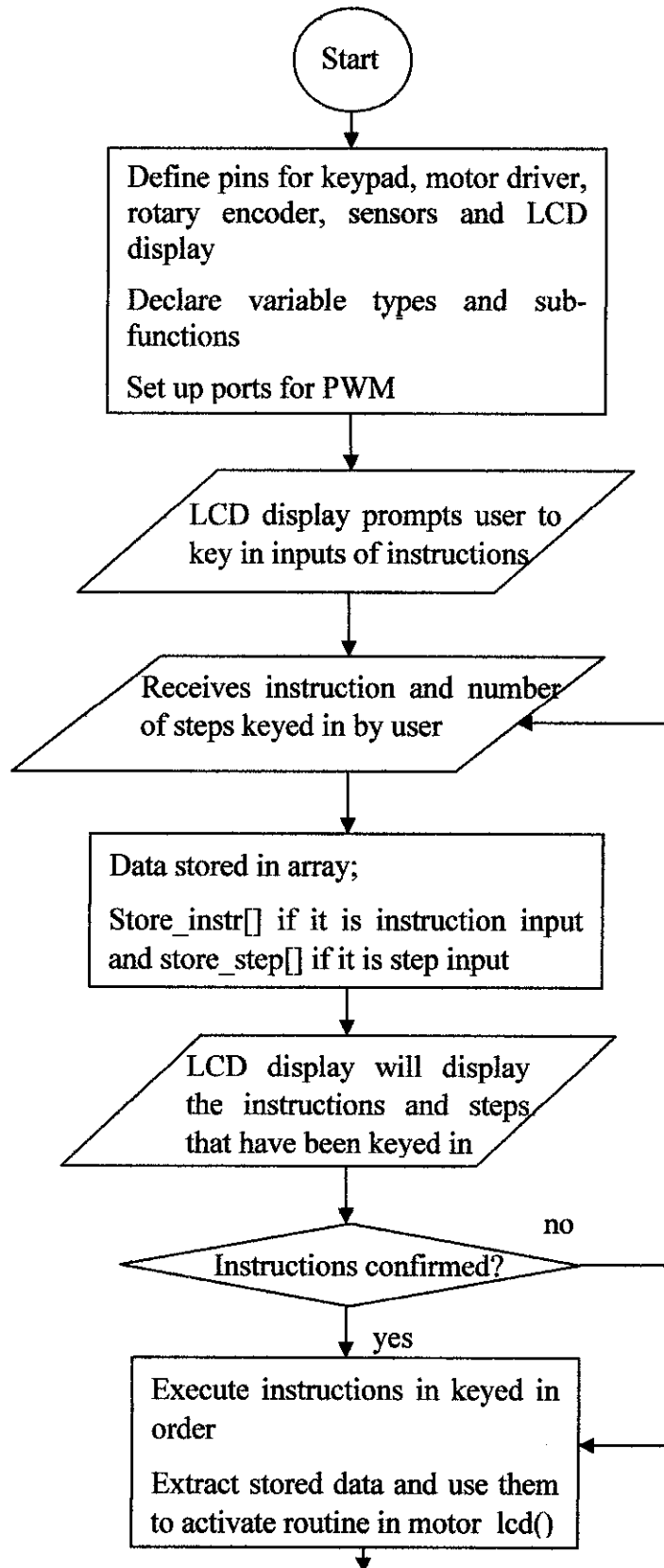


Figure 29 Flowchart for turn left/right routine

Count condition for both right and left wheel is different because time taken for the DC motor to stop after reaching the condition is slightly different. Therefore, for the wheel that takes a little more time to stop, the condition is 1 count less than the other, to balance the turning result.

6.8 Combined Firmware

6.8.1 Firmware Flowchart



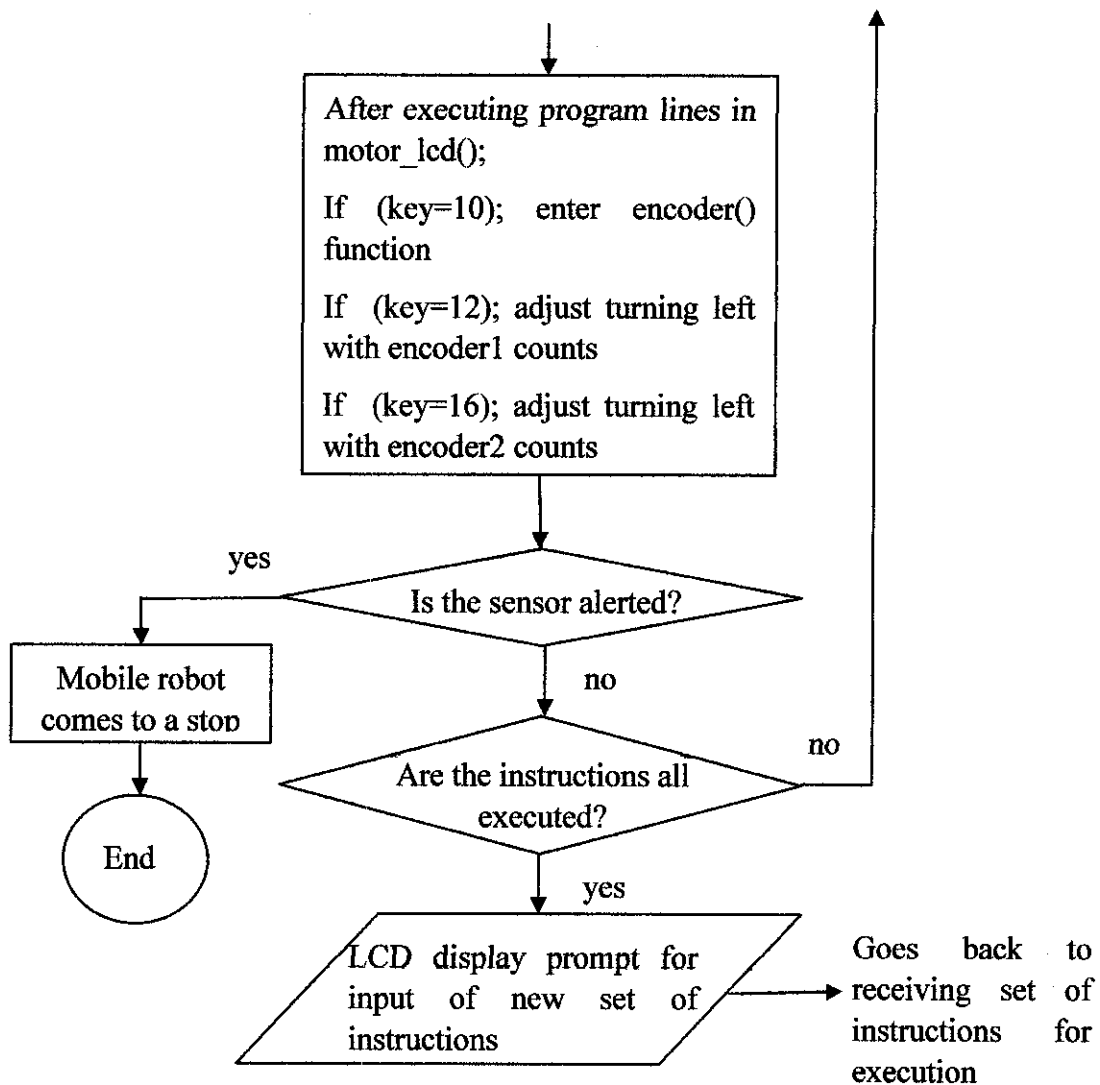


Figure 30 Full Firmware Flowchart

6.8.2 Firmware Description

Referring to the firmware shown in APPENDIX A, below are detailed description of the flow of the firmware. Numberings on the side of the firmware is meant for the ease of reference in the discussions of the firmware that follows.

From line 8 to line 21, it shows the pin declarations for keypad matrix, encoders and motor drivers. From line 27 to line 28, it declares the datatype for variables used in the firmware. Store_instr[30] is used to store the input instructions while store_step[30] is used to store the input steps. Array shows that it can store up to 30 insrtuctions. From line 30 to line 40, it shows the declarations of functions of

sub functions existing in the firmware, that will be called from time to time by the main function to execute specific task or read external input signals. From line 52 to line 60, it shows the port setups for timer for PWM and also shows that the initial duty cycle for both PWM is set to zero(0). Line 62 shows the initialization function for LCD display.

Line 67 to 78 will result in the LCD display showing “Pls enter instr & steps desired” as a prompt to the user to key in the desired set of instructions and steps for the instructions. Instructions that the users can key in include forward, reverse, left and right, and the steps available range from one to nine. Each step represents one inch of distance for forward and reverse while one step for turning right and left represents 45 degree turning. Therefore, to make a 90 degree turning, user will have to key in 2 steps for after keying in turn left or turn right.

Line 89 to 112 shows the reading of data from the keypad. Scanning of keypad (done by function keypad()) will be done continuously as long as value in key is 0, indicating that no key has been pressed. Once a key is pressed by the user, firmware will check if it belongs to a instruction key or a step key. If it is either, 10, 11, 12, 14, 15 or 16, the data will be stored into store_instr[h] as an instruction while if the key pressed is neither of those above, the data will be stored in store_step[i] as a step. For each instructions and steps keyed in, the LCD display will display the steps that was keyed in. This is done with the keypad_lcd() function. After keying in a set of instruction and step, user can continue to key in new instructions until key A(13) is pressed indicating the user has confirmed the set of instructions wanted. Firmware will then jump out of the (while(key==0)) loop to continue to the next instruction.

Line 121 to 125 will have the LCD display showing the user “Execute instr from the start” indicating that the mobile robot will now execute the series of instructions.

Values in the register storing signal data from rotary encoder is first cleared. Data from store_instr[] and store_steps[] are extracted. Data from store_instr[] will be used by function motor_lcd to activate the DC motor. Data from store_step[] is used to determine the duration each instruction is executed, thus determine the distance or

the turning angle for each instruction. All these are controlled by commands from line 144 to line 284. How far the mobile robot moves at 1 step is controlled by the signals received from the rotary encoder as explained in the sub-function sections earlier. After each instruction is executed, the mobile robot will stop for a short while to prevent messy navigation or error caused by uneven stopping and starting time by each wheel or DC motor.

How the keypad(), keypad_lcd(), row_scan(), forward(), reverse(), turn_left(), turn_right(), stop(), and encoder() functions works has been explained in the sections before this. One changes made in the encoder() function is the threshold for the firmware to detect mobile robot not moving in a straight line. Margin of 15 counts have been reduced to just 1 count as the firmware have proved to respond fast enough in this small and sensitive change or difference in the wheel rotations.

CHAPTER 7

PROBLEMS FACED AND SOLUTIONS

7.1 Friction on Course Surface

On the initial state of prototype testing, it was done mainly on tiled floor where the surface is smooth. For accuracy purpose, PWM duty cycle is lowered so signal reading from rotary encoder and respond can be done smoothly. However, when the mobile robot was tested out in the lab where the floor surface is course, it ceases to work. Two problems arise from this.

7.1.1 Mobile robot not moving in straight line

Most of the time, only one of the DC motor will rotate, causing the mobile robot to run off course or that huge calibration is needed for the mobile robot to get back on the track. As the PWM duty cycle is increased, even though both motor can rotate when the mobile robot is placed on the floor, it's moving too fast that the wheel calibration to stay in a straight line became a challenge.

7.1.2 DC motor is exhausted

Due to the need to overcome friction, current supplied to the DC motor is increased. Even though it results in the rotation of the DC motor, however, the increased current exhaust the DC motor as in a short time, some smell is detected from the DC motor. Therefore, it is not a good idea at all to increase the supply of current to the DC motor.

To solve this problem, wheels are changed. The mobile robot is initially attached with soft rubber wheels with air inside aiming to absorb any bumpiness on travelled surface. However, due to the friction problem, the wheel is changed to a hard rubber wheel. This reduces greatly the friction absorbed by the wheel.

7.1.3 *Turning is inaccurate*

The difference of friction on various surface also caused the previous firmware to not work accurately when it comes to turning. Previously, turning was made based on delays, and even though the time of delay was calibrated for it to turn nicely at wanted angle, once the surface friction is different, the turning angle was also affected.

To solve this problem, firmware was modified to use the signals from the rotary encoder as a guide to how much the mobile robot turns. No matter how fast or how slow the wheel rotates, the signals from the rotary encoder will not vary and thus, the turning became more accurate.

7.2 Insufficient Current Supply

At initial state of combining the applications of the mobile robot, even though the user interface and command execution works fine separately, when they are combined, things became uncontrollable. LCD display will show trigger error, where instructions and steps will appear without the keypad being pressed.

Later, it is found out that the current supplied to the circuit became insufficient when the voltage source has to supply to both the circuit (including LCD display) and the DC motor. This causes the circuit to not work properly. At the same time, values of the pulled-down resistor used by the keypad was also too large.

Therefore, to solve the problem, the resistors were changed to a smaller value. Then, the source supplied to the circuit is separated. A 9-volt battery is supplied to the circuit's voltage regulator for the use of the circuit while the the DC motor is supplied with a different power supply.

7.3 Circuit Instability

The circuit is sometimes instable due to amateur wiring and soldering. Sometimes, the turning and moving of the mobile robot from places to places also caused the connections to come loose.

To solve this problem, the circuit has to be constantly tested and debugged to ensure that it is in working condition.

CHAPTER 8

RECOMMENDATION AND CONCLUSION

8.1 Recommendations

8.1.1 Using Rotary Encoder with Higher Resolutions

Rotary encoder with higher resolutions can increase the accuracy of the mobile robot's navigations. For example, the signal counts for a 45degree turn is supposedly 7.5 and 90degree is actually 15. However, due to low resolution, the condition was set to be 8 counts for 45degree turn. Even though it is quite accurate, for 45degree turn and 90degree turn, as the angle of turning increase, the inaccuracy will become more apparent. If the rotary encoder's resolution is increased by 100%, the condition for a turn of 45degree can be set to 15 and 90degree to 30 and etc.

8.1.2 Replacing Keypad Wires with Wireless Connections

Improvements can also be made that the mobile robot can receive instructions from the keypad via wireless. It solves the matter of messy connections altogether and it also makes the end product to be more attractive and gives a more high technology packaging to the consumers.

8.1.3 Sensor to Avoid Obstacle

A sensor can be installed in front of the mobile robot as a safety measure. This sensor will act as indicator of obstacle approaching so the mobile robot will stop advancing even if it was programmed to do so. This will help to ensure mobile robot does not damage from repetitive crush into walls or obstacles.

8.2 Conclusion

This project is developed with the hope to provide an alternative to the toy industry, as a more challenging mobile toy to the young children where children can exercise their brain when they play. This project has completed the basis needed for a mobile robot such as this to function. It can prompt for input, it can receive input from users, it reads and retrieves instructions or data stored and then execute it to perform a task or to reach a destination.

However, for the mobile robot to be more marketable, it still needs other enhancements and improvements such as the recommendations mentioned in the previous page. Besides those, outlook of the mobile robot is also very important. That the circuit should be hidden, the casing should look attractive, the keypad should look modern and etc.

For the mobile robot to look more attractive, more technologies should be instilled. For example, the LCD display should be replaced with a colored LCD screen that is thin and sleek and there should be lightings or sounds as the mobile robot executes its command.

REFERENCES

1. <http://www.robots.com/glossary.php>
2. http://en.wikipedia.org/wiki/Mobile_robot
3. <http://homepages.which.net/~paul.hills/SpeedControl/SpeedControllersBody.html>
4. http://www2.renesas.com/motor_driver/en/motor_drv_info.html
5. ST L298 motor driver datasheet,
<http://www.st.com/stonline/books/pdf/docs/1773.pdf>

APPENDIX A

FULL FIRMWARE

```
1 #include <18F452.h>
2 #fuses XT,NOWDT,NOPROTECT,NOLVP,NOPUT,NOBROWNOUT,NODEBUG
3 #use delay(clock = 4000000)
4
5 #include "C:\Documents and Settings\User\My Documents\Past
6 semesters\Microcontroller\lab4\Flexlcd2.c"
7
8 //pin define for keypad
9 #define col1 PIN_D0
10 #define col2 PIN_D1
11 #define col3 PIN_D2
12 #define col4 PIN_D3
13 #define row1 PIN_C4
14 #define row2 PIN_C5
15 #define row3 PIN_C6
16 #define row4 PIN_C7
17
18 #define in1 PIN_B1 //set B6 to be output1
19 #define in2 PIN_B2 //set B7 to be output2
20 #define in3 PIN_B4 //set B4 to be output3
21 #define in4 PIN_B5 //set B5 to be output4
22
23 #define encoder1 PIN_B0
24 #define encoder2 PIN_B3
25 #define max 60000
26
27 #define sensor PIN_C3
28
29 int key, keyEn1, keyEn2, step, g, h, i, j, k, l, m;
30 int store_instr[30], store_step[30];
31
32 void keypad(void);
33 void keypad_lcd(void);
34 void motor_lcd(void);
35 void row_scan(void);
36 void encoder1_lcd(void);
37 void encoder2_lcd(void);
38
39 void encoder(void);
40 void reverse(void);
41 void turn_left(void);
42 void turn_right(void);
43 void slight_left(void);
44 void slight_right(void);
45 void forward(void);
46 void stop(void);
47
48 int16 count1, count2, f;
```

```

49 void main(void){
50
51  set_tris_a(0x00);
52  set_tris_c(0x08);
53  set_tris_d(0x0F);
54  set_tris_b(0xC9);
55  set_tris_e(0x0);
56
57  setup_timer_2(T2_DIV_BY_1,255,1); //postscaler=1, prescaler=1, PR = 255
58  enable_interrupts(INT_TIMER2); //enable interrupt for Timer2
59  enable_interrupts(global); //enabling interrupts
60
61  setup_ccp1(CCP_PWM); //set PIN_C2 to be in PWM mode
62  set_pwm1_duty(0);
63
64  setup_ccp2(CCP_PWM); //set PIN_C1 to be in PWM mode
65  set_pwm2_duty(0);
66
67  lcd_init();
68
69  for (k=0;k<200;k++)
70  {
71  for (l=0;l<3;l++)
72  {
73  lcd_gotoxy(1,1);
74  lcd_putc("Pls enter instr");
75  lcd_gotoxy(1,2);
76  lcd_putc("& steps desired");
77  }
78  }
79  lcd_gotoxy(1,2);
80  lcd_putc("\n");
81
82  while(1)
83  {
84
85  stop();
86  lcd_gotoxy(1,1);
87  lcd_putc("instr:");
88  lcd_gotoxy(1,2);
89  lcd_putc("steps:");
90
91
92  h = 0;
93  i = 0;
94  key = 0;
95  while(key != 13)
96  {
97  key = 0;
98  while (key == 0)
99  {
100  keypad();
101  }
102  if (key == 10 || key == 11 || key == 12 || key == 13 || key == 14 || key == 15 ||
103  key == 16)

```

```

104     {
105         store_instr[h]=key;
106         h++;
107     }
108     else
109     {
110         store_step[i]= key;
111         i++;
112     }
113     delay_ms(500);
114     keypad_lcd();
115     delay_ms(1500);
116 }
117
118 lcd_gotoxy(1,2);
119 lcd_putc("\f");
120
121 key = h - 1;
122 keypad_lcd();
123 delay_ms(1000);
124
125 lcd_gotoxy(1,1);
126 lcd_putc("Execute instr");
127 lcd_gotoxy(1,2);
128 lcd_putc("from the start");
129 delay_ms(3000);
130
131 lcd_gotoxy(1,2);
132 lcd_putc("\f");
133
134 lcd_gotoxy(1,1);
135 lcd_putc("instr:");
136 lcd_gotoxy(1,2);
137 lcd_putc("steps:");
138
139     count1 = 0;
140     count2 = 0;
141
142     for (j=0; j<h; j++)
143     {
144         key = store_instr[j];
145         motor_lcd();
146         step = store_step[j];
147
148         if (key == 10)
149         {
150             keypad_lcd();
151             forward();
152
153             for(g=1;g<step+1;g++)
154             {
155                 do
156                 {
157                     if (input(sensor))
158                         break;
159                     if(input(encoder1))

```

```

160     {
161     while(input(encoder1));
162     count1 = count1 + 1;
163     keyEn1 = count1;
164     encoder1_lcd();
165     }
166     if(input(encoder2))
167     {
168     while(input(encoder2));
169     count2 = count2 + 1;
170     keyEn2 = count2;
171     encoder2_lcd();
172     }
173
174     if (count1 > count2)
175     {
176     slight_right();
177     delay_ms(5);
178     }
179
180     else if (count2 > count1)
181     {
182     slight_left();
183     delay_ms(5);
184     }
185
186     else
187     {
188     forward();
189     }
190     }
191     while((count1 < 9) && (count2 < 9));
192
193     if(count1 < count2)
194     {
195     count1 = 1;
196     count2 = 0;
197     keyEn1 = count1;
198     encoder1_lcd();
199     keyEn2 = count2;
200     encoder2_lcd();
201     }
202
203     else if(count2 < count1)
204     {
205     count1 = 0;
206     count2 = 1;
207     keyEn1 = count1;
208     encoder1_lcd();
209     keyEn2 = count2;
210     encoder2_lcd();
211     }
212
213     else if(count1 == count2)
214     {
215     count1 = 0;

```



```

216     count2 = 0;
217     keyEn1 = count1;
218     encoder1_lcd();
219     keyEn2 = count2;
220     encoder2_lcd();
221     }
222
223     key = g;
224     keypad_lcd();
225     }
226     }
227
228     else if(key == 12 || key == 16 || key == 11)
229     {
230         count1 = 0;
231         count2 = 0;
232         motor_lcd();
233         step = store_step[j];
234     }
235
236     if(key == 11)
237     {
238         for(g=1;g<step+1;g++)
239         {
240             key = g;
241             keypad_lcd();
242             delay_ms(700);
243         }
244     }
245
246     if (key == 16)
247     {
248         for(g=1;g<step+1;g++)
249         {
250             do
251             {
252                 if(input(encoder2))
253                 {
254                     while(input(encoder2));
255                     count2 = count2 + 1;
256                 }
257                 keyEn2 = count2;
258                 encoder2_lcd();
259             }
260             while(count2 < 9);
261             count2 = 0;
262         }
263     }
264     key = store_instr[j];
265
266     if (key == 12)
267     {
268         for(g=1;g<step+1;g++)
269         {
270             do
271             {

```

```

272     if(input(encoder1))
273     {
274         while(input(encoder1));
275         count1 = count1 + 1;
276     }
277     keyEn1 = count1;
278     encoder1_lcd();
279     }
280     while(count1 < 8);
281     count1 = 0;
282     }
283     }
284     key = store_instr[j];
285
286     stop();
287     delay_ms(1000);
288     }
289     lcd_gotoxy(1,2);
290     lcd_putc("\f");
291
292     lcd_gotoxy(1,1);
293     lcd_putc("Pls key in new");
294     lcd_gotoxy(1,2);
295     lcd_putc("instr & steps");
296     delay_ms(3000);
297
298     lcd_gotoxy(1,2);
299     lcd_putc("\f");
300 }
401
402}
403
404 void keypad(void)
405 {
406     m = 0;
407     for(k=0; k<4; k++)
408     {
409         row_scan();
410
411         if(k == 0)
412         {
413             if (input(col1))
414             {
415                 key = 1;
416             }
417
418             if (input(col2))
419             {
420                 key = 2;
421             }
422
423             if (input(col3))
424             {
425                 key = 3;
426             }
427

```

```

428     if (input(col4))
429     {
430         key = 10;
431     }
432
433     delay_ms(10);
434     output_low(row1);
435 }
436
437 if (k == 1)
438 {
439     if (input(col1))
440     {
441         key = 4;
442     }
443
444     if (input(col2))
445     {
446         key = 5;
447     }
448
449     if (input(col3))
450     {
451         key = 6;
452     }
453
454     if (input(col4))
455     {
456         key = 11;
457     }
458     delay_ms(10);
459     output_low(row2);
460 }
461
462 if (k == 2)
463 {
464     if (input(col1))
465     {
466         key = 7;
467     }
468
469     if (input(col2))
470     {
471         key = 8;
472     }
473
474     if (input(col3))
475     {
476         key = 9;
477     }
478
479     if (input(col4))
480     {
481         key = 12;
482     }
483     delay_ms(10);

```

```

484 output_low(row3);
485 }
486
487
488 if (k == 3)
489 {
490 if (input(col1))
491 {
492 key = 13;
493 }
494
495 if (input(col2))
496 {
497 key = 14;
498 }
499
500 if (input(col3))
501 {
502 key = 15;
503 }
504
505 if (input(col4))
506 {
507 key = 16;
508 }
509 delay_ms(10);
510 output_low(row4);
511
512 }
513 }
514
515 }
516
517 void row_scan(void)
518 {
519 switch(k)
520 {
521 case 0: output_high(row1); break;
522 case 1: output_high(row2); break;
523 case 2: output_high(row3); break;
524 case 3: output_high(row4); break;
525 }
526 }
527
528 void keypad_lcd(void)
529 {
530 switch(key)
531 {
532 case 0:
533 {
534 break;
535 }
536 case 1:
537 { //1
538 lcd_gotoxy(7,2);
539 lcd_putc("1");

```

```

540     break;
541     }
542 case 2:
543     { //2
544         lcd_gotoxy(7,2);
545         lcd_putc("2");
546         break;
547     }
548 case 3:
549     { //3
550         lcd_gotoxy(7,2);
551         lcd_putc("3");
552         break;
553     }
554
555 case 10:
556     { //F
557         lcd_gotoxy(7,1);
558         lcd_putc(" ");
559         lcd_gotoxy(7,1);
560         lcd_putc("forward");
561         lcd_gotoxy(7,2);
562         lcd_putc(" ");
563         break;
564     }
565 case 4:
566     { //4
567         lcd_gotoxy(7,2);
568         lcd_putc("4");
569         break;
570     }
571 case 5:
572     { //5
573         lcd_gotoxy(7,2);
574         lcd_putc("5");
575         break;
576     }
577 case 6:
578     { //6
579         lcd_gotoxy(7,2);
580         lcd_putc("6");
581         break;
582     }
583 case 11:
584     { //E
585         lcd_gotoxy(7,1);
586         lcd_putc(" ");
587         lcd_gotoxy(7,1);
588         lcd_putc("reverse");
589         lcd_gotoxy(7,2);
590         lcd_putc(" ");
591         break;
592     }
593 case 7:
594     { //7
595         lcd_gotoxy(7,2);

```

```

606     lcd_putc("7");
607     break;
608 }
609 case 8:
610     { //8
621         lcd_gotoxy(7,2);
622         lcd_putc("8");
623         break;
624     }
625 case 9:
626     { //9
627         lcd_gotoxy(7,2);
628         lcd_putc("9");
629         break;
630     }
631 case 12:
632     { //D
633         lcd_gotoxy(7,1);
634         lcd_putc(" ");
635         lcd_gotoxy(7,1);
636         lcd_putc("left");
637         lcd_gotoxy(7,2);
638         lcd_putc(" ");
639         break;
640     }
641 case 13:
642     { //A
643         lcd_gotoxy(7,1);
644         lcd_putc(" ");
645         lcd_gotoxy(7,1);
646         lcd_putc("A");
647         lcd_gotoxy(7,2);
648         lcd_putc(" ");
649         break;
650     }
651 case 14:
652     { //0
653         lcd_gotoxy(7,2);
654         lcd_putc("0");
655         break;
656     }
657 case 15:
658     { //B
659         lcd_gotoxy(7,2);
660         lcd_putc("B");
661         break;
662     }
663 case 16:
664     { //C
665         lcd_gotoxy(7,1);
666         lcd_putc(" ");
667         lcd_gotoxy(7,1);
668         lcd_putc("right");
669         lcd_gotoxy(7,2);
670         lcd_putc(" ");
671         break;

```

```

672     }
673 }
674}
675
676 void motor_lcd(void)
677 {
678     switch(key)
679     {
680         case 11:
681             { //E
682                 lcd_gotoxy(7,1);
683                 lcd_putc(" ");
684                 lcd_gotoxy(7,1);
685                 lcd_putc("reverse");
686                 lcd_gotoxy(7,2);
687                 lcd_putc(" ");
688                 reverse();
689                 delay_ms(10);
690                 break;
691             }
692
693         case 12:
694             { //D
695                 lcd_gotoxy(7,1);
696                 lcd_putc(" ");
697                 lcd_gotoxy(7,1);
698                 lcd_putc("left");
699                 lcd_gotoxy(7,2);
700                 lcd_putc(" ");
701                 turn_left();
702                 delay_ms(10);
703                 break;
704             }
705         case 13:
706             { //A
707                 lcd_gotoxy(7,1);
708                 lcd_putc(" ");
709                 lcd_gotoxy(7,1);
710                 lcd_putc("A");
711                 break;
712             }
713         case 14:
714             { //0
715                 lcd_gotoxy(7,2);
716                 lcd_putc("0");
717                 break;
718             }
719         case 15:
720             { //B
721                 lcd_gotoxy(7,2);
722                 lcd_putc("B");
723                 break;
724             }
725         case 16:
726             { //C
727                 lcd_gotoxy(7,1);

```

```

738     lcd_putc(" ");
739     lcd_gotoxy(7,1);
740     lcd_putc("right");
741     lcd_gotoxy(7,2);
742     lcd_putc(" ");
743     turn_right();
744     delay_ms(10);
745     break;
746 }
747 }
748}
749
750
751 void forward(void)
752 {
753
754     set_pwm2_duty(115);
755     set_pwm1_duty(115);
756     output_high(in1);
757     output_low(in2);
758     output_high(in3);
759     output_low(in4);
760}
761
762 void reverse(void)
763 {
764     set_pwm2_duty(85);
765     set_pwm1_duty(85);
766     output_low(in1);
767     output_high(in2);
768     output_low(in3);
769     output_high(in4);
770}
771
772 void turn_right(void)
773 {
774     set_pwm2_duty(95);
775     set_pwm1_duty(0);
776     output_high(in1);
777     output_low(in2);
778     output_high(in3);
779     output_low(in4);
780}
781
782 void slight_right(void)
783 {
784     set_pwm2_duty(85);
785     set_pwm1_duty(110);
786     output_high(in1);
787     output_low(in2);
788     output_high(in3);
789     output_low(in4);
790 }
791
792 void turn_left(void)
793 {

```



```

794 set_pwm2_duty(0);
795 set_pwm1_duty(95);
796 output_high(in1);
797 output_low(in2);
798 output_high(in3);
799 output_low(in4);
800 }
811
812 void slight_left(void)
813
814 set_pwm2_duty(110);
815 set_pwm1_duty(80);
816 output_high(in1);
817 output_low(in2);
818 output_high(in3);
819 output_low(in4);
820}
821
822void stop(void)
823{
824 set_pwm2_duty(0);
825 set_pwm1_duty(0);
826 output_high(in1);
827 output_high(in2);
828 output_high(in3);
829 output_high(in4);
830}
831
832void encoder(void)
833{
834     keypad_lcd();
835     forward();
836
837     for(g=1;g<step+1;g++)
838     {
839     do
840     {
841     if (input(sensor))
842     break;
843     if(input(encoder1))
844     {
845     while(input(encoder1));
846     count1 = count1 + 1;
847     keyEn1 = count1;
848     encoder1_lcd();
849     }
850     if(input(encoder2))
851     {
852     while(input(encoder2));
853     count2 = count2 + 1;
854     keyEn2 = count2;
855     encoder2_lcd();
856     }
857
858     if (count1 > count2)
859     {

```

```

860     slight_right();
861     delay_ms(5);
862     }
863
864     else if (count2 > count1)
865     {
866     slight_left();
867     delay_ms(5);
868     }
869
870     else
871     {
872     forward();
873     }
874 }
875 while((count1 < 9) && (count2 < 9));
876
877 if(count1 < count2)
878 {
879 count1 = 1;
880 count2 = 0;
881 keyEn1 = count1;
882 encoder1_lcd();
883 keyEn2 = count2;
884 encoder2_lcd();
885 }
886
887 else if(count2 < count1)
888 {
889 count1 = 0;
890 count2 = 1;
891 keyEn1 = count1;
892 encoder1_lcd();
893 keyEn2 = count2;
894 encoder2_lcd();
895 }
896
897 else if(count1 == count2)
898 {
899 count1 = 0;
900 count2 = 0;
901 keyEn1 = count1;
902 encoder1_lcd();
903 keyEn2 = count2;
904 encoder2_lcd();
905 }
906
907 key = g;
908 keypad_lcd();
909 }
910}
911
912 void encoder1_lcd(void)
913 {
914     switch(keyEn1)
915     {

```

```

916 case 0:
917     {
918         break;
919     }
920 case 1:
921     { //1
922         lcd_gotoxy(10,2);
923         lcd_putc("1");
924         break;
925     }
926 case 2:
927     { //2
928         lcd_gotoxy(10,2);
929         lcd_putc("2");
930         break;
931     }
932 case 3:
933     { //3
934         lcd_gotoxy(10,2);
935         lcd_putc("3");
936         break;
937     }
938
939 case 4:
940     { //4
941         lcd_gotoxy(10,2);
942         lcd_putc("4");
943         break;
944     }
945 case 5:
946     { //5
947         lcd_gotoxy(10,2);
948         lcd_putc("5");
949         break;
950     }
951 case 6:
952     { //6
953         lcd_gotoxy(10,2);
954         lcd_putc("6");
955         break;
956     }
957
958 case 7:
959     { //7
960         lcd_gotoxy(10,2);
961         lcd_putc("7");
962         break;
963     }
964 case 8:
965     { //8
966         lcd_gotoxy(10,2);
967         lcd_putc("8");
968         break;
969     }
970 case 9:
971     { //9

```

```

972     lcd_gotoxy(10,2);
973     lcd_putc("9");
974     break;
975     }
976 }
977}
978
979void encoder2_lcd(void)
980{
981  switch(keyEn2)
982  {
983   case 0:
984     {
985       break;
986     }
987   case 1:
988     { //1
989       lcd_gotoxy(13,2);
990       lcd_putc("1");
991       break;
992     }
993   case 2:
994     { //2
995       lcd_gotoxy(13,2);
996       lcd_putc("2");
997       break;
998     }
999   case 3:
1000    { //3
1001      lcd_gotoxy(13,2);
1002      lcd_putc("3");
1003      break;
1004    }
1005
1006   case 4:
1007    { //4
1008      lcd_gotoxy(13,2);
1009      lcd_putc("4");
1010      break;
1011    }
1012   case 5:
1013    { //5
1014      lcd_gotoxy(13,2);
1015      lcd_putc("5");
1016      break;
1017    }
1018   case 6:
1019    { //6
1020      lcd_gotoxy(13,2);
1021      lcd_putc("6");
1022      break;
1023    }
1024
1025   case 7:
1026    { //7
1027      lcd_gotoxy(13,2);

```

```
1028     lcd_putc("7");
1029     break;
1020 }
1031 case 8:
1032     { //8
1033         lcd_gotoxy(13,2);
1034         lcd_putc("8");
1035         break;
1036     }
1037 case 9:
1038     { //9
1039         lcd_gotoxy(13,2);
1040         lcd_putc("9");
1041         break;
1042     }
1043 }
1044 }
```

Task\Week	1	2	3	4	5	6	7	Mid-semester Break							8	9	10	11	12	13	14
Meet with Supervisor and discuss about topic	█																				
Research and Literature review on suitable parts • Determine electronic parts for projects		█	█																		
Design project circuits			█	█																	
Preliminary Report				█																	
Solder and assemble circuit Assemble prototype					█	█	█														
Test and debug circuit							█														
Progress Report																					
Project Seminar																					
Develop firmware flowchart																					
Develop DC motor Firmware • Firmware tested on hardware																	█	█	█		
Interim Report																					
Oral Presentation																					█

Task\Week	1	2	3	4	5	6	7	Mid-semester Break							8	9	10	11	12	13	14
Rotary encoder attachment	█																				
Firmware development for reading signals from rotary encoder	█	█																			
Firmware testing and calibration for rotary encoder		█	█	█	█	█	█														
Schematic Design for User Interface Devices			█																		
Hardware assembly for updated schematic				█																	
Firmware development for new hardware parts • Keypad • LCD display				█	█	█	█														
Firmware testing and debugging							█	█	█												
Progress Report							█	█													
Combine all parts of firmware for whole project																					
Firmware testing and debugging																	█	█	█	█	█
Final Draft Report																					
Prototype testing																					
Final Report																					
Technical Report																					█