Department of Electronic and Telecommunication Engineering

University of Moratuwa

EN1093 – Laboratory Practice I

Project Report

Line Following and Obstacle Avoiding Robot

Name	Index No.
Athukorala A.U.P.H.	180051F
Bandara P.M.N.S.	180066F
Chandanayake S.M.	180085L
Yalegama M.M.K.A.B.	180720G

Supervisor: Ms. Sakila Jayaweera

This is submitted as a partial fulfillment for the module EN1093: Laboratory Practice I
Department of Electronic and Telecommunication Engineering,
University of Moratuwa
March 6, 2020

Contents

Abstract

In the modern world, Robotics could be vastly identified as a sustained modern scientific field in which interdisciplinary braches of Engineering and Science collaborates together for a one or several defined mission(s) and/or task(s). Furthermore, the emerging technologies of the current world such as Machine Learning and Artificial Intelligence have boosted the way of Robotics in a wide range of manner. Thus, it is evident to state that the fundamental knowledge on Robotics is an essential for an Engineer for sustaining in the current technological phenomenon.

In the following project, we have implemented a relatively simple Robot Project for EN1093: A Line Following and Obstacle Avoiding Robot. The robot is expected to follow a given white path on a black arena, avoiding any obstacle on that specific path. The given arena is basically consisted of a 3cm width single path including slight curves and 90 degree turns and further, at several points with 5cm diameter cylindrical obstacles.

The designed robot of our team used Atmega32A microcontroller and Atmel Studio 7 to program the robot. Orcad Capture CIS, Orcad Layout and Altium Designer Software were utilized for circuit design, simulation and PCB layout design purposes. The main hardware of the robot crucially consists of the main circuit (microcontroller circuit), motor controller circuit, IR sensor array circuit, sensor indication circuit and the power supply circuit.

Maneuvering the robot on the path by getting the inputs from IR sensor array on white line on the black surface, while detecting and avoiding any obstacle on that path requires simple but efficient closed loop functionality. In this project, we have sufficiently attempted to achieve that specific task in a collaborative manner.

Introduction

Robotics is a developing and useful field in the present field. For this project we use micro controllers which are like small computers. These microcontrollers play a vital role in robotics. For this project a simple robot is designed using microcontroller to navigate through a line follow arena. Then detect a small obstacle and avoid it.

Arena Specifications

- Width of the track = 30 mm
- There are two 90 degree bends and one arched path
- The obstacle is cylindrical with 50mm diameter

Robot Specifications

- Only one robot is allowed per participating team.
- Maximum dimensions of the robot are 250mm x 250mm x 250mm.
- No weight requirement for the robot.
- The robot must be battery powered.
- The robot must follow line order to navigate on the arena

Scientific literature

When we started our project, we had not done any robotics or any programming with microcontrollers. First we divided our PCBs (main PCB, IR sensor panel, voltage regulator, motor controller) amongst ourselves. Afterwards the PCBs were decided, then made and the components mounted on the board. Then while two members were mounting the hardware of the robot were being mounted on the robot and were being trouble-shooted other two members looked at the coding part. Then the final troubleshooting was done together.

Microcontroller

The decided micro controller is atmega32A because of its wide selection of pins.

Line follow sensor

The sensor is capable of sensing the color of the line under the sensor (black or white) according voltage out of the TCRT5000.

Motor Controller

The motor controller was based on the L298N IC. Speed of the gear motors driving the robot can be controlled using PWM pins of the motor controller

Main circuit

This mainly includes the atmega 32A micro controller and all the headers for the output of sensor panel and ultrasonic sensors. Atmega32A micro controller takes commands from sensor panel and ultrasonic sensors and control the motor accordingly. Hence the micro controller must be programmed in such a way that robot follows white line in a black surface and avoids obstacles before placing on the main circuit. We used atmel studio 7 to program, used port A for the sensor panel and Pin B3, Pin D7 are the PWM pins to control the speed of the motor (To enable the motor circuit). To give inputs to the motors, we used Pin C0, C1, C6 and C7 Pins. When we consider about the obstacle avoiding part, our idea was to connect the three ultrasonic sensors to the port B.

Sensor panel

Sensor panel consists of 5 TCRT 5000 sensors with a photo transistor and an IR emitter. All the sensors are connected to the one variable resistor to set a threshold to compare the outputs of the sensor. To compare the voltage levels of the photo transistor, LM324 comparators are used. All the sensors are placed with 1cm. Middle 3 sensors are to identify the white line and other sensors are to identify the black surface. So, if the robot goes out of the track, sensor outputs change accordingly. Therefore, we were to identify different outputs for different instances to keep the robot in the right track.

Motor Controller and Power Circuit

The Motor Controller Circuit consist the following components.

1. L298N motor controller IC
   Main Integrated Circuit (IC) used for the purpose of controlling the speed of motors and for the easiness of using a heat sink.
2. Female headers
   Used to connect the jumpers to the microcontroller and to the motors
3. Jumpers and circuit wires

   Used to connect several components between different PCBs.
4. Connectors

   Used as a fixing method of circuit wires to the PCBs.
5. Diodes

   To avoid the negative electro motive power of motors
6. 220uF Capacitor

We have implemented LM2596-Adj and LM7805 in the power circuit for getting sufficient amount of accuracy and easy trouble shooting purposes as we have identified using datasheets.

Conceptual Approach to the target

The Project was fundamentally approached in two manners in which the structural flow could be conveyed as follows.

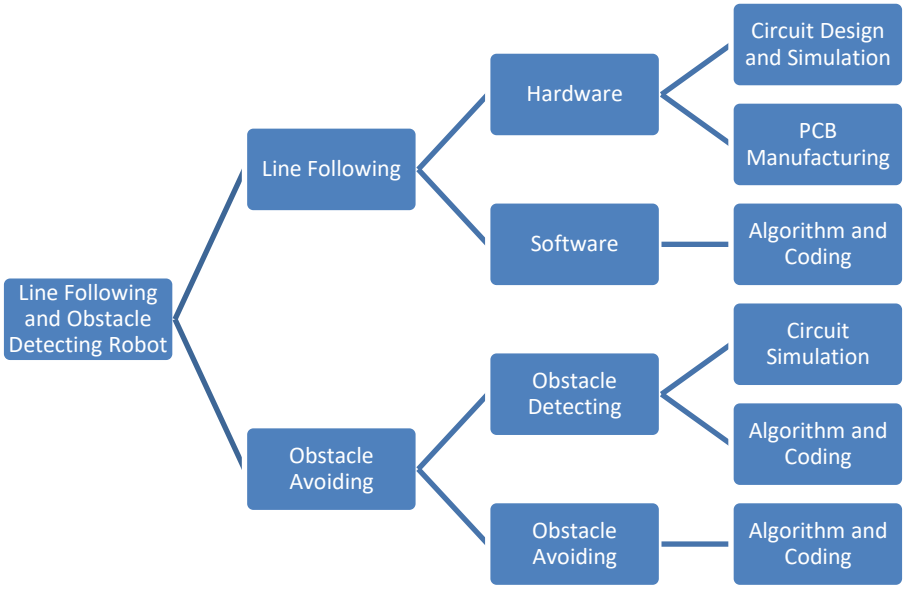
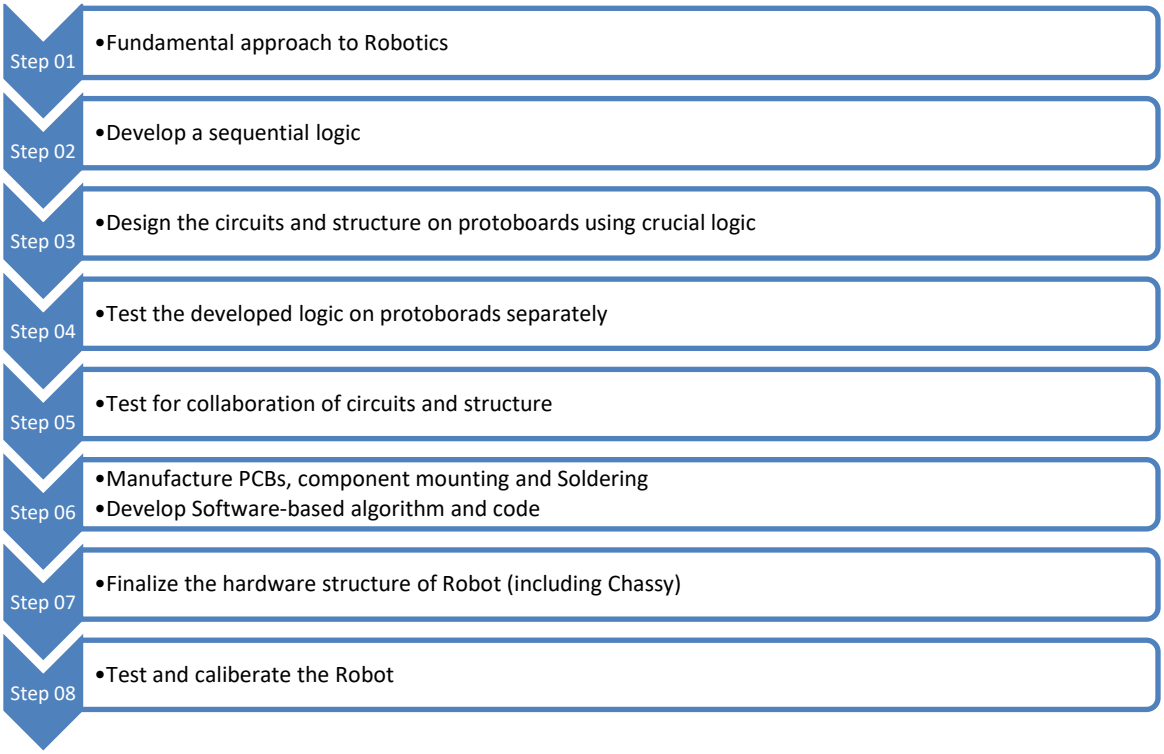


Figure 02 – Conceptual Approach

Main Steps of the Project



Applied Logic

The below chart sequentially depict the logic that was applied to achieve the given task via developed C code.

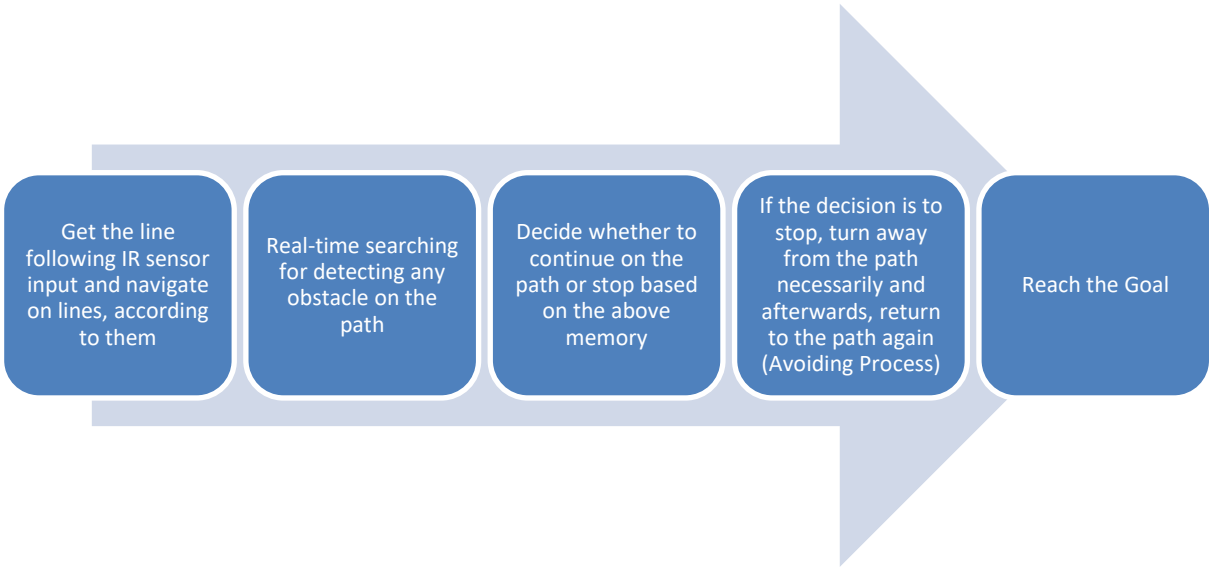


Figure 03 – Applied Logic

Main Parts of the Code

The below parts could be obviously considered as the main parts of the developed C code, manipulated based on the above applied logic.

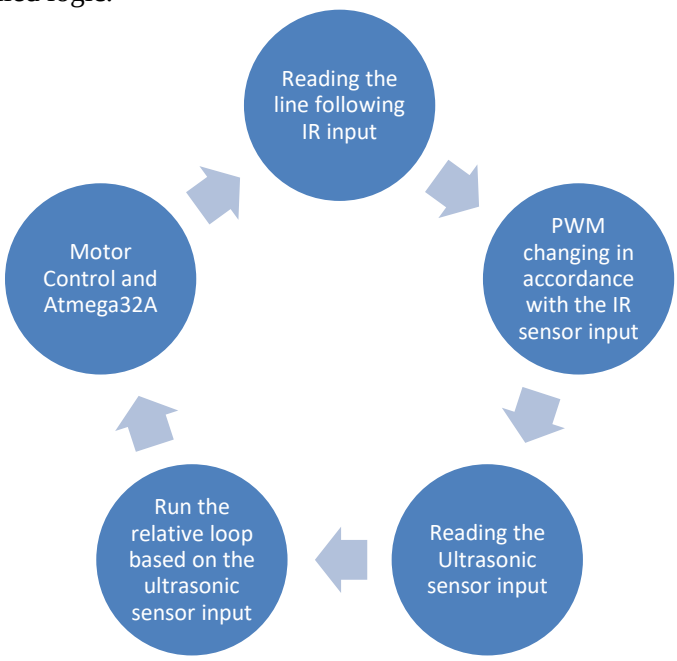


Figure 04 – Main Parts of the Code

Algorithm

The following flow chart for the algorithm is developed by implementing the above logic (prior to coding) using SmartDraw 2019.

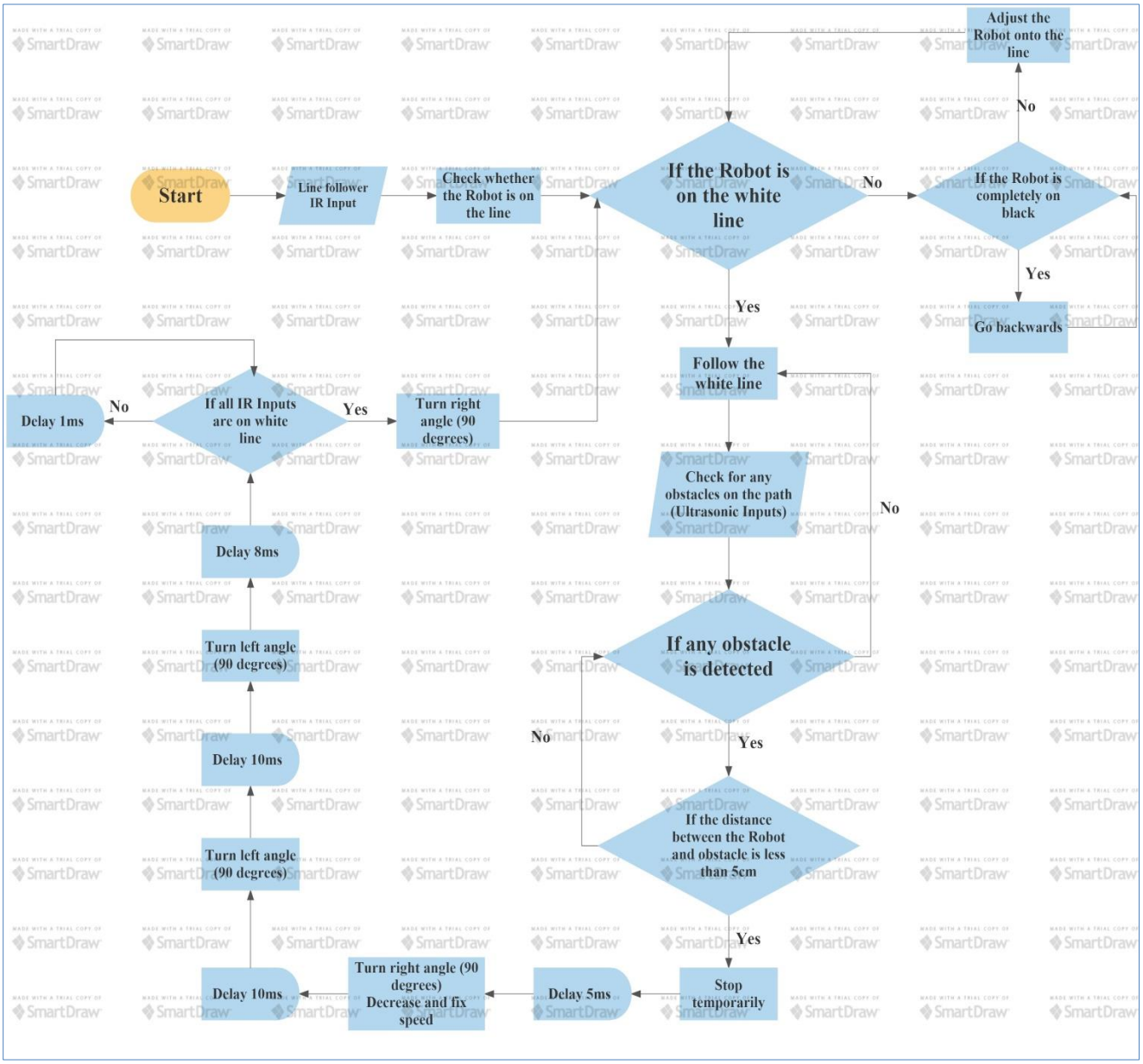


Figure 05 – Algorithm in a flow chart (manipulated using SmartDraw 2019 Software)

RESULTS

- The test runs are done using **a black arena with a white line designed based on given specifications.**
    - The arena is used to calibrate the robot and its sensors to first follow a white track on a black arena and to calculate the total times taken to complete the task as well as to test whether the logic required identifying the correct path.

Success of following the white line

The robot is first calibrated using the arena described above to follow a white line on the black arena.

Table: Test Results of Following a straight white line

Trial number	Successful/ Unsuccessful
1	Unsuccessful
2	Unsuccessful
3	Unsuccessful
4	Successful
5	Successful
6	Unsuccessful
7	Unsuccessful
8	Successful

Average successions of following a straight white line = **50%**

Success of avoiding the obstacle

The robot is then tested for detecting the obstacle correctly and extinguishing the correct path again.

Table: Test Results of detecting the correct path at the Junction

Trial number	**Successful/ Unsuccessful**
1	Unsuccessful
2	Unsuccessful
3	Unsuccessful
4	Unsuccessful
5	Unsuccessful
6	Unsuccessful
7	successful
8	Successful

Average successions of following a straight white line = **25%**

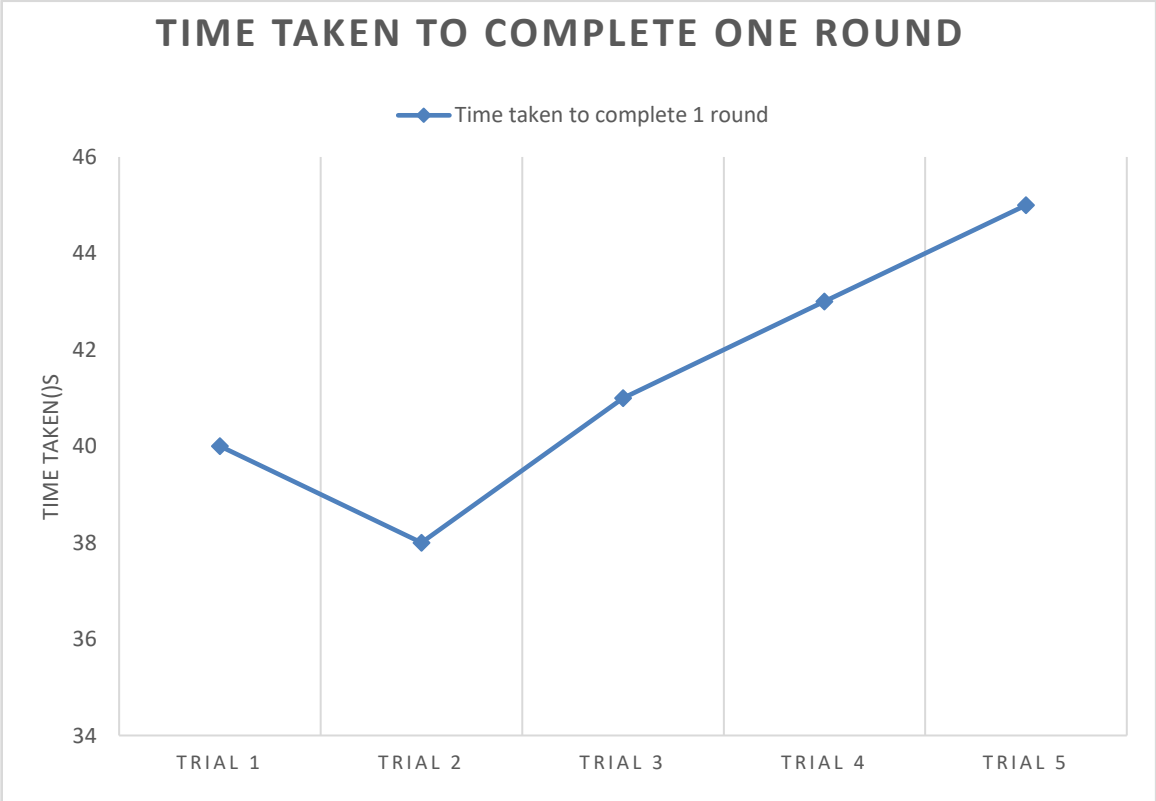
Average time taken to complete the task

Time was measured for one round from the starting point to ending pointing disregarding whether the robot didn't follow the line

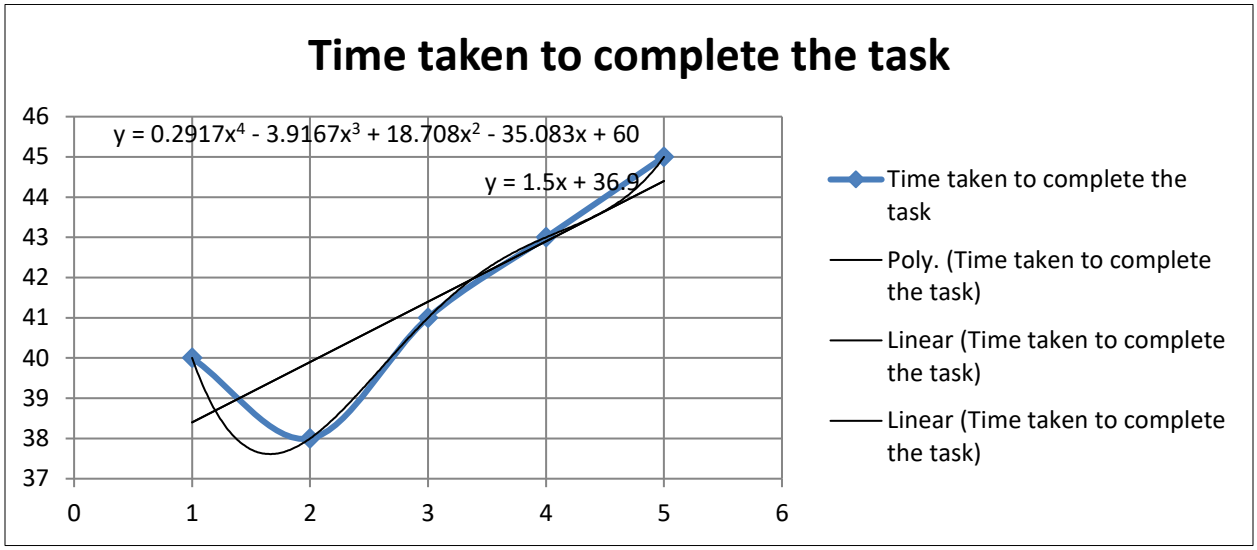
Table: Average time taken to complete the task

Trial number	Time taken (seconds)
1	40
2	38
3	41
4	43
5	45

Average time taken = **41.4**



If we statistically the analyze the above graph,

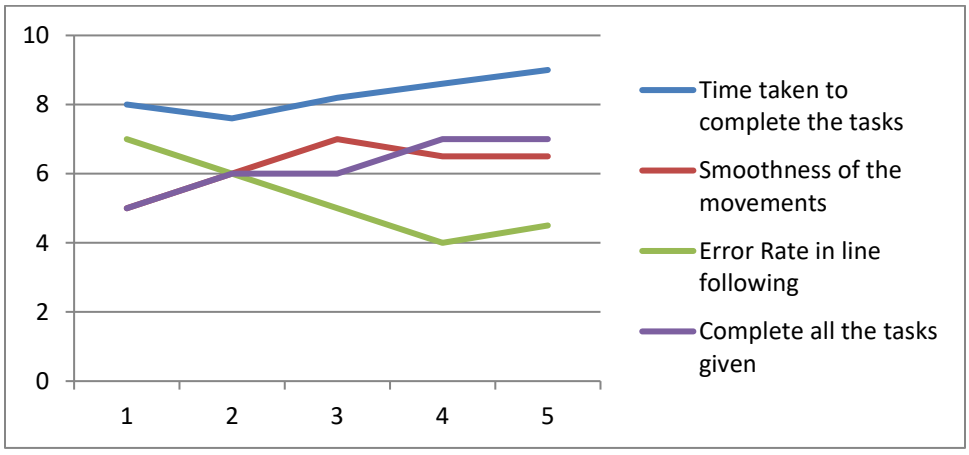


It could be seen that the time per one trial has been slightly increased as per we have implemented more complex code blocks for the Robot to achieve its full work with optimization. The above trend-lines will be stabilized as we have observed under more fine tuning process.

The qualitative performance matrix of the robot should essentially include,

- The time taken to complete the task
- The smoothness of the movements
- Less error rate in line following
- Complete all the tasks given

For quantitatively depict these parameters, we have gained several observations through our trials,



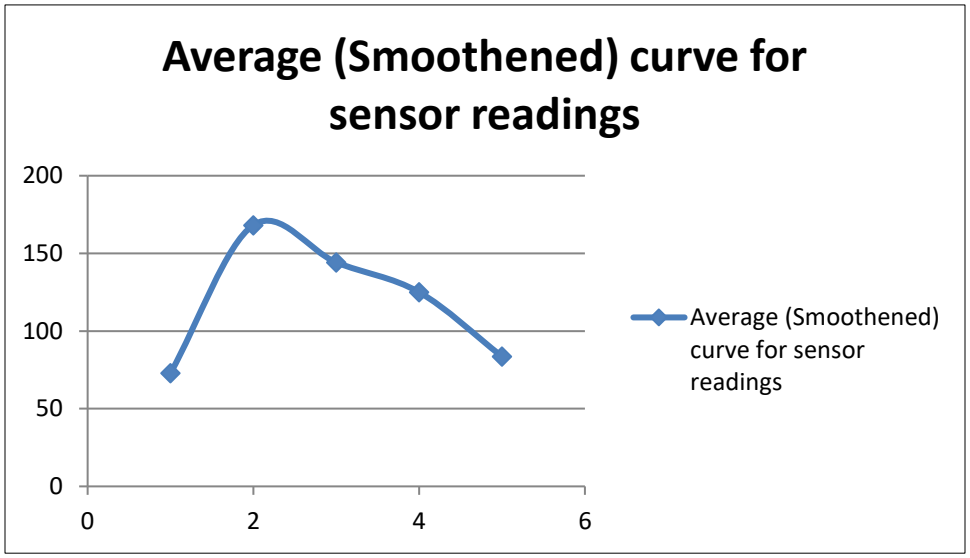
(Here, we have implemented a qualitative 0 to 10 scale for evaluation)

For the line following task, we have thoroughly observed that the steady IR sensor readings are heavily needed, so that, further, we have analyzed the sensor readings for better clarification.

In the configuration of the robot is on the path such that three sensors sense white and two sensors sense black, (ADC values are taken)

Trial	Sensor 01	Sensor 02	Sensor 03	Sensor 04	Sensor 05
1	61	176	144	130	70
2	83	183	160	150	94
3	71	153	133	111	81
4	72	158	134	125	83
5	78	171	150	109	90

The above data shows us quantitatively that there is a significant ADC value difference between each situation with possible confidence interval.



By considering the above analysis, it is obvious to say that the performance matrix of a robot is complex to analyze, as there are more undetermined parameters and/or variables, so that what we can achieve is probability density functions and observation data assuming that all-not considered facts are constant

DISCUSSION

- As beginners for the robot designing, we had to face many problems while doing this project. We learnt everything from basics such as:
  (Using Atmel programming, Coding using C programming and hardware designing)

    - **Sense-Plan-Act architecture** (SPA) has been used to design this project and the vision system is sensor based. Six (6) (separate) Printed Circuit Boards (PCB) were designed for the motor controller circuit, main circuit, sensor panel, for LED indicating and for the buck converter and voltage regulator in power supply circuit for the easiness.

The problems we faced while carrying on the project are listed below.

1. When the robot was tested for its functionality using Bread boards, we ran into issues with burned ICs which we used in different circuits.
2. Some of the jumper cables we used in circuits were not properly working.
3. As the voltage dropped in the batteries drastically during the test runs hence the motors didn't move.
4. After the manufacturing of PCBs and soldering the components, there were several issues we faced.
    - The sensor panel was not working.
    - The main circuit was not properly working.
5. After the PCBs were assembled, the code did not work as we expected. Hence we had to drop certain plans like using Proportional-Integration-Differential (PID) Control System along with PWM to navigate and control the robot.
6. When simulating the code on Atmel, we had issues with Pulse Width Modulation(PWM) not working properly hence we had to alter the code several times.
7. The torque of the DC motors we used was high hence the speed variations cannot be done in a vast range of PWM.
8. The wires on the PCBs were messy and it was hard to troubleshoot errors.
9. During the test runs, the robot didn't follow even a straight line.
10. During the test runs, the robot didn't avoid any obstacle.

The causes we have identified for these problems are listed below

1. The ICs had been burned due to the heat generated in the circuits. So we had to replace them with new ones.
2. We identified that most of the circuit problems were occurred due to the connection errors in jumper wires. So we replaced them with circuit wires.
3. The diodes on the motor controller circuit had been burned due to the high voltage we used and that was the reason to the reduction in the voltage of the batteries towards the motors.
4. The sensors had been soldered in the opposite polarity and the distances between the sensors were to be adjusted and the resistance of the variable resistors needed to be varied.
5. The copper paths in the PCBs had been burned when the components are soldered. There were some loose connections, bad soldering, paths being disconnected due to some reasons in the PCBs.
6. We used a bread board instead of the main circuit PCB to avoid the errors in that Circuit board and to avoid the mess of wires.

These issues and causes we identified were rectified by having multiple test runs. Some PCBs (sensor unit and the main circuit) had to be remade due to some functional issues we faced.

Acknowledgement

The completion of this project was not an easy task for us as we were all novices to the robotics. Since this is our first robotics project, we faced a lot of challenges. So, we should remind of the people who were behind there to help us to accomplish this project.

First, we would like to thank our supervisor, Ms. Sakila Jayaweera who always encouraged us to complete this project and guided us through this project. Weekly meetings and mid reviews were great support to share our ideas with her and clear our doubts.

And, we would like to pay our gratitude to all the lecturers and instructors who were always there to help us. We are thankful to all personal in charge of laboratories and workshop for allowing us to use instruments and giving us the technical support. In addition, we must pay our gratitude to our colleagues for helping and sharing their knowledge with us.

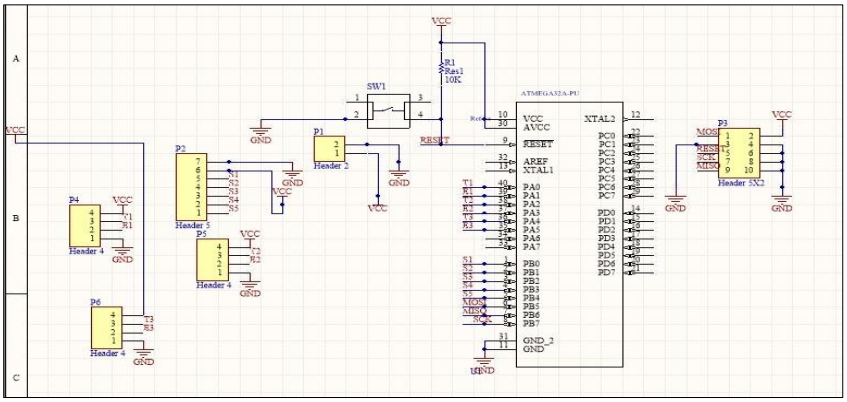
References

- https://www.microchip.com/mplab/avr-support/atmel-studio-7
- http://wiki.sunfounder.cc/index.php?title=Motor_Driver_Module-L298N
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1455479/
- https://www.instructables.com/id/DIY-Line-Follower-Sensor-Array/
- https://www.instructables.com/id/Robot-Line-Follower/
- http://gbulinefollowerrobot.blogspot.com/2015/02/blog-post.html?m=1
- https://ishankgulati.github.io/posts/Line-Follower-using-AVR-ATmega-32/
- https://electrosome.com/ultrasonic-distance-sensor atmega32/#Circuit_Diagram
- https://robotresearchlab.com/2019/02/16/pid-line-follower-tuning/
- https://pdf1.alldatasheet.com/datasheet-pdf/view/77378/ATMEL/ATMEGA32.html
- https://pdf1.alldatasheet.com/datasheet-pdf/view/106291/ETC/7805.html
- https://www.avrfreaks.net/forum/how-do-i-read-single-pin
- https://www.youtube.com/watch?v=jcDS8iaYS_Q
- https://www.electronicwings.com/avr-atmega/ultrasonic-module-hc-sr04-interfacing-with-atmega1632
- https://www.electronicwings.com/avr-atmega/atmega1632-pwm

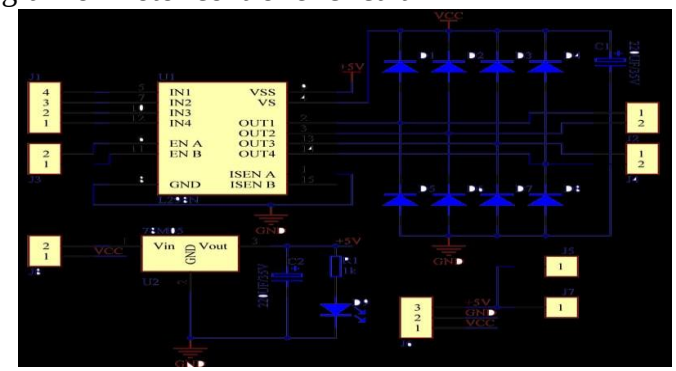
Appendix – 01

The following schematic diagrams were designed using Orcad Capture CIS 9.2 and Altium Designer software while the layouts were manipulated using Orcad Layout 9.2 and Altium Designer software.

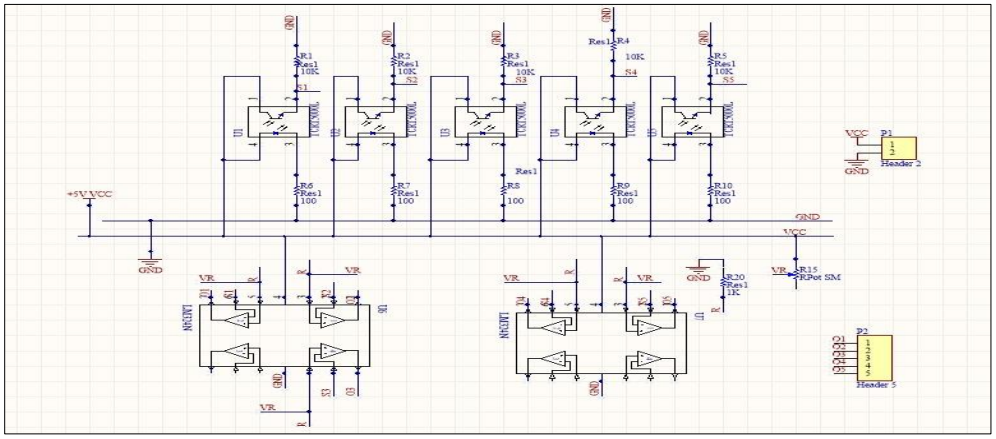
- The schematic diagram of main circuit



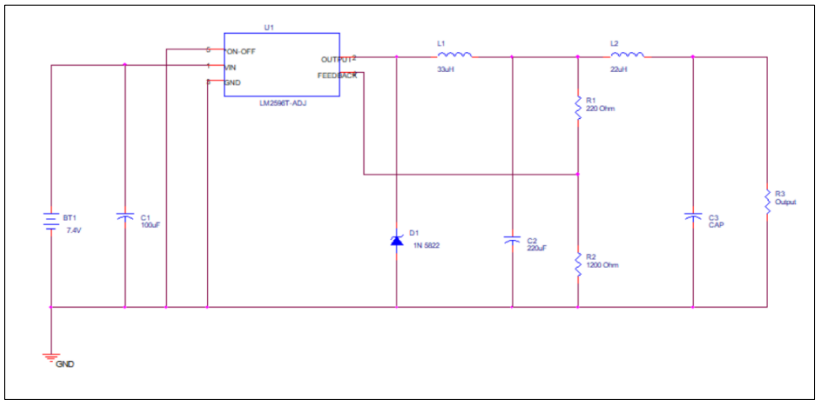
- The schematic diagram of motor controller circuit



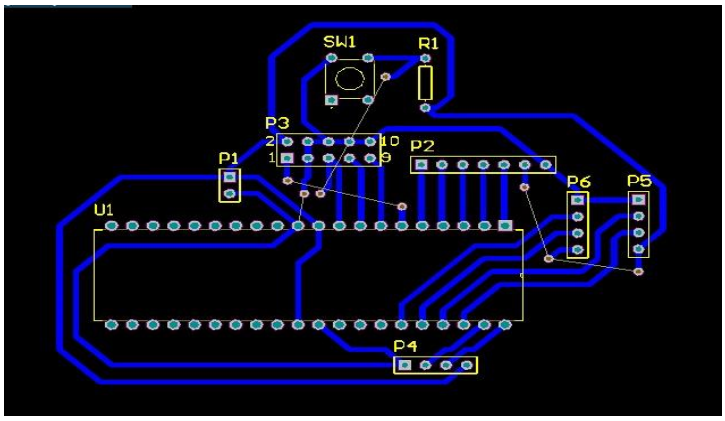
- The schematic diagram of IR sensor panel



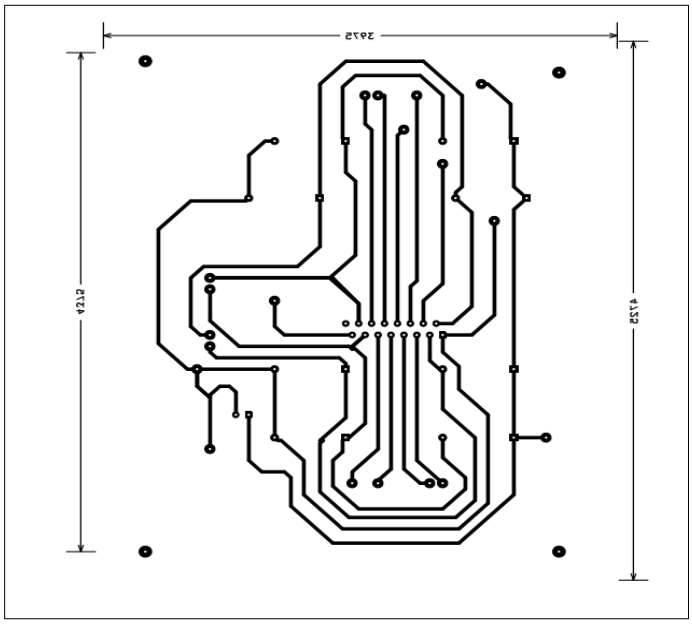
- The schematic diagram of power supply unit



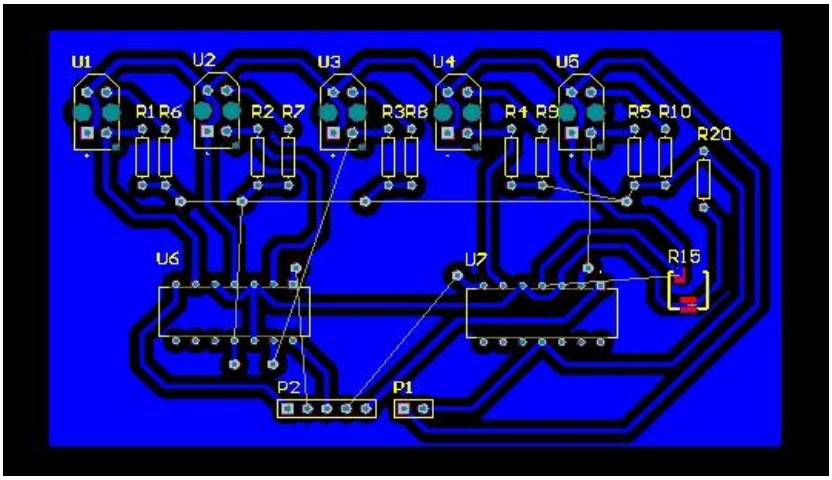
- PCB layout of main circuit



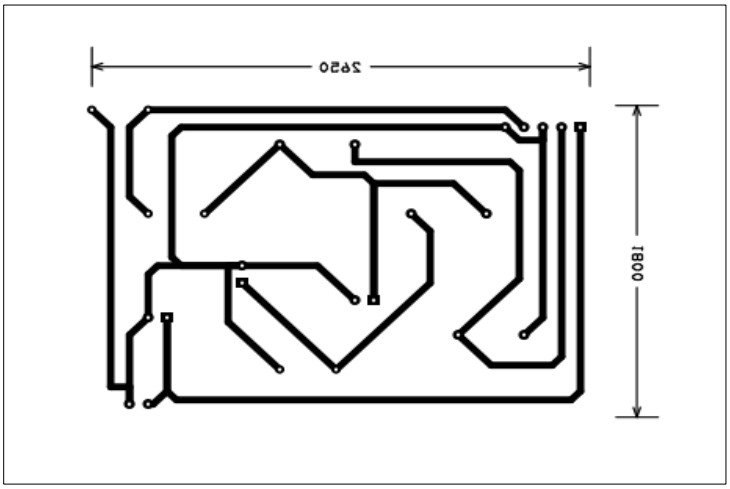
- PCB layout of motor controller circuit

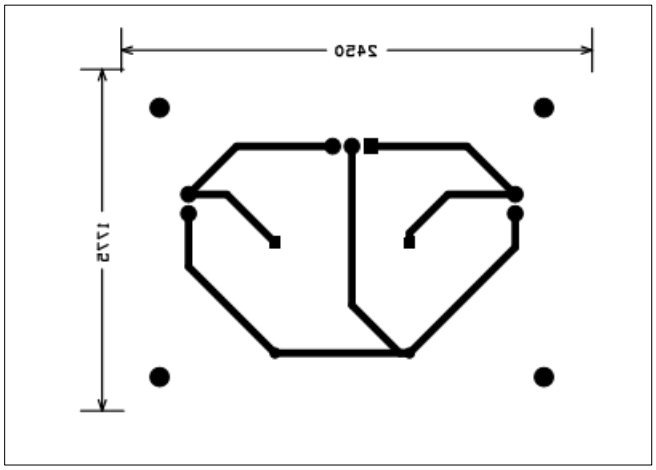


- PCB layout of IR sensor array



- PCB layout of power supply circuit





Appendix – 02

The following code is developed as a GCC C Executable Project on Atmel Studio 7.0 and was utilized in Atmega 32A microprocessor for achieving the tasks.

```
#ifndef F_CPU
#define F_CPU 1600000UL
#endif

#include <avr/io.h>
#include <util/delay.h>
#include "motors.h"
#include <stdlib.h>
#include <avr/interrupt.h>
#define MAXSPEED 150
#define ROTATESPEED 150
#define TURNSPEED 150

static volatile int pulse = 0;
static volatile int i = 0;
void line_follower();
char sensors=0;
char line_break=0;
char location = 3;
char locationIdentifier = 0;
int main(void){
	DDRA = 0xE0;
	init_motors();
	while(1)
	{
		Ultra_sound();
		if(Ultra_sound() == 1){
```

```
            set_motors(0,0);
            break;
}
else
{
sensors = PINA;
if (sensors == 0b00000000)
{
            if(location == 0)
                        {}
            else if ((location == 1) && (locationIdentifier = 1)) // Right turn
                        set_motors(TURNSPEED,-ROTATESPEED);
            else if (((location == 2) || (location == 3)) && (locationIdentifier = 1)) // left turn
                        set_motors(-ROTATESPEED,TURNSPEED);
            else if ((location == 2) && (locationIdentifier = 2))
                        set_motors(TURNSPEED,-ROTATESPEED);
            else if ((location == 3) && (locationIdentifier = 2))
                        set_motors(-ROTATESPEED,TURNSPEED);
            set_motors(MAXSPEED,MAXSPEED);
}
else if (sensors == 0b00011111)
{
            if(location == 0)
                        {}
            else if ((location == 1) && (locationIdentifier = 1))
                        set_motors(TURNSPEED,-ROTATESPEED);
            else if (((location == 2) || (location == 3)) && (locationIdentifier = 1)) // left turn
                        set_motors(-ROTATESPEED,TURNSPEED);
            else if ((location == 2) && (locationIdentifier != 1))
                        set_motors(TURNSPEED,-ROTATESPEED);
```

```
                else if ((location == 3) && (locationIdentifier != 1))
                        set_motors(-ROTATESPEED,TURNSPEED);}
        else
        {
                line_follower();
        }
        }}}
ISR(INT0_vect)
{
        if(i==1){
                TCCR1B = 0;
                pulse = TCNT1;
                TCNT1 = 0;
                i=0;
        }
        if(i==0){
                TCCR1B |= 1<<CS10;
                i=1;
        }
}
int Ultra_sound()
{
        DDRD = 0b00000001;   // PIND0 -- TRIGGER ,  PIND2 -- ECHO
        GICR |= 1<<INT0;
        MCUCR |= 1<<ISC00;
        sei();
        while(1)
        {
                float distance;
                PORTD |= 1<<PIND0;
```

```
                _delay_us(20);
                PORTD &= ~(1<<PIND0);
                distance = pulse/58;
                if (distance < 5){
                        return 1;
                }
                else{
                        return 0;
                }
}}
void line_follower()
{
        while(1){
                sensors = PINA;
                //GO STRAIGHT
                if (sensors == 0b00011011){
                        set_motors(MAXSPEED,MAXSPEED);}
                else if(sensors == 0b00010001){
                        set_motors(MAXSPEED,MAXSPEED);
                }
                //RIGHT TURN
                else if((sensors == 0b00000001) || (sensors == 0b00000011) || (sensors == 0b00000111) || (sensors == 0b00001111) || (sensors == 0b00000101))
                {
                        set_motors(TURNSPEED,-ROTATESPEED);
                        _delay_ms(30);
                }

                //SLIGHT RIGHT TURN
                else if((sensors == 0b00010011) || (sensors == 0b00010111))
                {
```

```
                set_motors(MAXSPEED,(0.5*MAXSPEED));
                _delay_ms(10);
            }
            //LEFT TURN
            else if((sensors == 0b00010000) || (sensors == 0b00011000) || (sensors == 0b00011100) || (sensors == 0b00011110) || (sensors == 0b00010100))
            {
                set_motors(-ROTATESPEED,TURNSPEED);
                _delay_ms(30);
            }
            //SLIGHT LEFT TURN
            else if((sensors == 0b00011001) || (sensors == 0b00011101))
            {
                set_motors((0.5*MAXSPEED),MAXSPEED);
                _delay_ms(10);
            }
            else{
                set_motors(ROTATESPEED,-ROTATESPEED);
                //PORTB &= ~(1<<PINB0);
            }
        }
}
```

Motors Driving Code (developed as a library in Atmel Studio 7.0)

```
#ifndef MOTORS_H_
#define MOTORS_H_
#include <avr/io.h>
#define leftMotorPWMPin         OCR0 //pinb3
#define rightMotorPWMPin        OCR2 //pind7
void init_motors(void);
void set_motors(int leftMotorSpeed, int rightMotorSpeed);
```

```
/*      Initializes the OCR0 and OCR2 pins for phase correct PWM.
 *      with non-inverting mode. OCR0/OCR2 is cleared on compare match
 *      when up-counting and is set on compare match when down-counting.
 */
void init_motors(){
        //Configure PWM pins OCR0 and OCR2 to output mode
   DDRD |= (1<<PIND7);
        DDRB |= (1<<PINB3);
        //Configure motor direction control pins to output mode
        DDRC |= (1<<PINC0) | (1<<PINC1) | (1<<PINC6) | (1<<PINC7);
        //set phase correct PWM, NON inverting mode and set a pre scalar of 64
        TCCR0 |= (1<<WGM00) | (1<<COM01) | (1<<CS00) | (1<<CS01);
        TCCR2 |= (1<<COM21) | (1<<WGM20) | (1<<CS22) ;
        //Sets TOP value to be 250
        TCNT0 = 250;
        TCNT2 = 250;
}
/*      Sets the speed and direction of the two motors. The value of each
 *      of the arguments can range from -250 to +250. A negative sign
 *      is used when the direction of the motor is to be reversed.
 */
void set_motors(int leftMotorSpeed, int rightMotorSpeed){
   if(leftMotorSpeed >= 0){
      leftMotorPWMPin = leftMotorSpeed;
                PORTC |= (1<<PINC0);
                PORTC &= ~(1<<PINC1); //SET THE MOTOR TO ROTATE IN THE FORWARD
DIRECTION.
   }
   else{
      leftMotorPWMPin = -leftMotorSpeed;
        PORTC |= (1<<PINC1);
```

```
        PORTC &= ~(1<<PINC0);           //SET THE MOTOR TO ROTATE IN THE OPPOSITE DIRECTION.
    }
        if(rightMotorSpeed >= 0){
      rightMotorPWMPin = rightMotorSpeed;
        PORTC |= (1<<PINC6);
        PORTC &= ~(1<<PINC7); //SET THE MOTOR TO ROTATE IN THE FORWARD DIRECTION.
    }
    else{
      rightMotorPWMPin = -rightMotorSpeed;
      PORTC |= (1<<PINC7);
      PORTC &= ~(1<<PINC6); //SET THE MOTOR TO ROTATE IN THE OPPOSITE DIRECTION.
    }
}
#endif
```