# Bug 2.0 Implementation on Epuck robot

Sandeep MANANDHAR
Omid MORADTALAB

April 22,2015

Instructor: Prof. Xevi Cufi

# 1 Introduction

In this report we will look at implementation of bug 2 algorithm for motion planning.
The core components of the algorithm are :
1) Create an M-line starting from start position to goal
2) Follow the line while sensing for any obstacles
3) On finding some obstacle, turn left and follow the obstacle until the intersection with M-line is met.

## 1.1 M-Line

The M-Line is described by the start location and goal point. From the two coordinates, we can produce an equation of line which shall be used for intersection with robot's trajectory later.

```
slope = gy/(gx+0.000001);
//we start from the goal location
//gx and gy are the goal coordinates
//0.0000001 added to avoid divide by 0 error
```

## 1.2 Move to goal

We constantly keep track of angular difference $\delta$, the distance to m-line and the distance to goal. To do this, we need coordinates of the bot as well as the heading angle. This is provided by the odometry computation.

```
void getDistances(){
  delta = atan2(gy - y, gx - x) - ang;
  dline = fabs(gy*x - gx*y)/sqrt(gy*gy + gx*gx);
  goalD = sqrt(pow(gx - x, 2) +(pow(gy - y,2)));
}
```

Before the robot starts to displace in xy-plane, we shall rotate the robot along z-axis so that it will directly face to the goal and minimize the $\delta$ error value.

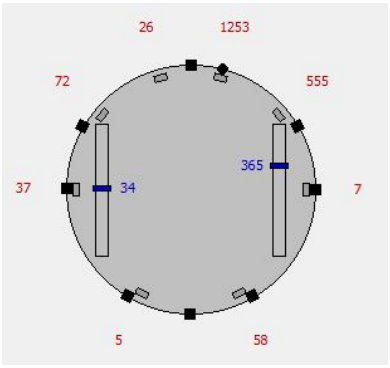
```
if(fabs(delta) > 0.035){
    if(delta > 0) bias = 100;
    else bias = -10;
    sr = delta*300 + bias;
    //p-controller for right speed
    if(sr > 800) sr = 800;
    else if(sr < -800) sr = -800;
    //sanity check
    sl = -sr; //speed for left
    flagRot = 1;
}
else{
sl = sr = 0;  //speed set to 0
flagRot = 0;
if(goalD > 0.005) flagMove = 1;
//if goal point has not been found
}
if(flagRot == 0 && flagMove == 1)
{
    if(fabs(goalD) > 0.008){
        sr = goalD*10 + 50;
        //p-controller for right speed
        if(sr > 800) sr = 800;
        else if(sr < -800) sr = -800;
        //not required though
        sl = sr;
    }
    else{
        sl = sr = 0;
        flagMove = 0;
        printf("MISSION COMPLETE!!!\n");

        for (i=0; i<10; i++) {
            wb_led_set(led[i],1);
        }
    }
}
```

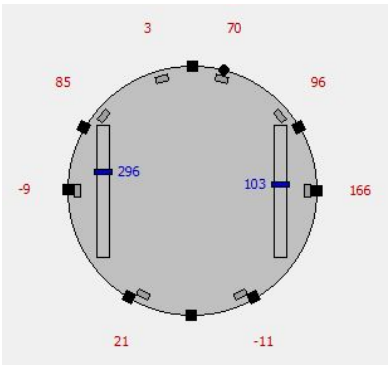
Here, we have employed P-controller for the speed control for both wheels. Since the error in angle as well as goal is being calculated intermittently, we control both of the wheels in their proportion. After few experimentation, the proportional constant can be finely tuned.

## 1.3 Obstruction

Our strategy to avoid and follow any obstacle is to turn left and follow it until the next m-line intersection. We have employed only 4 of the IR sensors out of the available 8 for simulation purpose which is sufficient. THe figure 1 show the two behaviours of the robot when an obstacle is found.




(a) Obstruction ahead towards right
Speed of right wheel > left wheel to rotate left




(b) Following the obstacle: Rightmost sensor shows high values while robot follows straight

Figure 1: Avoid and following behaviour:
The figures depict the sensors reading on obstacle and corresponding wheel speeds

```
if(wallEncounter == 1 && !findIntersection())
{
    double psErr =
    540 - (ps_values[2]+(ps_values[1] + ps_values[0]));
    //erros from sensor 0, 1 and 2 are compared with a constant
    //constant value was found with trial and observation
    sl = psErr/6 + 200;  //p-controller for left wheel speed
    sr = -psErr/6 + 200;//p-controller for right wheel speed
    if(sl > 800) sl = 800; else if(sl < -800) sl = -800;
    if(sr > 800) sr = 800; else if(sr < -800) sr = -800;
    //sanity check for the wheel speed
}
```

Here, we have employed a P-controller for the speed for the wheels. This makes the smooth rotation possible as well as a sufficient wheel control to follow the line with some oscillation.

### 1.4 M-Line found again!!

Once the robot has strayed away because of the obstruction, we keep looking for the next intersection with the m-line. To do this, we only need slope of the m-line and current coordinates of the robot.

```
bool findIntersection(){
    if(fabs(y - slope*x) < 0.03){
    printf("wait wait wait wait \n");
    return true;}
    return false;
}
```

The error function in the routine assumes that the m-line intercepts y-axis at origin. Our assumption is that the robot always starts from origin. For some arbitrary starting location, initial calculation is necessary to find the y-intercept and input it to the error function. The drawback of our implementation is that the execution of this $findIntersection()$ routine is sequential. Since, the robot is mobile most of the time, the function might not get called during the exact intersection point. We have put less tolerance for error to account for this factor. Nonetheless, this adds up to the global error. An efficient way to tackle this problem would be to call this routine from interrupt service table of the micro-controller.
After intersecting with the m-line, the robot heads towards the goal.

## 2 Conclusion

Bug 2.0 algorithm provides a good way to move from point A to point B. Unlike its other variants, this technique seems to be effective and predictable. The implementation was done in simulation and few initial tests in real system. The system can be further improved by the better control system of the wheels and better software implementation with interrupts to improve the response of processor for every change in state of the robot.

## 3 Demo Link

A video of the simulation can be found here:
`https://www.youtube.com/watch?v=tUP_Js2iTZQ`