Master in computer VIsion and roBOTics

Report on

Bug 2 Algorithm with The e-puck Mobile Robot

Autonomous Robotics Lab number 1

Made by:

Chalikonda Prahbu Kumar (Chakon)

Viktor Stefanovski

Mentors:

Marc Carreras Perez

4/07/2013

Contents

1. Introduction

This coursework is based on familiarization with the basic functions, structure and programmability requirements of the e-puck mobile robot.

E-puck represents small differential wheeled mobile robot which possesses some extra convenient characteristics like flexibility, compact design and robustness, which alongside its affordability make it the extremely appealing educational tool.

E-puck is well equipped device, containing battery, motors, Bluetooth and IR sensors, just to name a few supporting devices.

The abovementioned devices are the ones utilized for execution of one of the basic control architecture – Bug 2 algorithm, being exactly the objective of the successful completion of this coursework.

2. Overview

Bug 2 algorithm is member of the family of bug algorithms. Not in all, but in the most cases it beats the performances of its "siblings". It is popularly called greedy algorithm because it takes the first thing that appears better without exploring all possibilities.

There are two behaviors mutually intertwined in order for the ultimate objective of this algorithm to be achieved. Those two behaviors are the head-towards-goal behavior and obstacle-following behavior.

The head-towards-goal behavior is activated at the beginning of the algorithm and stays active unless an obstacle is found along the trajectory of movement to the specified goal position. When this situation occurs (obstacle detection), the behavior is switched to obstacle-following.

This obstacle-following behavior remains active until the m-line, initial trajectory of movement from start to goal position, is met after circumnavigating the obstacle and only if the distance between the point where the m-line is again encountered and the goal position is smaller in comparison to the distance between the goal and the initial point where the obstacle-following behavior is activated.

One of the key systems for the execution of this algorithm is the e-puck Infrared sensor composition. This composition encompasses eight Infrared sensors equally distributed throughout the robots surface, or more precisely two frontal sensors, two rear sensors and four peripherally placed by pairs of twos on each side, all in all contributing to proper distribution, leading to full, robust and correct object (or in our case obstacle) detection.

Many factors can influence the sensor detection process like the surface of the object and the ambient light, which leads to the necessity of sensor calibration prior to the actual implementation process.

3. Design and Implementation

This phase starts with carefully analyzing the algorithm requirements, planning and developing the separate stages of implementation, actual implementation of each and every part of the algorithm and finally establishing a compact solution incorporating all the integral parts.

The design and implementation process primary programmability state is the sensor calibration process, followed by implementing the obstacle-following behavior and the head-towards-goal behavior and ends with the incorporation of these two behaviors using the switching between behaviors process conditioned by the obstacle detection and distance to m-line measures, which is actually the bug 2 algorithm itself.

3.1. Sensor Calibration

As mentioned previously the main reason for checking and calibrating the sensors of the robot are the external influences such as the ambient light, surface of the material sensed as well as the distance between the sensors and the actual object.

Given all of these reasons is enough of a proof why precisely the sensor calibration can play major role in whatever situation and setting the experiments are undertaken, because given the different external variables of every environment of experimental conduction the experimental results may prove not to match the ones that are expected.

When it comes to the coding process, the modifications need to be made using the low level controller with negative linear speed and angular speed of zero, indicating a backward movement of the robot, accompanied with call to the *read sensors* and *compute odometry* functions, and storing the results in new variables *readsen* and *distan* – arrays initially set as empty and then updated with the results of the first IR sensor and the absolute value of the global variable X. All this translated into matlab language would appear as this addition to the main program function:

```
global IR
global X
readsen=[];
distan=[];
lowLevelController(-10,0);
readSensors();
computeOdometry();
readsen=[readsen;IR(1,1)];
distan=[distan;abs(X)];
```

Obtained values of the calibration can be used for updating the coefficients of the sensors but only if they diverge a great deal from the coefficients initially given.

On the next figure we can observe a plot of the experimental results of the initial sensor calibration against its 7th degree polynomial approximation.

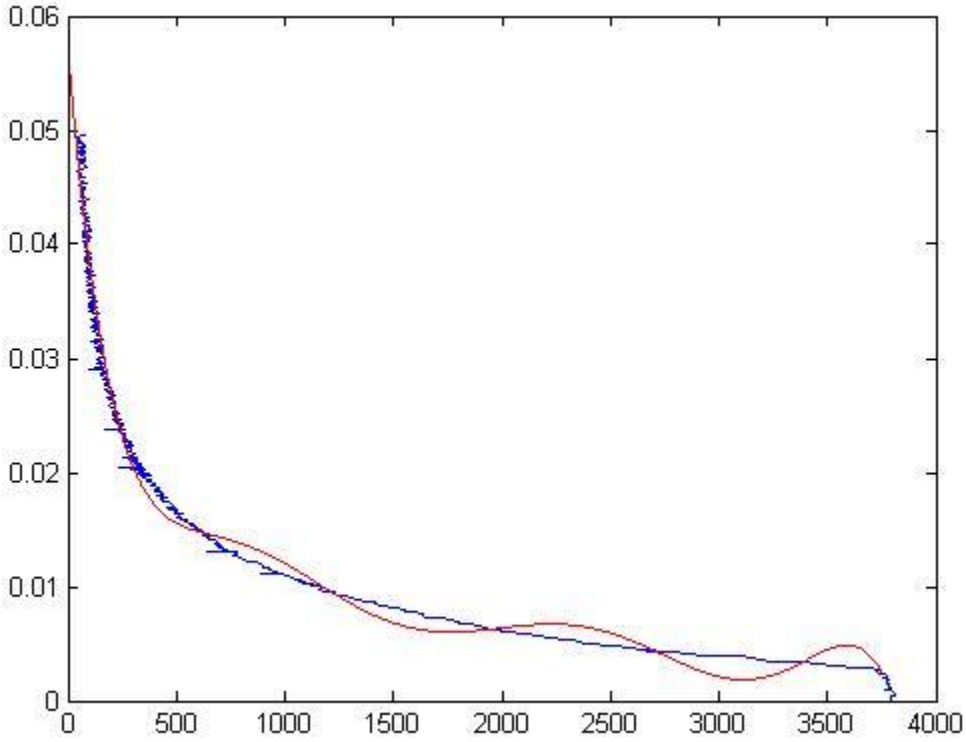

Figure 1 Sensor readings against their 7th degree polynomial approximation

3.2. Obstacle following behavior

Implementation of the obstacle following behavior is one of the main focuses of the code building process and without doubt the most extensive (sized) part. In other words it represents the creation of a high-level controller function.

On a personal scale it represents yet another switching structure between three separate states, or what we call *stage*(s) in the coding counterpart.

The first state (*stage*=0) is the obstacle detection state where all the frontal sensors values are inspected whether their value is smaller than a certain threshold value or as called *dis*, it being set to 2. If this condition is met the state switching takes place taking the function in the state 1. Alternative to the fulfillment of the condition is straight movement of the robot with linear speed (*linearTel* equal to 80.0).

The second state, or *stage*=1 performs rotation of the robot with angular speed (*angluarTel*) = 100.0 until the robot becomes perpendicular to the obstacle, inspected by the first three sensors (IR0, IR1 and IR2), which indicates transition to the third state.

Stage 3, as called in the code, performs examinations whether the robot is too near of too far from the obstacle, and makes the necessary adjustments or correction of the position of the robot. This is done with the help of predefined slope value *m*=10 and three angles *l1, l2 and l3* which provide the necessary smooth or sharp turning at the corners. Also, if the robot is too near to the obstacle while moving parallel to its position a correction step of angular speed of 20.0 is used. Next thing that is examined is whether the front two sensors – *IR0 and IR7* detect any new obstacle, and if this proves to be true the robot goes back to stage 1, being perpendicular to the new obstacle. Lastly, we condition the loosing of the obstacle, which is the case when the distance registered through the sensors on the left side exceed certain value (i.e. 3.5) which results in switching back to the stage 0 and moving straight until a new obstacle is detected. This is the corresponding matlab segment of the obstacle-following behavior (function):

```
function [linearTel,angularTel]=highlevelcontroller()
  global go
  global IR
  global stage
  % Threshold value
    dis=2;
    switch (stage)
        case 0
            % If sensors detect an obstacle in front of the robot change
            % stage to 1
            if(IR(1,1)<dis || IR(1,2)<dis || IR(1,7)<dis || IR(1,8)<dis)
                stage = 1;
                linearTel =0.0;
                angularTel = 0;
```

```
% Otherwise the robot moves straight with constant speed of 80
            else
                stage = 0;
                linearTel =80.0;
                angularTel = 0;
            end
case 1
            % If the sensors detect that the robot is perpendicular to the
            % obstacle change to stage 2
            if((IR(1,3)<(dis+0.6)) && (IR(1,1)>(dis-0.6) && (IR(1,2)>dis)))
                stage = 2;
                linearTel=0.0;
                angularTel=0.0;
            % The alternative is spinning (rotating) with angular speed of
            % 100 until the robot becomes perpendicular to the obstacle
            else
                stage = 1;
                linearTel=0.0;
                angularTel=100.0;
            end
case 2
            % In the case of the object getting near to the obstacle
            if(IR(1,3)<(dis+0.5))
                % Definition of slope and parameters (angles) for correction
                m=10;   % slope
                l1=5;   % angles
                l2=45;
                l3=10;
                % Finding the sensor that has minimum distance to the obstacle
                diff=IR(1,2)-IR(1,3);
                min_sen=0;
                if(IR(1,2)>IR(1,3))
                    min_sen=IR(1,3);
                else
                    min_sen=IR(1,2);
                end
                % If distance is too big (the robot is losing the obstacle)
                % then the robot is at right corner
                if(diff>1)
                    % If the robot is too near to the corner turning smoothly
                    if(min_sen<dis-0.5)
                        linearTel=80.0;
                        angularTel=-((m*min_sen)+l1);   % smooth turn at corner
                    % If it is far sharp turn
                    elseif(min_sen>dis)
                        linearTel=90.0;
                        angularTel=-((m*min_sen)+l2);   % hard (sharp) turn at
corner
                    else
                        linearTel=100.0;
                        angularTel=-((m*min_sen)+l3);
                    end
                % If the robot is too near to the obstacle correct the position
                elseif((diff<=1)&&(IR(1,3)<dis))
                        linearTel=80.0;
                        angularTel=20;
```

```
                % In contrary go straight
                else
                        linearTel=80.0;
                        angularTel=0.0;
                end
            % If an object is detected in front of the robot go back to
            % stage 1 being perpendicular and with free front range
            elseif((IR(1,1)<dis) || (IR(1,8)<dis))
                        stage = 1;
                        linearTel=0.0;
                        angularTel=0.0;
            % If the distance becomes too high stop following the obstacle
            % and go back to stage 0 moving with constant speed straight
            else(IR(1,2)>=3.5&& IR(1,3)>=3.5)
                        stage =0;
                        linearTel=80.0;
                        angularTel=0.0;
            end
    end
end
```

In the following figure we have a obstacle following trajectory.

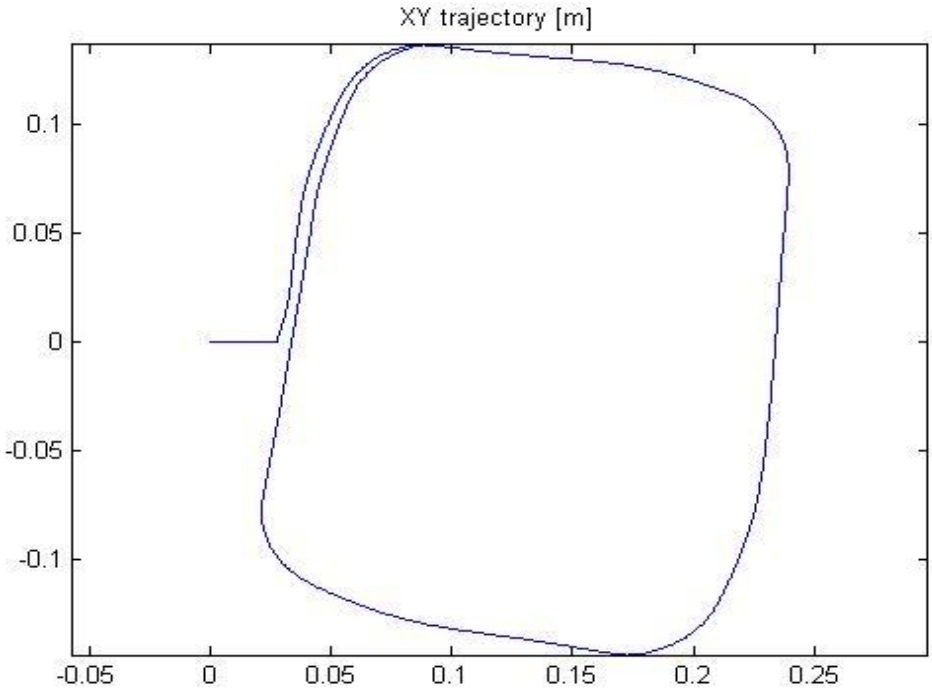

Figure 2 Obstacle-following behavior

3.3. Head towards goal behavior

Head-towards-goal behavior is the second integral part equally as important as the previously described obstacle-following behavior needed for building and performing the bug 2 algorithm.

Here, the main parameters are the distances and angles variables. After the usual variable definition the Euclidean distance between the current position of the robot and the goal position is calculated. But for the successful calculation to be made the program needs to have a predefined goal position. This is asked from the user to be inserted manually with the input commands in the main program function.

After that, the second main value that needs to be calculated is the delta angle, being the angle between the X-axis of the robot and the robot goal position. For this purpose the *atan2* function is used.

The main condition set is the reaching of the goal. For achieving this end a threshold value of 0.02 is used in our case for examining if the Euclidean distance is smaller that this value. If the response is positive then that means that the speed (both linear and angular) are set to zero and the *go* is set to 0, indicating that the goal is reached.

The alternative consists in conditioning whether the angle theta, which is the angle between the X-axis of the robot and the direction of the robot, is greater than the sum of delta with a threshold angle (20 rad – converted to degrees with the *rad2deg* function) or is smaller than their difference.

If one of this two conditions is satisfied than the robot turns (rotates) in place to the direction determined by the sign of the delta angle with angular speed of 60.0.

On the other hand, if the conditions are not satisfied the robot starts moving straight with constant linear speed of 100.0.

Here follows the matlab function responsible for the performance of the head-towards-goal behavior:

```
function HeadTowardsGoal()
global X
global Y
global Theta
global Delta
global Goalx
global Goaly
global linearTel
global angularTel
global go
Thresh_angle = 20;      goal_thresh = 0.02;
% Calculating Euclidian distance to goal
Eucl = sqrt((Goalx - X)^2+(Goaly - Y)^2); slope=Goaly/Goalx;
% Calculating the angle delta
Delta = atan2(Goaly-Y,Goalx-X);     disp('Delta');
% If the Euclidean distance is smaller than a threshold ->Stop, goal is reached
if(Eucl<goal_thresh)
    disp('Goal'); go=0;
    linearTel =0.0;
    angularTel =0.0;
% Otherwise if the goal is not reached
else
    % If the robot is not facing the right way, rotate until it does
    if((Theta>(Delta+(deg2rad(Thresh_angle)))
||(Theta<(Delta-(deg2rad(Thresh_angle))))))
    angularTel = sign(Delta)*60.0;
    linearTel = 0.0;
    % Go straight with constant speed of 100
    else
    linearTel=100.0;
    angularTel=0;
    end
end
end
```

And here are the trajectories of this behavior when the user gives as goal position the coordinates (40,20), (40,0) and (0,40) respectively.

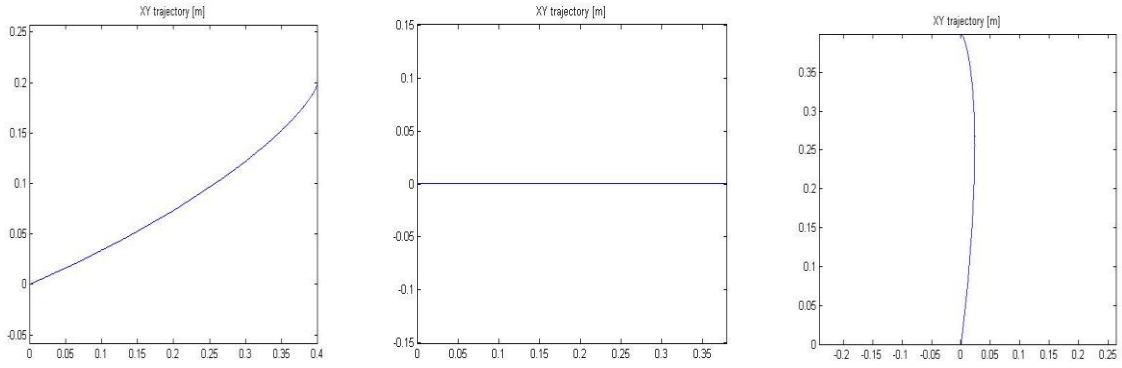

Figure 3 Head-towards-goal behavior with goal positioned at (40,20), (40,0) and (0,40)

3.4. Bug 2 algorithm

Bug 2 algorithm represents only mixing between the two previously described behaviors, with the conditioning of: whenever the Euclidean distance between the first point where the obstacle-following starts and the goal position is greater than the Euclidean distance between the goal and the point where the robot senses again the m-line. And for sensing the m-line again, a threshold value of 0.02 is used.

The only problem that we are experiencing is the robot not sensing the m-line and losing the obstacle on corners and reaching the goal position that we manually input.

Here is one experimental result obtained with the usage of the bug 2 algorithm:

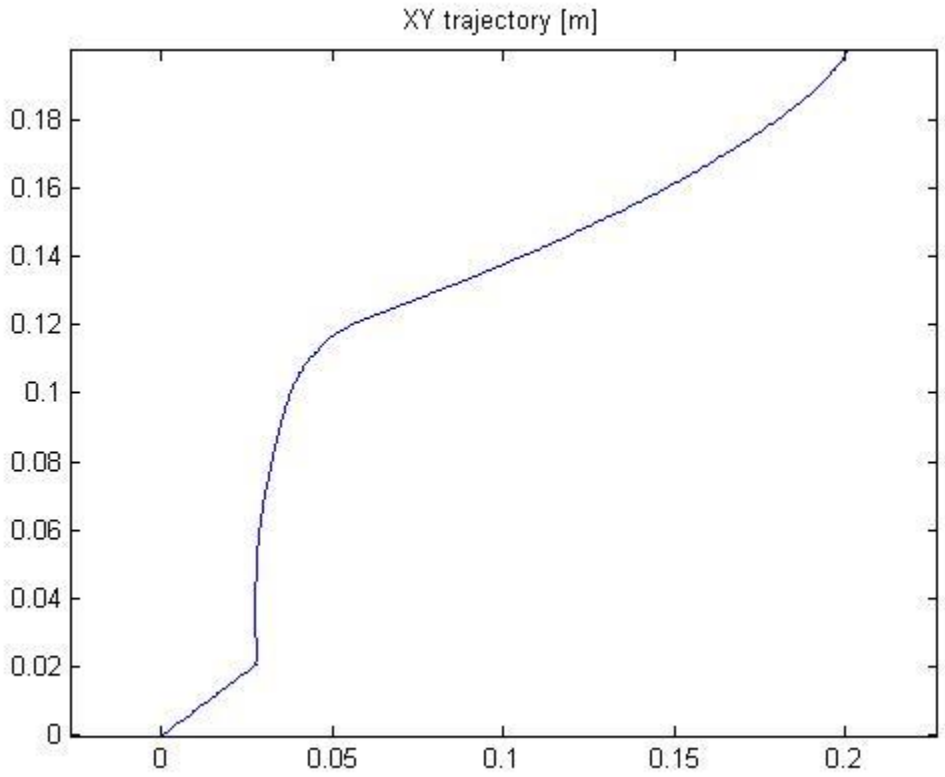

Figure 4 Bug 2 algorithm atempt – goal position (20,20)

4. Discussion and Problems Encountered

Undertaking this task was a great challenge and as expected it came with a lot of obstacles and difficulties during its execution. For starters we learnt how to be patient and composed with the inability or running the code for two consecutive times without experiencing matlab problems and having to close and open again the program, as well as bad Bluetooth connection problems (even though it shows connected, but the connection is not working well we check with comport it constant always), which might have cost us half a day. Finally, we discovered that we can us the time variable in the while loop in the main program and give the exact time of execution of the program, which proved to be sworn with two blades at times.

Also we have to place this coursework at the top three projects (it seems almost like an entire project) considering the level of excitement and thrill of performance, but on the flip side it is one of the top three by the level of difficulty in the masters program.

The initial position of the robot can play crucial role in defining the trajectory of movement.

Another problem we encountered was the passing of the x and y coordinates of the goal position. Initially we used full numbers but when the results did not come as expected we realized that by giving full numbers actually we pass meters, and since then used decimal values (i.e. 0,20 = 20cm).

5. Conclusion

Working and dealing with the difficulties which were presented by the enforcement of the task itself was invaluably beneficial and at the same time source of new skills and knowledge in the state of robotics art.

Among these freshly obtained skills are improving of teamwork and cooperation abilities, learning how to connect to primarily and then program e-puck as one of the representatives of the mobile robot family to follow commands - be manipulated to head toward a specified position and even follow or circumnavigate given obstacle blocking its final destination, then take on, break and switch between different behaviors, how can the art of sensor calibration be important part of the metrics and results obtained as well as arguable the most valuable thing obtained which is actual firsthand experience with real robotic device.

6. References

http://en.wikipedia.org/wiki/E-puck_mobile_robot

http://www.cs.jhu.edu/~hager/Teaching/cs336/Notes/Chap2-Bug-Alg.pdf

Course slides and lab coursework