Autonomous Robotics Lab 3 Report

Master in Computer Vision

UNIVERSITE DE BOURGOGNE

Centre Universitaire Condorcet - UB, Le Creusot

01 May 2017

Submitted to: Prof. Xevi Cufí

Submitted by: Mohit Kumar Ahuja

Marc Blanchon

Aim:

- To implement any of the bug algorithms.

We chose to implement BUG 0 algorithm.

BUG 0 ALGORITHM: As shown in figure 1, we have to set a goal anywhere in the plane and there might be or might not be obstacles in the path. And the Robot have to first identify the Goal in comparative to his own location and then start moving towards it and it should turn according to the angle of rotation required for facing towards goal.

If there is any obstacle in between robot and the goal, The Robot should follow the obstacle till he see its goal again (or till the obstacle is finished). The robot should calculate the angle of rotation again to face towards the goal and then start moving towards it and once the robot reaches the destination (Goal) it should stop (As shown in figure 2.

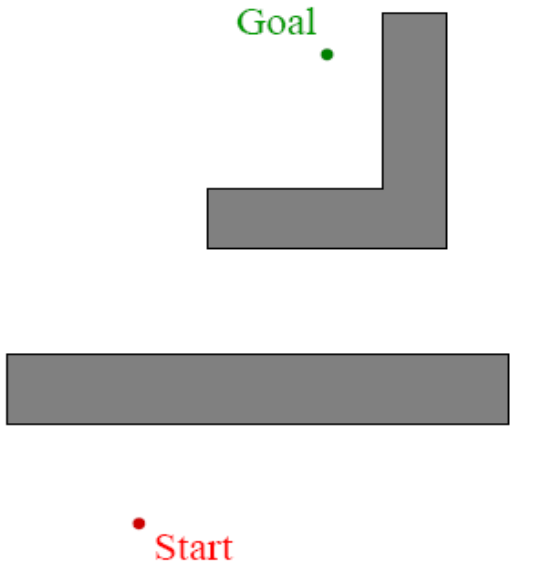

Figure 1

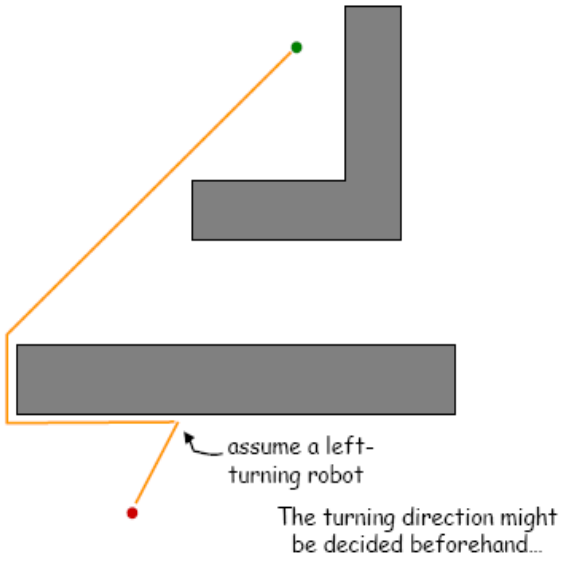

Figure 2

Implementation: To implement this algorithm, we made a function of Wall_follower in which we used the wall following technique which is being explained in project 1 report. And we also reused the update_odometry function to update the {x,y} coordinates and the angle of rotation for robot.

About The Code:

It's very important first to know about the code before doing any experiment to the code. So, here are some points to know about the code in details:

- Open webot software and choose "your Project" among four options. Now click on open world icon and in the appeared Windows open folder Project_3 (attached in mail).
- In Project_3, open worlds and select "e-puck" file. A world will appear on the screen with editor and console.
- In the starting of the code, we defined some global variable like sensor_value of sensor's 0,1,4,5,6,7 and some other variables. Because these variables were used in main as well as in functions.
- Then we defined our GOAL { X , Y } coordinates globally.
- Then we defined functions,

  void wall_follower();
  void update_odometry();

- And in the main program, after defining some variables and initializing Webots, we initialized and enabled distance sensors and got values from distance sensors.
- In the very first "if statement", we stored the square root value of distance of goal from robot:

  store_value = sqrt((Goal[0] - d[0])(Goal[0] - d[0])+(Goal[1] - d[1])*(Goal[1] - d[1]));*

  Where, Goal[0] = x coordinate of goal and Goal[1] = y coordinate of goal
  And, d[0] = x coordinate of robot and d[1] = y coordinate of robot

  And then we calculated the distance and the angle of rotation. So, if the goal lies on the left side of the robot it will start moving towards left side. And if the goal lies on the right side of the robot, it will start moving towards right side of the robot. The same is applicable for the back left and back right side of the robot, it will start moving forward and then the robot will slowly rotate 180 degree and will start facing towards goal.

  The angle between the robot and the goal is being calculated by the formula mentioned below:

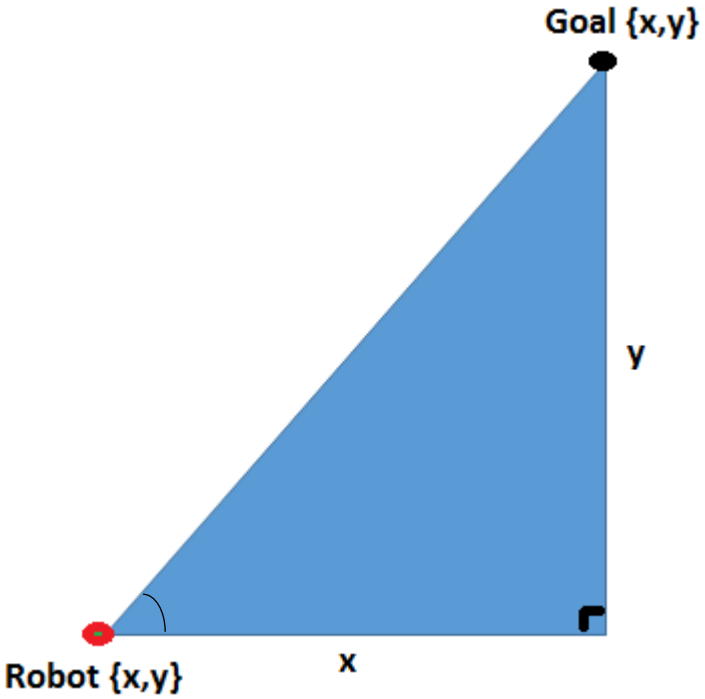

Figure 3

$$Angle = \text{atan}(y/x)$$

By doing this the robot will follow the minimum distance path. If any one of the sensors 0,1,6,7,5 detect obstacle ie sensor_value>500 it will jump into wall_follower function and will start following obstacle but will simultaneously update the {x,y} coordinates too. And as soon as the wall (or obstacle) has passed, it will go straight till the value of back sensor reduces from 400(This is because sometimes using only sensor 5 will follow wall but at the edges of obstacles, the robot starts hitting the edges which leads to maximizing errors). So we used sensor 4 too, to check if the wall has passed completely or not.

And when the obstacle is completely passed, it again calculate the minimum distance between current location and the goal coordinates and then calculate angle simultaneously and start rotating towards goal.

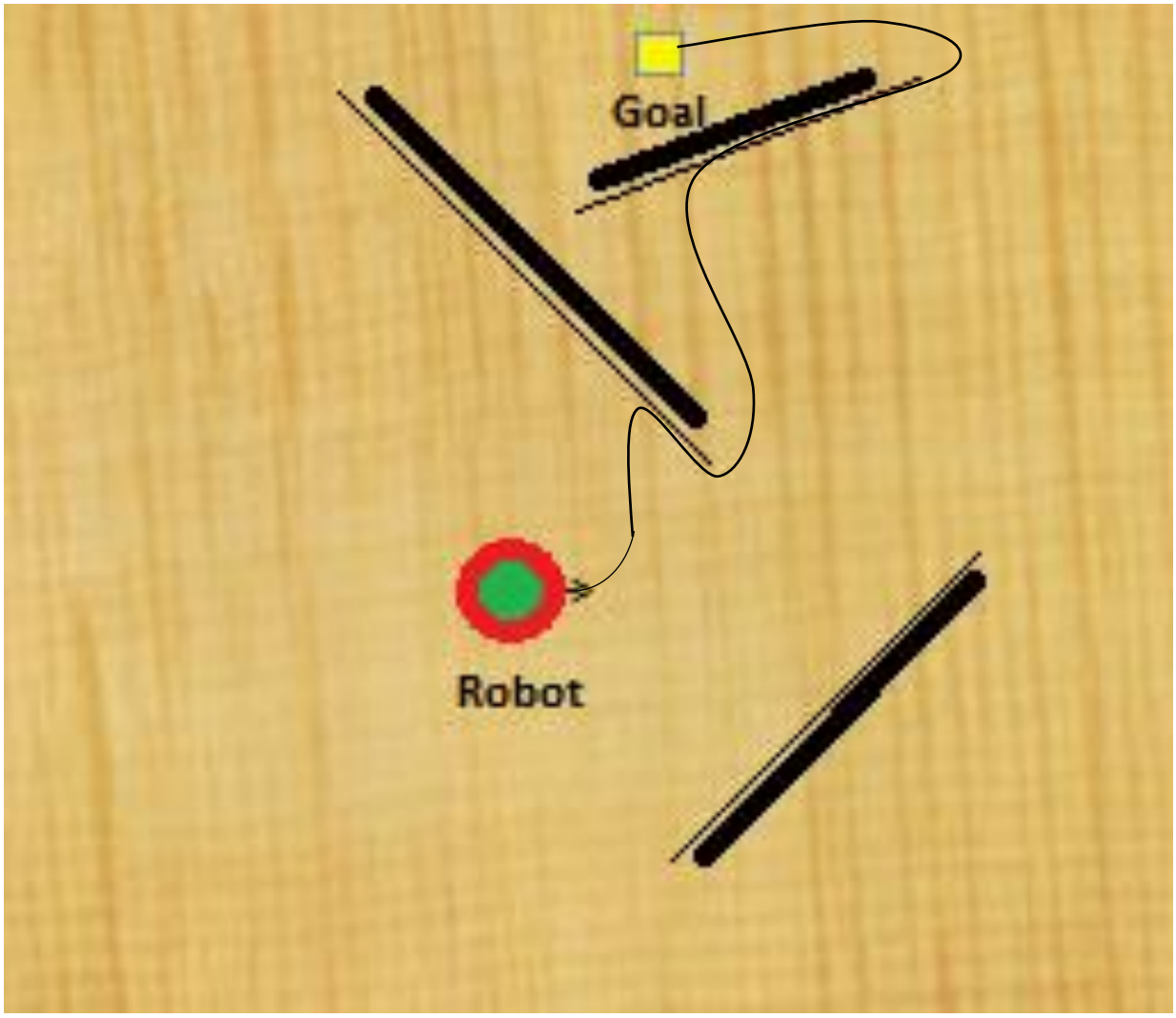

Figure 5

As shown in Figure 5, the robot will calculate the minimum distance to goal and will start following that distance, but as soon as the robot detects wall, it will follow that wall (or obstacle) till it finishes and then again calculate the minimum distance to goal and angle of rotation and start moving towards robot. The Thin black line in the picture is the path which will be followed by the robot in simulator.

As explained earlier, to detect wall and follow wall we used the wall_follower function which is explained in details in report 1. And to update the {x,y} coordinates of the robot, we used update_odometry function which is being explained in details in report 2. But still I am explaining both the functions again

{If you have read those functions before please skip the function explanation}

Note: If you have read Wall_follower functions in previous report please skip the function explanation.

Wall_follower function:

In this function, we first declared device tag variables for storing robot sensor values. And then enabled all sensors we want to use and after enabling we get values from sensors. And then we implemented seven "if conditions" for wall following. That is explained in details below:

1. In First "if conditions", if the sensor value of any one of 0,1,6,7 sensors exceeds 500 AND if sensor value of sensor 5 is less than 500 it will turn right. Which means robot is either facing towards wall or inclined towards it. The syntax is shown below:

if(((sensors_value_0>=500)||(sensors_value_1>=500)||(sensors_value_6>=600)||(sensors_value_7>=500)) && (sensors_value_5<=500))

2. In Second "if condition", now the robot is turned, and the sensor value of sensor 5 will toggle from less than 500 to greater than 500 as the left side of the robot will face towards wall. The syntax is shown below:

elseif(((sensors_value_0>=500)||(sensors_value_1>=500)||(sensors_value_6>=600)||(sensors_value_7>=500)) && (sensors_value_5>=500))

   So if the sensor value of any one of 0,1,6,7 sensors exceeds 500 AND if sensor value of sensor 5 is greater than 500 it will continue turning right till the sensor value of any one of 0,1,6,7 sensors decreases from 500.

3. In third "if condition", till the values of all the sensors 0,1 AND sensors 6,7 is less than 500 and the value of sensor 5 ranges from 500 to 550 OR sensor 4 should be greater than it will go straight so in this condition, the robot is maintaining a particular distance of 1 cm from wall. As shown in fig 1. The syntax is shown below:

elseif((((sensors_value_0<=500)||(sensors_value_1<=500))&&((sensors_value_6<=500)||(sensors_value_7<=500))) && ((sensors_value_5>500)||(sensors_value_4>500)))

4. In forth “if condition”, till the values of all the sensors 0,1 AND sensors 6,7 is less than 500 and the value of sensor 5 ranges from 400 to 500 which means the robot is going far from the wall, so it will take slight left turn and will come in the third if condition to maintain 1 cm distance from wall. The syntax is shown below:

```
elseif((((sensors_value_0<=500)||(sensors_value_1<=500))&&((sensors_value_6<=500)||(sen
sors_value_7<=500))) && (500>sensors_value_5 && sensors_value_5>400))
```

5. In fifth “if condition”, till the values of all the sensors 0,1 AND sensors 6,7 is less than 500 and the value of sensor 5 ranges from 550 to 600 which means the robot is going near to the wall, so it will take slight right turn and will come in the third if condition to maintain 1 cm distance from wall. As shown in fig 1. The syntax is shown below:

```
elseif((((sensors_value_0<=500)||(sensors_value_1<=500))&&((sensors_value_6<=500)||(sen
sors_value_7<=500))) && (600>sensors_value_5 && sensors_value_5>550))
```

6. In Sixth condition which is now “else condition”, if none of the if conditions is being executed which means that the wall is not being detected yet, the robot will move forward.

Note: If you have read update_odometry functions in previous report please skip the function explanation.

Update_odometry function:

Odometry is the use of data from the movement of actuators to estimate change in position over time. In the case of wheeled robots, from the movement of each wheel. The change in position will depend on the rotation of the wheels, on its radius and the width between them. In the starting of every code, the initial positions are being set to zero. You can find this in code:

```
double d[3] = {0.0,0.0,0.0};
```

And for odometry we made a function update_odometry in which as the robot moves the {x,y} coordinated of robot gets updated automatically.

```
double da = (((dr - dl))/ AXLE_LENGTH); // delta orientation // axle_length = width
double dis = ((dl+dr)/2); // mean distance of both tires
disp += ((dl+dr)/2); // mean distance of both tires stored in another variable
d[0] += dis*cos(d[2]); // calculate and update x coordinate
d[1] += dis*sin(d[2]) // calculate and update y coordinate
d[2] += da; // calculate and update angle
```

So, in this function, we calculate the delta orientation by dividing the distance of both wheels by width of wheel. And the mean distance “dis” and the {x,y} coordinates are being updated by multiplying dis with cos ⍉ and by sin ⍉. And the {x,y} coordinates are being updated in every iteration.

Youtube Link’s: For better understanding of wall follower program, how it runs in simulation and in real world, I made a video and you can find the links below:

1. Simulation Link: https://youtu.be/PSEXqcLw2Ig