Autonomous Robots
Bug 2 algorithm with the e-puck mobile robot

Al Arafat,Abdullah Abdulaziz,Micheale Hadera
April 2015
MSc in VIBOT 2014/15

1. INTRODUCTION

Obstacle following is one of the most basic problems in robot controlling and it constitutes an important step in higher level autonomous motion planning. It consists in obtaining information from the sensors and, based on that acquired knowledge, move the robot in the environment in such a way that it can avoid obstacle collision. This problem can be solved using the behavioral approach, which generates a specific motor response from a given perceptual stimulus.[1]

Bug 2 constitutes one of the Bug algorithms for path planning, and despite its simplicity, but effectiveness, it can guarantee that the goal position will always be found. In this algorithm, the robot heads towards the goal along the line that goes from the initial position to the goal, called m-line. When it senses an obstacle on its way, it follows the obstacle until it meets the m-line again.

The objective of this lab work is to implement the Bug 2 algorithm to go to the goal without crashing with obstacles. The algorithm exhibits two behaviors: head toward goal behavior and obstacle following. In this lab work, we combined the obstacle following behavior of the robot with its head toward goal behavior.And the report is organized in following manners. In the next section, the Bug algorithm is described briefly. Depending on the algorithm all the robot behavior is divided in to some states. Those states are over viewed and explained in details in the algorithm and implementation section.Some conclusive remarks on the laboratory assignment, e-puck robot and bug2 algorithm are mention in the Conclusion.

2. Bug Algorithm

The Bug algorithms are a path finding algorithms, which simplifies a bug movement into some simpler algorithms, to be adaptable for sensor based robots.

- The first algorithm is **Bug 0** algorithm. Here the robot starts moving towards the goal until obstacle is detected, once obstacle is found, it follow the obstacle wall until it can head toward the goal again (Figure 1 a). The draw back of Bug 0 algorithm is that it is not robust for complex environment(map).

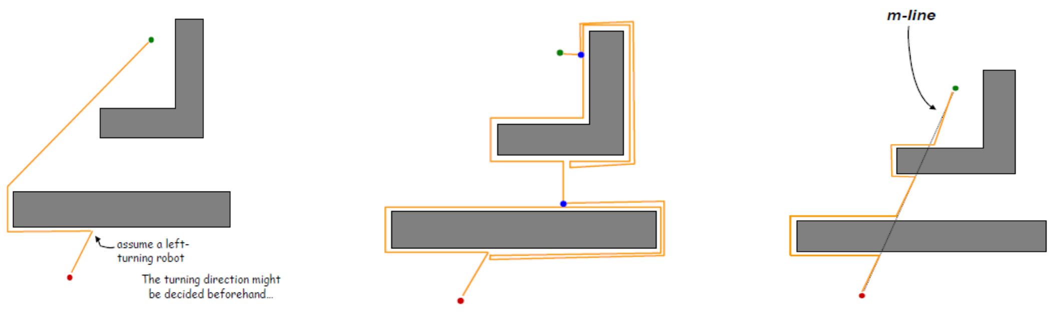
- In **Bug 1** algorithm, where the robot do not stop following the obstacle once direct path to goal is found, rather it keeps following the obstacle, until it complete the circumnavigation of the whole object. Then it goes back to the point from where the distance to goal is the lowest and start heading towards the goal again (Figure 1 b). Comparing to Bug 0 algorithm, Bug 1 algorithm is advantageous because no matter whether we start left or right we can reach the goal even for complex environment .

- The **Bug 2** algorithm is another type of algorithm where it do not circumnavigate the whole object rather first it computes a straight line between the initial robot position and goal position called m-line. Then it starts moving towards the goal following that line, once obstacle is detected, it follows the wall of the obstacle until the m-line is found again. Once found, it leaves the obstacle and follow the m-line until the goal is reached (Figure 2 c).

3. Algorithm and Implementation

3.1. Heading towards the goal

The aim of this behavior is to approach the m-line and follow it until reaching the goal. However, if an obstacle is encountered on the path, the robot will follow the obstacle which is another behavior until it finds again the m-line. While following the head towards goal behavior, the following 3 geometric conditions should have to be satisfied.


assume a left-turning robot
The turning direction might be decided beforehand...
m-line

(a) (b) (c)

Figure 1: Illustration of Bug algorithms (a) Bug 0 (b)Bug 1 (c) Bug 2

- Distance to the goal: It is the Euclidian distance between the center of the robot (rx, ry) and the goal position (gx, gy).

$$d_{goal} = \sqrt{(g_x - r_x)^2 + (g_y - r_x)^2} \quad (1)$$

- Distance to the m-line: It is the shortest Euclidian distance between a point (center of robot (rx, ry)) and the line connecting between the starting position to the goal position.To do so we define the m-line as line equation as follows:

$$\begin{aligned} ax + by + c = 0 \\ a = s_y - g_y \\ b = g_x - s_x \\ c = s_x * g_y - g_x * s_y \end{aligned} \quad (2)$$

where (s_x, s_y) is the starting coordinate of the robot.

$$d_{mline} = \frac{|ar_x + br_y + c|}{\sqrt{a^2 + b^2}} \quad (3)$$

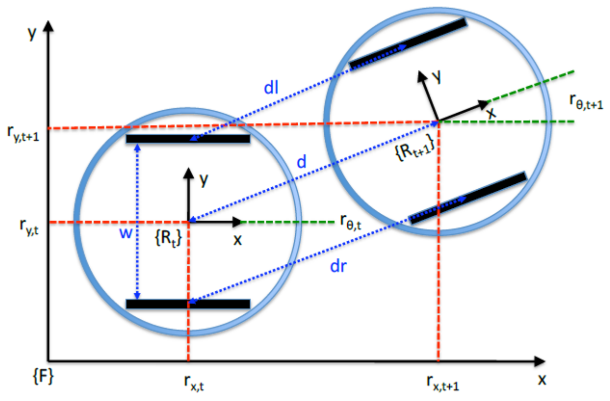
- Delta: It is the angle between the x-axis of the robot and the goal position, see figure (2).

$$delta = tan^{-1}\left(\frac{g_y - r_y}{g_x - r_x}\right) - \theta \quad (4)$$

Where θ is the angle between the horizontal and the x-axis of the robot.The implementation for the three geometry is described below

```
% Euclidean distance between the goal position
    and the robot position
Euclidean_dist = sqrt((goal_x - X)^2 + (goal_y
    - Y)^2);
% Angular distance between the X-axis of the
    robot and the robot-goal vector
Delta = wrapToPi(atan2(goal_y - Y, goal_x - X)
    - Theta);
Delta_degrees = Delta*180/pi;

```



(a)

Figure 2: Geometric representation of the robot.

To control this behavior, the previous equations for δ and the distance to the goal, will be used. If delta is higher than a threshold angle (i.e. 20°), we apply the maximum angular speed (60) and 0 linear speed in the proper direction. Otherwise, we apply the angular speed from maximum to 0 proportionally and constant linear speed (300).

When the robot is sufficiently close to the goal point, the distance to the goal will be relatively small (2 cm) and the robot stops considering that it has already reached the point.

```
function [linear,angular] = Head2Goal(goal_x,
    goal_y)
global go
global X
global Y
global Theta

% Euclidean distance between the goal position
    and the robot position
```

```
Euclidean_dist = sqrt((goal_x - X)^2 + (goal_y - Y)^2);
% Angular distance between the X-axis of the robot and the robot-goal vector
Delta = wrapToPi(atan2(goal_y - Y, goal_x - X) - Theta);
Delta_degrees = Delta*180/pi;

if Delta_degrees > 20
angular = 60;
linear = 0;
elseif Delta_degrees < -20
angular = -60;
linear = 0;
else
angular = 3 * Delta_degrees;
linear = 300;
end

if Euclidean_dist < 0.01
go = 0;
end

end

```

Finally the implemented code is tested with different positions and their trajectory results is shown in figure 3.

3.2. Obstacle following

The robot can conduct an obstacle following behavior using the high-level controller, that is, when it meets an obstacle, it circumnavigates the object. The main purpose is to keep the robot moving around the obstacle keeping almost a constant distance of 1 cm. The obstacle following behavior is divided into two subsequent steps :

- **Perpendicular rotation**: In this state, if the robot encounters an obstacle on its right side, then rotate left until it becomes perpendicular to the obstacle and vice-versa. By default, if the robot encounters an obstacle in front of it then it would tend to go left.
- **circumnavigation**: In this state, the robot would likely circumnavigate the obstacle till it finds a m-line which is nearer to the goal but not the previous hit point.

3.2.1. Perpendicular rotation. Whenever the robot encounters an obstacle in front of it, the rotational behavior mode gets activated. The goal of such rotation is to align the robot parallel to the obstacle, which then switch the robot to another state to circumnavigate the obstacle.
For this operation, the direction of the obstacle appearance has been checked. Two front right side sensors

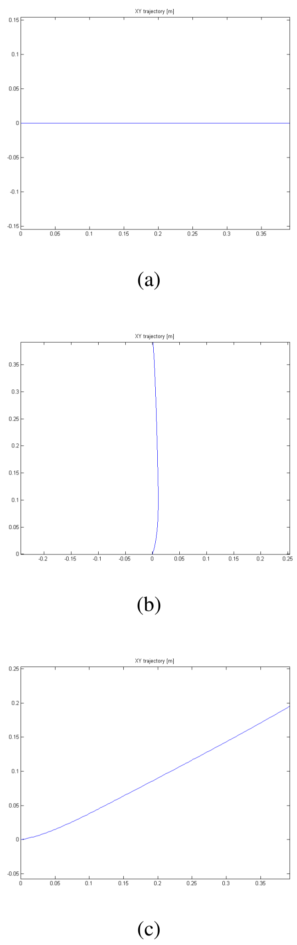


Figure 3: Trajectory results of head towards goal (a) goal (40,0) (b) goal (0,40) (c) goal(40,20)

(IR0 & IR1) have been used to measure the distance to check whether there is an obstacle on the right side of the robot. If it finds some obstacle it rotates till the (IR2) senses 1cm distance form the obstacle.
And two of the front right side sensors (IR7 & IR6) have been used to check whether there is an obstacle on the right side of the robot. It will check the (IR5) sensor measurement from the obstacle and continues rotating till the distance remains 1cm.
The following figures show the strategy for perpendicular rotation.

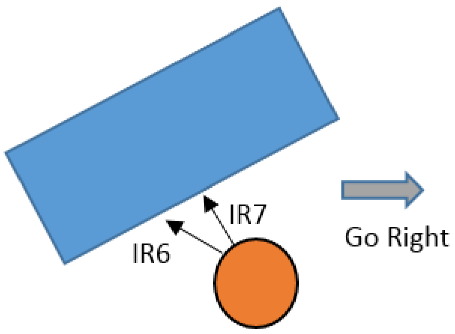


Figure 4: Perpendicular rotation to right

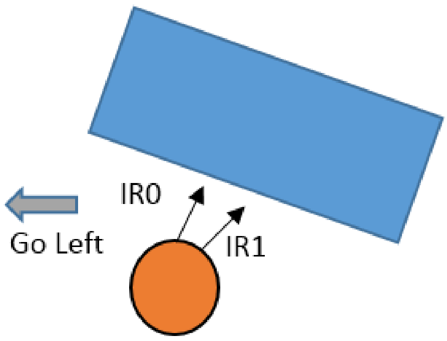


Figure 5: Perpendicular rotation to left

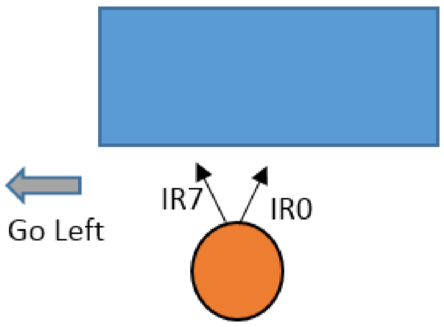


Figure 6: Perpendicular rotation to straight

The MATLAB implementation is attached with the report.

3.2.2. Circumnavigation. Once the robot is aligned to the obstacle, the robot shifts to circumnavigation state. First, the front sensor (IR1) is used to keep constant distance between the obstacle and the robot. If the sensor gives a reading less than a predefined threshold, it indicates that the robot is too close to the obstacle which will set the angular speed to 60° to rotate the robot right. And if the robot is too far from the obstacle then the angular speed to -60° to rotate the robot left. Otherwise go straight with a linear speed 120 and angular speed 0°. This cases are also used for rotating at the corner of the obstacle.
While the robot continues to circumnavigating the obstacle, the distance of the robot to the m-line has always been checked. If the distance is less than 2 cm, the robot shifts away from this obstacle following behavior. Here, to avoid following the m-line from the point of start, or extension of m-line another condition has been used.
The hit-points have been used to check the distance to goal when it re-encounters the m-line, and if it is smaller than the distance to goal from the start point of obstacle following (with some threshold), the obstacle following phase terminates and the robot starts propagating through m-line.

The way of navigating the robot at the corner is shown in figure 7:

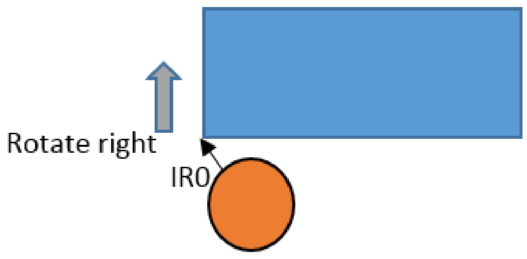


Figure 7: Circumnavigating at the corner

When there is no m-line the robot circumnavigates the whole obstacle. The trajectory is given in figure 8:

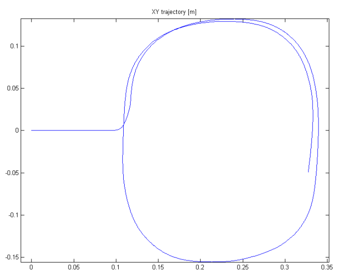
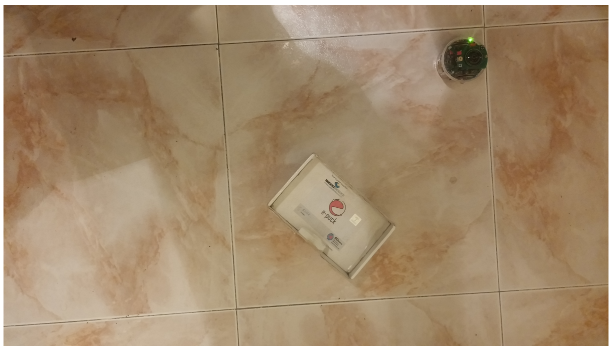


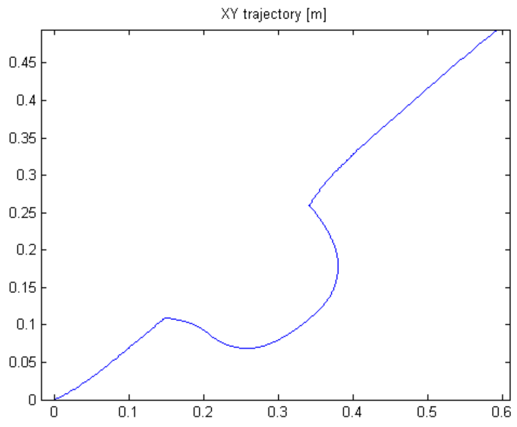
Figure 8: Circumnavigating the obstacle with out m-line

3.3. The complete Bug 2

After implementing the Obstacle following and Head towards goal behaviors, we have combined them to complete the bug 2 algorithm. Always the robot starts with the behavior Head towards goal and when it encounters an obstacle, the head towards goal behavior is deactivated and the obstacle following behavior starts and the robot follows the obstacle. The condition that terminates the obstacle following and trigger the head towards behavior is the distance to m-line. while the robot is following the obstacle, if its distance between the robot and the m-line is less than the threshold given, the obstacle following behavior stops and the head towards goal restarted. The results are shown in Figure 9 and 10 for different environments.

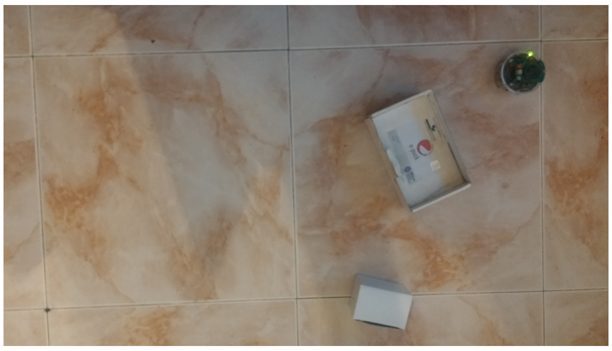


(a)

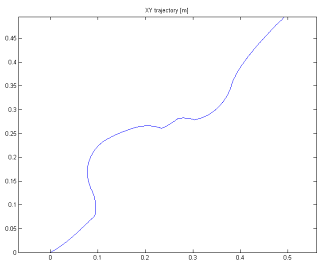


(b)

Figure 9: a)Environment with one obstacle b)trajectory following one obstacles,



(a)



(b)

Figure 10: a)Environment with two obstacle b)trajectory following two obstacles

4. Conclusions

Bug 2 algorithm is an efficient path planning algorithm that lets the robot find the goal position in the presence of obstacles in a systematic way. This algorithm is better than the Bug 0 and Bug 1 algorithm. It has lesser chance to get trapped and better chance to reach the goal without losing much time by circumnavigating whole object.

In this lab work we implement Bug 2 algorithm. The results show that the algorithm performs well under the tested conditions. One drawback of the algorithm is depending on the map it can perform less efficient, in such scenario Bug 1 performs better than Bug 2.

References

[1] J. A. Oroko, and G.A Nyakoe, Obstacle Avoidance And Path Planning Schemes for Autonomous Navigation of a Mobile robots

[2] Buniyamin N., Wan Ngah W.A.J., Sariff N., Mohamad Z. A Simple Local Path Planning Algorithm for Autonomous Mobile Robots