Introduction to Robotics (UE15EE347)

Mini Project

MOPBOT
Smart and autonomous mopping of floors

By
Kenneth Joel (01FB15EEE015)
Mohan S P (01FB15EEE019)

Guided by: Prof. Venkatrangan M.J

Department of Electrical and Electronics Engineering
PES University
Bengaluru-560085

Table of Contents

Part I
Literature review

1 Objective

To enable autonomous mopping of floors using a mobile robot equipped with sensors and necessary actuators

- Plan a path for the room such that the entire surface is optimally covered.
- Mopping the floor by means of a robotic actuator.

2 Introduction

Robots for household cleaning purposes have seen widespread acceptance over the past few years. Taking inspiration from this we intend to design one of our own so we gain some exposure to the challenges associated with the development of such products. We hope to develop a prototype of an autonomous house mopping bot taking inspiration from a few commercially available products.

3 Methodology

Interface the following components with Firebird V

- DC Motors (For movement of the bot)
- LCD (For debugging)
- Sharp IR sensor (To measure the distance from any obstacle while navigating)
- Ultrasound (To measure the distance from the wall while navigating)
- Servo motors (To enable the motion of the actuator)

After interfacing all the above components successfully integrate all of them to make MOPBOT.

Part II

Hardware used

4 Firebird V by Nex Robotics

5 MeArm

6 Ultrasound sensor

7 Sharp IR sensor

Part III

Mathematical modeling of MeArm

Given below is the DH representation of the MeArm.

a_{i-1}	α_{i-1}	d_i	θ_i
$0.0cm$	$0°$	$3.0cm$	$\theta_1°$
$1.5cm$	$-90°$	$2.7cm$	$\theta_2°$
$8.5cm$	$0°$	$0.0cm$	$\theta_3°$
$8.0cm$	$0°$	$0.0cm$	$0°$

Using the above representation the transformation matrix from the universal frame to the end effector was obtained as shown below

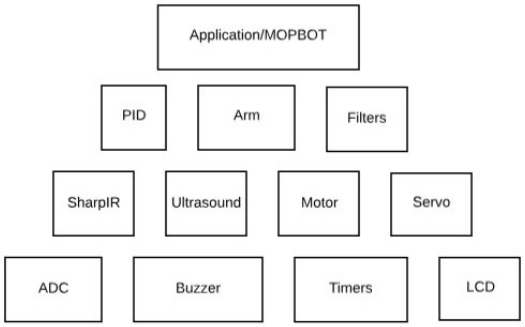
$$\begin{pmatrix} \cos(b+c)\cos(a) & -\sin(b+c)\cos(a) & -\sin(a) & \cos(a)\,\sigma_1 - 2.7\sin(a) \\ \cos(b+c)\sin(a) & -\sin(b+c)\sin(a) & \cos(a) & 2.7\cos(a) + \sin(a)\,\sigma_1 \\ -\sin(b+c) & -\cos(b+c) & 0 & 3 - 8\sin(b) - 8\sin(b+c) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

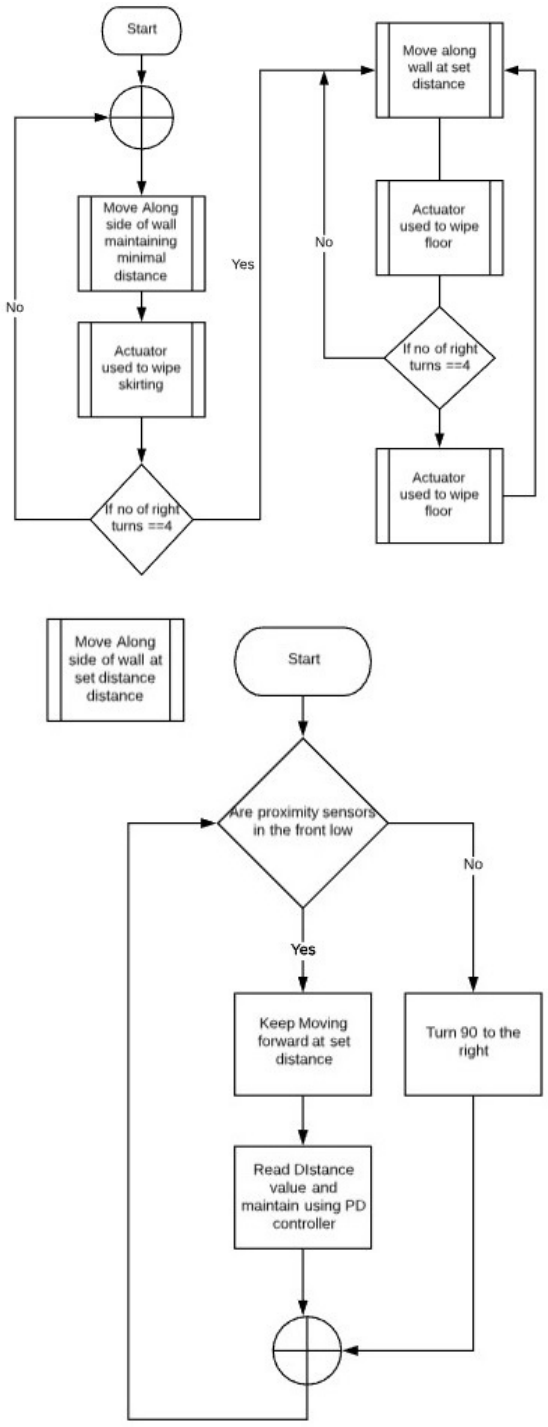
$$\sigma_1 = 8.0\cos(b+c) + 8.0\cos(b) + 1.5$$

Part IV

Software architecture



Part V
Flowcharts



Part VI
Design details

The design specifications of the project can be broadly classified into

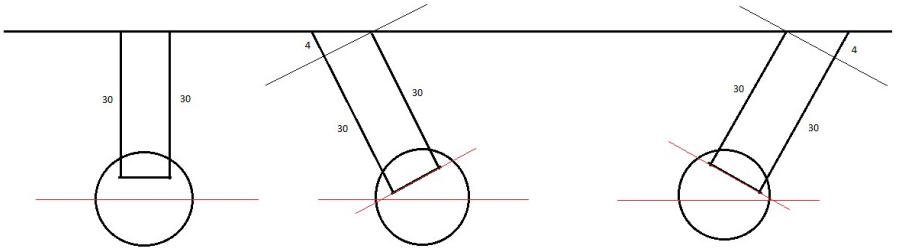
- Navigation
- Actuator kinematics

8 Navigation

The core problem statement of the navigation aspect of this project was to efficiently cover the floor space to be cleaned. The approaches we took are described in the flowcharts that are shown later on in this report. However, one key design concern was obtaining reliable distance data from the left and front of the bot. To ensure quality of this data a few steps were taken.

Firstly, a moving average of sensor data was taken to act as a low pass filter, the length of this filter depended on both the type of sensor and the type of response required.

Secondly, it was found that due to the angle the bot makes while turning to correct itself as instructed by the controller there is sometimes an error in the sensor reading, to take care of this, two sensors of the same type are used on the left side and the lower of the two readings is considered as the true distance. This point is illustrated in the figure below.



Lastly, the P controller used had a dead band that helped reduce the over-sensitivity of the system and helped it stay on a straight path.

8.1 Ultrasound sensor interfacing code

The ultrasound sensor we used was a time-based distance measurement sensor. Since, we wanted to interface two ultrasound sensors and we were running short of timers we decided to use a single timer and operate the two ultrasounds serially. The code is as shown below

8.1.0.1 Timer initialization

```
35  // Timer 4 initialized in normal mode for ultrasound sensor
36  // Prescale:256
37 void timer4_init()
38  {
39      TCNT4 = 0;
40      TCCR4A = 0x00;
41      TCCR4B = (0<<CS40) | (0<<CS41) | (1<<CS42);  // Timer mode with 256 prescler
42      sei();
43  }
```

8.1.0.2 Port Initialization

```
27 void ultrasound_config()
28  {
29      DDRJ = 0x3F;  //PJ0-5 as output and PJ6-7 as input
30      PORTJ = 0x00; //Output is set to 0
31  }
32
33  //PJ2 ----> Trig1
34  //PJ3 ----> Trig2
35  //PJ7 ----> Echo1
36  //PJ6 ----> Echo2
```

8.1.0.3 Trigger pin function

```
38  //Function to make the trigger pin high for a very short duration
39 void ultrasound1_trigger()
40  {
41      PORTJ = 0x04;
42      _delay_us(50);
43      PORTJ = 0x00;
44      _delay_us(10);
45  }
```

8.1.0.4 Distance calculation

```
//Function to calculate the distance
int distance1_us()
{
   ultrasound1_trigger();
   TIFR4=0x01;
   while ( (TIFR4 & 0x01) == 0x00 )
   {
       if ( ((PINJ & 0x80) == 0x00) && us_flag1==0 ) //Echo is low
       {
           TCNT4=0;
       }
       if ( ((PINJ & 0x80) == 0x80) && us_flag1==0 ) //Echo goes from low to high, timer starts
       {
           TCNT4=0;
           us_flag1=1;
       }
       if ( ((PINJ & 0x80) == 0x00) && us_flag1==1) //Echo goes from high to low, timer stops and the total time is calculated which is then used to calculate the distance
       {
           us_time1= (TCNT4H*256) + TCNT4L;
           us_dist1=(us_time1*33)/125;
           us_dist_int1=floor(us_dist1);
           filter_array3[i3]=us_dist_int1;
           i3++;
           break;
       }
   }
   TIFR4=0x01;
   us_flag1=0;
   if (i3>length-1)
   {
       i3=length-1;
       for (j3=0;j3<length-1;j3++)
       {   sum3=sum3+filter_array3[j3];
           filter_array3[j3]=filter_array3[j3+1];
       }
       sum3=sum3+filter_array3[length-1];
       filtered_distance3=(sum3/(length)) + 5;
       sum3=0;
   }
   return filtered_distance3;
}
```

8.2 Sharp IR sensor interfacing code

Unlike the ultrasound sensor, this gave a voltage based distance measurement. Therefore using ADC the voltage values were directly converted to distance using a relation given by the manufacturer.

8.2.0.1 Distance calculation

```
int distance_IRF()
{
   ADC_value2=ADC_Conversion(11);
   dist2 = (2799.6*(1/(pow(ADC_value2,1.1546))));
   filter_array2[i2]=dist2;
   i2++;

   if (i2>lengthF-1)
    {
       i2=lengthF-1;

       for (j2=0;j2<lengthF-1;j2++)
        {  sum2=sum2+filter_array2[j2];
           filter_array2[j2]=filter_array2[j2+1];
        }
       sum2=sum2+filter_array2[lengthF-1];
       filtered_distance2=sum2/(lengthF);
       sum2=0;
    }

   return filtered_distance2;
}
```

8.3 P control with deadband for navigation

```
int pid_speed_control(int setpoint, int process_variable)
{
   //The goal of this module is to maintain the orientation of the robot by controlling only the right motor, marginally speeding it up
   //or slowing it down as required.
       int error=0, correction =0, kp=2;
       error=setpoint-process_variable;

       //Note that is error > 1 then robot is moving away from the wall, if less than 1 then it is moving towards it.

       if (error > 1 || error < -1)
        {
           correction = kp*error;
           // Saturation of PWM values
           if (correction > 130)
           correction = 150;
           if (correction < -70)
           correction = -70;
        }

       else
        {
           correction=0;
        }

       return correction;
}
```

9 Actuator kinematics

Working with the servos at a relatively lower lever gave us a greater degree of control over the motor. The mathematical model of the arm was first derived by hand. These equations were then substituted in MuPad and then solved for, in MATLAB.

Care was taken to ensure that solutions provided is out of the reachable workspace by using thresholds to limit angles. Once point solutions were found and verified using inverse kinematics solutions, linear trajectories were solved for to generate the cleaning motion.

Power for the servo was not an issue as the Firebird V provides 3 ports that are dedicated for these high current applications.

9.1 Servo interface

9.1.0.1 Timer initialization

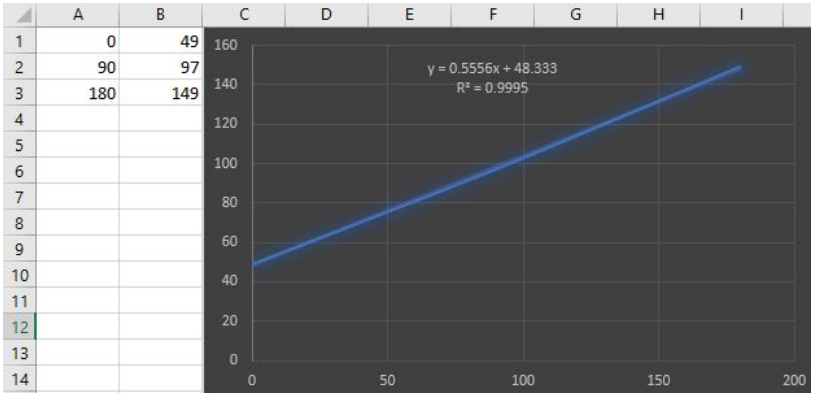
```
// Timer 1 initialized in PWM mode for servo control
// Prescale:256
// PWM 10bit fast, TOP=0x03FF
// Timer Frequency:56.000Hz
void timer1_init()
{
 TCCR1B = 0x00; //stop
 TCNT1H = 0x00; //Counter high value to which OCR1xH value is to be compared with
 TCNT1L = 0x00; //Counter low value to which OCR1xH value is to be compared with
 OCR1AH = 0x00; //Output compare Register high value for servo 1
 OCR1AL = 0x61; //Output Compare Register low Value For servo 1
 OCR1BH = 0x00; //Output compare Register high value for servo 2
 OCR1BL = 0x61; //Output Compare Register low Value For servo 2
 OCR1CH = 0x00; //Output compare Register high value for servo 3
 OCR1CL = 0x61; //Output Compare Register low Value For servo 3
 TCCR1A = 0xAB; /*{COM1A1=1, COM1A0=0; COM1B1=1, COM1B0=0; COM1C1=1 COM1C0=0}
                  For Overriding normal port functionality to OCRnA outputs.
                 {WGM11=1, WGM10=1} Along With WGM12 in TCCR1B for Selecting FAST PWM Mode*/
 TCCR1C = 0x00;
 TCCR1B = 0x0C; //WGM12=1; CS12=1, CS11=0, CS10=0 (Prescaler=256)
}
```

9.1.0.2 Port initialization

```
13 void servo_pin_config()
14 {
15     DDRB  = 0xE0;  //making PORTB 5,6,7 pin output
16     PORTB = 0xE0; //setting PORTB 5 pin to logic 1
17 }
```

9.1.0.3 Degree calculation

Servo uses the technique of PPM(Pulse Position Modulation) to turn to specified degrees. So first we took a few standard points such as 0°, 90° and 180°. Once we got the timer values for these angles, we linear curve-fit the data in Microsoft Excel to obtain the relation between timer values and degrees moved by the servo as shown below.



After obtaining the general relation as shown above we implemented the same in a function to turn the servo to any desired angle between 0° and 180° as shown below.

```
20 void servo_1(unsigned char degrees1)
21 {
22     float PositionPanServo1 = 0;
23     PositionPanServo1 = ((float)degrees1 * 0.5556) + 48.33;
24     OCR1A = (uint16_t)PositionPanServo1;
25 }
```

Using the above mathematical model and servo interface we obtained joint angles for the desired trajectories and coded it so that the actuator performs the desired cleaning actions.

10 MATLAB implementation of MeArm

Editor - D:\Projects\Robotics_project\Mearm_sim.m

EDITOR | PUBLISH | VIEW

Mearm_sim.m

```
1   clc; clear;
2
3   syms a;
4   syms b;
5   syms c;
6
7   %Desired 3D coordinates for which joint angles are to be obtained
8   x=7; y=3; z=3;
9
10  %Transformation Matrices
11  F01=[cos(a) -sin(a) 0 0 ; sin(a) cos(a) 0 0 ; 0 0 1 3 ; 0 0 0 1];
12  F12=[cos(b) -sin(b) 0 1.5; 0 0 1 2.7 ; -sin(b) -cos(b) 0 0 ; 0 0 0 1];
13  F23=[cos(c) -sin(c) 0 8.5 ; sin(c) cos(c) 0 0 ; 0 0 1 0 ; 0 0 0 1];
14  F3H=[1 0 0 8 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 1];
15  F0H=F01*F12*F23*F3H;
16
17  %Forming a system of linear equations and solving for the unknowns
18  eqns=[F0H(13)==x,F0H(14)==y, F0H(15)==z];
19  vars=[a,b,c];
20  [sola , solb, solc, param, cond] = solve(eqns,vars,'ReturnConditions', true);
21
22  %Limiting solutions to preferred range and obtaining the solutions
23  assume(cond(1));
24  interval = [sola >=-pi, sola <=pi];
25  solk=solve(interval,param(1));
26  vala=subs(sola, param(1), solk);
27
28  assume(cond(2));
29  interval = [solb >=-pi, solb <=pi];
30  soll=solve(interval,param(2));
31  valb=subs(solb, param(2), soll);
32
33  assume(cond(3));
34  interval = [solc >=-pi, solc <=pi];
35  solm=solve(interval,param(3));
36  valc=subs(solc, param(3), solm);
```

11 Navigation implementation

Using all the above obtained results and our decided algorithm for navigation, the navigation part of the project is as shown below.

MOPBOT.c — D:\Projects\Robotics_project\MOPBOT\MOPBOT\MOPBOT.c

```
30  int main(void)
31  {
32     init_devices(); //Initialize various devices on Firebird
33
34     while(1)
35      {
36         filtered_distanceF= distance_IRF();
37         filtered_distanceL_US1= distance1_us();
38         filtered_distanceL_US2= distance2_us();
39
40         true_distance= fmin(filtered_distanceL_US1,filtered_distanceL_US2);
41         forward(base+correction,base-offset);
42         correction=pid_speed_control(setpoint,true_distance);
43
44         if (filtered_distanceF < turn_distance_max && filtered_distanceF > turn_distance_min)
45         {
46             right_degrees(90);
47             _delay_ms(500);
48             turn_count++;
49         }
50
51         if (turn_count==4 && setpoint<55)
52         {  skirting_flag=0;
53             turn_count=1;
54             setpoint+=5;
55             turn_distance_max+=5;
56             turn_distance_min+=5;
57         }
58
59         if (skirting_flag==1)
60          {
61             arm_z_linear();
62          }
63
64         else
65          {
66              arm_y_linear();
67          }
68     }
69  }
```

Part VII
Learning experience

12 Indoor navigation

- Developing a module for the ultrasound sensor from the ground up was a good insight into the finer details of its working and the complexities involved.
- Since we used both IR and Ultrasound sensors we were able to actively compare their performance.
- Seeing the effect of moving average as a low pass filter in action was a good learning experience.
- We got to put our control systems theory to the test for the navigation algorithm for maintaining a fixed distance from the wall.
- The tricky algorithmic aspects like finding the orientation of the robot without any orientation sensors and using two ultrasound sensors with a single timer was a fun challenge.

13 3 DOF manipulator (MeArm)

- It was great to see our course theory applied to a real world use case when we developed a mathematical model for the MeArm.
- Seeing the inverse kinematics on a physical model and verifying our calculations was an exciting experience.
- We got to explore various possibilities for simulation of robotic manipulators
- Assembling the arm was a fun hands on experience which was crucial to developing an accurate model
- Working with the servo motors at a very low level despite being challenging gave us some useful insights.

Trying our best to adhere to efficient coding practices was also a good learning experience.

Part VIII
Result

14 Deliverables

- Program for Firebird and MeArm on Atmel studio
- MATLAB Simulations and MuPad Files.

15 Things achieved

- Ultrasound interfacing from scratch.
- Moving average filters on sensor data.
- Successful navigation of small rooms by maintaining distance from the wall using a P controller with deadband.
- Highly Modular, reasonably well structured code.
- Forward and inverse kinematics of 3DOF MeArm in MATLAB, which was verified on the hardware.
- Used the above model to plot linear trajectories.
- Servo interfacing from scratch using 10 bit Fast PWM on the Atmega 2560.
- Integration of both aspects of the project such that they work well together.

Part IX
Future work

- Develop a realistic simulation in MATLAB while retaining the current techniques used.
- Exploring more options for simulation (SimMechanics, ROS).
- Contribute to the MeArm community by making simulation available on open source software.

- Extend the basic navigation problem to something similar to the fuzzy controller used in the paper included in the Eyantra DVD.
- Obtain symbolic solution for inverse kinematics.
- Add linear segments with parabolic blends for joint angle rotations.
- Improve navigation to account for rooms of different shapes and for obstacles in the room.

Part X
References

16 Irobotbraava

Given below is a link to the Irobotbraava which is what we based our prototype on.
https://www.youtube.com/watch?v=gy2sqoWtCCQ

17 Irobotbraava jet

The braava jet is a variation of the same, without the centralized navigation system. Given below are a few useful links about the braava jet.
https://www.youtube.com/watch?v=fhI62jgVZhQ
https://www.youtube.com/watch?v=IOi_LIn0Qao