



Faculty of Engineering, Architecture and Science

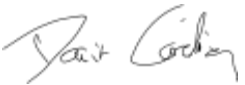




Department of Mechanical and Industrial Engineering

Program: Mechanical Engineering

Course Number	MEC 733
Course Title	Microprocessor Systems
Semester/Year	Winter 2019
Instructor	Dr. He

Project Report

Section No.	5
Group No.	5
Submission Date	
TA Name	Mehraz Haque

Name	Student ID	Signature*
Davin Cornelius	xxxx00051	
Alexandru-Georgian Porojan	xxxx71344	
Parsa Manafzadeh Tabriz	xxxx23033	
Sina Sartipzadeh Amirghasemi	xxxx01049	
Yalda Alemi	xxxx58274	

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*

<http://www.ryerson.ca/senate/policies/pol60.pdf>

Abstract

The purpose of this project was to familiarize with:

In the first part, encoder and distance sensor, method of following wall and method of turning in the second part of the project. And to learn and use the concepts presented in this course. The program was designed to follow line A after pressing the bump switch and wait for 3 seconds after reaching line B. The Mechbot would then follow the maze and show mission accomplished on the LCD after exiting the maze.

Table of Contents

Abstract	2
Table of Contents	2
Introduction	4
Problem Definition	5
Solution and Implementation	6
Code	6
Software flow chart	7
Discussion of Team Roles	9
Conclusion and Recommendations	10
Appendix	11
Program listing	11

Introduction

This project was a practice on programming the mechbot to perform several functions. The device would first follow a winding line with cuts and sensor blocks on its path, then follow a maze with the goal of not only exiting the maze but also doing so without collision and as fast as possible. Each section was practiced throughout the course labs, and really put together in the final project. The program written will only work best on the robot it was written on as each mechbot is considerably different in what the sensors pickup and motor power, even though they all have the same components.

The report is formatted in such a way as to, first describe the challenge in detail, then the code used for the mechbot is shown and a description is given on why each section was used, a flowchart is then given to help better understand the program flow. Finally we will describe how each team member contributed to the final project.

We will end on recommendations for improvements to the code and the mech bot itself for future iterations.

Problem Definition

This case study is consisted of two tasks : 1. Line following part , 2.Maze part . For the line following part the MECHBOT is placed at line A and the bump switches are read through the program .The screen reads “press bump switch to start”, once pushed , it starts moving forward following the black tape to reach line B while showing on the screen "end of the line!" and delay for 3 seconds .Further , to enter the maze after the delay , the motors are set to a quick forward of value 800 for both left and right set of wheels to have sufficient power to go through the ramp and enter the maze and giving a “we are in boys” feedback on the screen as the time delay is carried out .For the second task of the case study , after the MECHBOT has entered the maze the wall following part of the code is being used . In this part , by taking advantage of the P-control material from week 10 lecture notes , the positioning of the bot is adjusted by reading the distance sensor (analog 6) and finding the deviation from the goal (error) and multiplying by P closed loop value which is unique to the MECHBOT that was used to develop the code .This updated left and right adjusted motor speed is being changed constantly as the it progresses through the maze .The screen shows “Going through maze” while in maze and “Mission accomplished “ as it exits the maze going down the ramp and coming to a stop .

Solution and Implementation

1. Code

- a. Strategy , algorithm , method from the original that improved the control

The code is commented to explain what is happening in each section. The code was improved in several iterations, where we went from a long main function to a more object oriented approach.

- b. List one iteration

The main function represents the only iteration that gets run. This will call many of the underlying commands we coded.

```
int main(void)
{
/*
*We first setup the LCD screen, this will be used to debug the mecbot in real time, as it will
*show the sensor outputs*/
    initSoftSerial();
    setLCDBackLight(147);
    clrLCD();
    moveLCDCursor(0);
/*The init commands will allow for running the motor and reading the sensors*/
    initMotor();
    initADC();
/*portSetup() will require no inputs, and will only work on the mecbot layout, go to the function
for more information on it. */
    portSetup();
/*This section will perform the actual challenges. The user can start the program by pushing the
*bump switch. Each section will then have a quick forward function which will take care of any
*transition movements needed between states.
*Each function is explained separately inside the function itself.*/
    waitForBumpStart();
    clrLCD(); moveLCDCursor(0); lcdPrint("following line!");
    quickForward(1000);
    followLine();
    quickForward(2000);
    clrLCD(); moveLCDCursor(0); lcdPrint("Going through maze");
    navigateMaze();
    clrLCD(); moveLCDCursor(0); lcdPrint("Mission accomplished");
```

```
quickForward(500);  
/*We end the program with applying the breaks*/  
motor(0,0);  
}
```

c. What sensors used

The sensors used for this part were distance sensors (left , centre and right)

Software flow chart

Line Following:

ADC was initialized to make the analogs able. Line 1,2,3,4 were read from analog 0,1,2,3 respectively. The strategy used in this part was to set the motor to go forward if lines 2 and 3 are on the tape and the reading is more than the threshold for both of them. If line 2 is not on the tape which means that the reading of the analog is less than the threshold but line 4 is on the tape and the reading is more than the threshold, then the motor's speed for the right side of the Mechbot would be zero and the speed of the left side would be a positive value to force the machine to turn right. When line 1 is not on the tape but line 4 is, the situation is the same and the machine needs to turn right. On the other hand, if line 1 is on the tape but line 3 or 4 are not, then the Mechbot needs to turn left. Therefore, the speed of the left side would be zero and the speed of the right side would be a positive value.

Encoder Reading:

Encoders were used for turning at a special angle. The left encoder was connected to Port C pin 2 and the right encoder was connected to Port C pin 3. They were initialized as inputs and the high pull-up resistors were activated. The strategy used in this part was to read the value of the encoder and for any toggle in the value, increment the total number of the readings by 1. This was possible by using if loops and checking if the current value of the encoder is equal to the previous value or not. When the total number of the readings reached the desired value which was calculated for the required angle, then the motor would stop.

Starting the program using bump switches:

To start the Mechbot, one of the bump switches had to be pressed. Three bump switches were connected to Port D, pins 3,4,5. The pins were initialized as inputs and the internal pull-up

resistors were activated. Using a while loop the program would stay on the loop until one of the switches is pressed. Then it would exit the loop and continue reading the rest of the code.

Discussion of Team Roles

For this project, the parts for the coding were done during the lab hours and all of the group members participated equally. And the report was done through meetings and the workload was distributed equally.

Conclusion and Recommendations

For this setup we utilized many functions that will only apply to the specific mech bot we used. This does become an issue if there are no batteries available or the mechbot needs service. A solution for future editions is to include a auto calibration feature in the code. Since the LCD is on all mech bots, it can be used to communicate to the user how to position the bot for calibration. This will create a more robust code, and something much more applicable for a final project design for real world applications.

Appendix

TABLE 1 Effects of independent P, I, and D tuning on closed-loop response.

For example, while K_I and K_D are fixed, increasing K_P alone can decrease rise time, increase overshoot, slightly increase settling time, decrease the steady-state error, and decrease stability margins.

	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
Increasing K_P	Decrease	Increase	Small Increase	Decrease	Degrade
Increasing K_I	Small Decrease	Increase	Increase	Large Decrease	Degrade
Increasing K_D	Small Decrease	Decrease	Decrease	Minor Change	Improve

Source: http://eprints.gla.ac.uk/3815/1/IEEE_CS_PID_01580152.pdf

Program listing

```
#include <mechbotShield.h>
#include <avr/io.h>
#include "MEC733_I_O.h"

// global variables
int Lms, Rms, measure, derivative, error, last_error;
int turnSpeed = 900, maze_speed = 700, forward_stopping_distance = 200, goal = 280; //
forward_stopping_distance = 180, goal = 300;
float kp = 5;
float kd = 15;
float error_scale = 1.5;
kp = kp * error_scale;
kd = kd * error_scale;

int t = 300;
int surface_changes = 0;
bool surface_previous;

/*
 * play around with right distance sensor angle, maze_speed, kp
 * get some numbers for forward_stopping_distance, goal
 */
```

```

int main(void)
{
/*
*We first setup the LCD screen, this will be used to debug the mecbot in real time, as it will *show the
sensor outputs*/
  initSoftSerial();
  setLCDBackLight(147);
  clrLCD();
  moveLCDCursor(0);
/*The init commands will allow for running the motor and reading the sensors*/
  initMotor();
  initADC();
/*portSetup() will require no inputs, and will only work on the mecbot layout, go to the function for
more information on it. */
  portSetup();
/*This section will perform the actual challenges. The user can start the program by pushing the *bump
switch. Each section will then have a quick forward function which will take care of any *transition
movements needed between states.
*Each function is explained separately inside the function itself.*/
  waitForBumpStart();
  clrLCD(); moveLCDCursor(0); lcdPrint("following line!");
  quickForward(1000);
  followLine();
  quickForward(2000);
  clrLCD(); moveLCDCursor(0); lcdPrint("Going through maze");
  navigateMaze();
  clrLCD(); moveLCDCursor(0); lcdPrint("Mission accomplished");
  quickForward(500);
/*We end the program with applying the breaks*/
  motor(0,0);
}

void navigateMaze(void)
{
  bool check = 0;

  while (1)
  {
    check = hugRight(); // hugRight returns 1 if out of the maze
    if (check)
    { return; }
    delay_ms(t);
    // turn away from the wall in front

```

```

    if( analog(4) < 200 ) // if path clear to the left, turn left
    { turnLeft(); delay_ms(t); }
    else
    { turnAround(); delay_ms(t); }
  }
}

bool hugRight(void)
{
  int bump1, bump2, bump3;
  int turn = 0;
  last_error = 0;

  while (1)
  {
    // stop and exit with 0 if there is a wall in front
    if (analog(5) > forward_stopping_distance)
    { motor(0,0); return(0); }

    // exit with 1 if out of maze
    measure = analog(6);
    if ( trackSurface() )
    {
      if( surface_changes > 2 )
      {
        if ( measure < 50 && analog(4) < 50 )
        { motor(0,0); return(1); }
      }
    }
  }

  // keep adjusting speed based on distance from right wall
  error = measure - goal;
  derivative = error - last_error;
  last_error = error;
  turn = int( kp*error + kd*derivative );
  Lms = maze_speed - turn;
  Rms = maze_speed + turn;
  motor(Lms,Rms);
}
}

```

```

bool trackSurface(void)
{
    /*For the mecbot to change between the line following and maze sections it will utilize expected sensor
    readings to determine if it has exited or entered a new section\*/
    uint16_t line_1, line_2, line_3, line_4, mazeThresh = 900;
    bool surface;
    line_1 = analog(0);
    line_2 = analog(1);
    line_3 = analog(2);
    line_4 = analog(3);
    if (line_1 > mazeThresh || line_2 > mazeThresh || line_3 > mazeThresh || line_4 > mazeThresh) // in
    maze
    {
        surface = 0;
    }
    else { // out of maze
        surface = 1;
    }
    if (surface != surface_previous) {
        surface_changes++;
        surface_previous = surface;
    }
    return (surface);
}

/*turn left and around are used to quickly force the mecbot in a specific direction if it goes off course or
is about to hit. These are used in maze. */
void turnLeft(void)
{ encoderDrive(15, -turnSpeed, turnSpeed); }

void turnAround(void)
{ encoderDrive(31, -turnSpeed, turnSpeed); }

void encoderDrive(int enReadings, int velL, int velR)
{
    /*This function is used for mecbot rotation.
    *>> enReadings >> is the total number of encoder readings needed to turn the mecbot. Use the formula :
    number of encoder readings = (angle * 10.3) / (10.2*10^(-1))
    * velR, velL are the corresponding wheel speeds. These are generally kept the same. */
    int left_en_total=0, right_en_total=0, left_en_cur, right_en_cur, left_en_pre, right_en_pre;
    motor(velL, velR);

    while (1)
    {

```

```

left_en_cur=PINC&(1<<PINC2);
right_en_cur=PINC&(1<<PINC3);

if(left_en_cur!=left_en_pre)
{ left_en_total++; left_en_pre = left_en_cur; }

if(right_en_cur!=right_en_pre)
{ right_en_total++; right_en_pre = right_en_cur; }

if ((left_en_total>=enReadings)&&(right_en_total>=enReadings))
{ motor(0,0); return; }

delay_ms(10);
}
}

void quickForward(int t)
{
  motor(800,800); //a quick straight line forward
  delay_ms(t);
}

void followLine(void)
{
  /*Line follow will check each of the 4 sensors on the bottom of the mechbot, there are 5 possible
  conditions of what each sensor can see. These conditions are outlined in the 5 if statements below. As
  each sensor (1 & 4 ) see black tape, corrective measures are taken */
  uint16_t line_1, line_2, line_3, line_4, thresh = 750, floorR = 600, lineR = 850, t = 10;
  int turn, line_speed = 800, velF = line_speed, velR = -velF;

  motor(line_speed,line_speed);
  delay_ms(500);

  while (1)
  {
    line_1 = analog(0);
    line_2 = analog(1);
    line_3 = analog(2);
    line_4 = analog(3);

    if (line_2 > thresh & line_3 > thresh)
    { motor(velF, velF); }
    else if (line_2 < thresh & line_4 > thresh)

```

```

    { motor(velF, 0); }
    else if (line_1 > thresh & line_3 < thresh)
    { motor(0, velF); }
    else if (line_1 < thresh & line_4 > thresh)
    { motor(velF, velR); }
    else if (line_1 > thresh & line_4 < thresh)
    { motor(velR, velF); }
    else
    { motor(velF, velF); }

    if (line_1 > thresh && line_2 > thresh && line_3 > thresh && line_4 > thresh)
    { motor(0,0);
      clrLCD(); moveLCDCursor(0); lcdPrint("end of the line!");
      delay_ms(3000);
      return; }

    // delay_ms(t);
  }
}

void waitForBumpStart(void)
{
  /*This function will allow the user to control when to start the challenge. The user can press any of the
  buttons, and the mechbot */
  int bump1, bump2, bump3;
  clrLCD(); moveLCDCursor(0); lcdPrint("press bump switch to start");

  while (1)
  {
    bump1 = PIND & (1 << PIND3);
    bump2 = PIND & (1 << PIND4);
    bump3 = PIND & (1 << PIND5);
    if ((bump1 == 0) | (bump2 == 0) | (bump3 == 0))
    { delay_ms(500); return; }
  }
}

void portSetup(void)
{
  /*The portSetup() function will be called from main. This will set up our bump switches as inputs, *These
  are connected to the PORTS on pind PD3,PD4 and PD5 are the Left, Left-center and *right switches
  respectively*/

```

```

// port D pins 3, 4 and 5 for bumper switches
DDRD = DDRD & (~(1<<DDRD3)|(1<<DDRD4)|(1<<DDRD5));
PORTD = PORTD | ((1<<PORTD3)|(1<<PORTD4)|(1<<PORTD5));
/*The Encoders are on PORT C pins 2 and 3, where 2 is for the right and left wheels *respectively. The
encoders will be used to determine the angle of rotation, and command related to will spin. Each encoder
will use the following formula for rotation:
*      number of encoder readings = (angle * 10.3) / (10.2*10^(-1)), where angle is in radians */
// port C pins 2 and 3 for encoder readings
DDRC = DDRC & (~(1<<DDRC2)|(1<<DDRC3));
PORTC = PORTC | ((1<<PORTC2)|(1<<PORTC3));
}

```