Bug 0 Algorithm
Lab Work (P2) for Autonomous Robotics
Master Computer Vision and Robotics (CVR)
Universite De Bourgogne, Le Creusot Antenna

Pamir Ghimire, GopiKrishna Erabati

April 28, 2017

1 Summary:

Bug 0 is a reactive-navigation algorithm that uses odometry. The robot starts with knowledge of the position of goal relative to its initial pose, but no knowledge of the environment. The strategy is then to use odometry to orient and move toward goal while reactively avoiding obstacles.

2 Introduction:

The Bug 0 algorithm can be stated as follows: [1]
1. Head toward goal
2. If can not head toward goal, follow obstacle until you can head toward goal again
3. Continue

The algorithm can be seen as putting the robot in different behavioral modes based on the position of the goal relative to itself and the state of its immediate surroundings. The different behaviors can be 'orient toward goal', 'follow obstacle', etc.

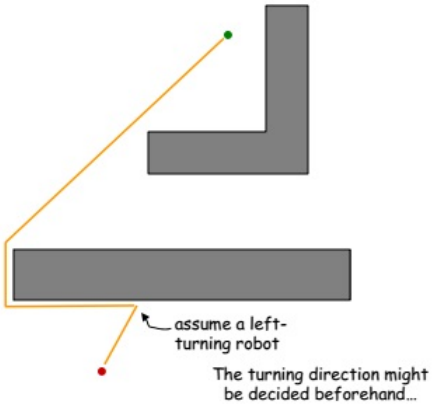

Figure 1: Illustration of execution of bug0. [2]

To maintain the knowledge of self's position and orientation, the robot uses odometry. Concretely, a differential drive robot like e-puck depends exclusively on information from right and left wheel encoders and uses the following equations:

$\triangle D_{LWh} = \frac{\triangle E_{LWh}}{E_{Res}}.R_{LWh}$ change in displacement of right wheel

$\triangle D_{RWh} = \frac{\triangle E_{RWh}}{E_{Res}}.R_{RWh}$ change in displacement of left wheel

$\triangle D_{Robot} = \frac{D_{LWh}+D_{RWh}}{2}$ change in displacement of the robot

$\triangle x = D_{Robot} cos(\theta)$ change in displacement along X-axis

$\triangle y = D_{Robot} sin(\theta)$ change in displacement along Y-axis

$\triangle \theta = \frac{\triangle D_{RWh} - \triangle D_{LWh}}{D_{Axel}}$ change in angle CCW relative to positive X-axis.

D_{Axel} is the length of axel joining the left and right wheels of a differential drive robot. E_{Rw} and E_{Lw} are encoder read outs from right and left wheels respectively. E_{Res} is the resolution of the encoder in ticks/revolution per 2π.

3 Method:

We assigned a 3-element state vector to our robots that captured the displacement of the robot in orthogonal X-Y directions and orientation of the robot with respect to the X-axis. The starting pose of the robot was chosen as $[x, y, \theta] = [0, 0, 0]$. Angles were measured in radians, positive angles being counter clockwise with respect to the positive x-axis.

We translated the Bug 0 algorithm to the following strategy of switching between behaviors:

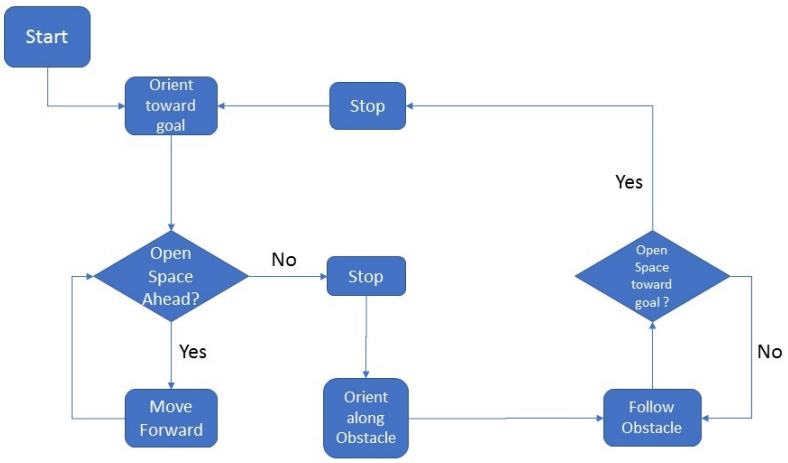

Figure 2: Behavior transition diagram for implementing Bug0.

Deviation from the Diagram: We replaced the decision block 'open space ahead ?' with 'wall present to right ?'. The Bug 0 algorithm requires a robot that is following an obstacle to stop doing so when it can sense open space in direction of the goal, orient itself toward goal and then pursue it. This, however, proved problematic to implement on the real e-puck.

The Bug 0 algorithm assumes a perfect 'sensor skirt', i.e, an area around the robot that can be sensed with excellent accuracy and **resolution**. This does not hold for e-puck since it has only 8 distance sensors around its body, and the distance sensors sense in a conical area (cone beam of IR led), not along narrow straight direction. Further, the angular separation between sensors ranges from 30 to 60 degrees. This, combined with inaccuracy in orientation of angle of the robot when computed using odometry makes it difficult to select the sensor facing the goal. Concretely, the error in robot's orientation is in order of the angular separation of the sensors.

We also discovered that a problem arises because of non-zero size of the robot; in designing the bug 0 algorithm, the robot is thought of as a point. For this reason, we also implemented a 'clear obstacle' behavior that was executed when a robot could no longer find a wall to its right. Because this implied open space to the right of the robot, we programmed the 'clear obstacle' behavior to move the robot slowly in an arc toward its right.

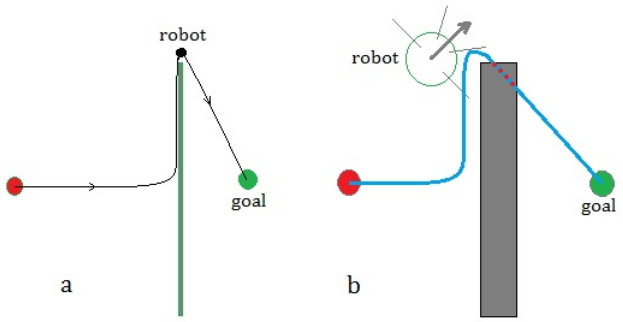

Figure 3: Necessity for a 'clear obstacle' behavior due to non-zero size of robot.

We used the following schemes to implement our behaviors:
+ For turning a certain angle:
⋆ Compute target angle as current angle + desired change
⋆ Turn ccw until current angle is close to target angle

+For following wall: [3]
Minimize an error term which signifies distance from and orientation along wall being followed using a PID controller; Our gains $k_p = 0.03, k_d = 0.14, k_i = 0.11$.

$$e(t) = (kD_2(t) + (1-k)D_0(t)) - D_w, \quad k \in \mathcal{R}^1,\ 0 < k < 1 \qquad (1)$$
$$v_{rw} = v_f \qquad (2)$$

$$v_{lw} = v_f + k_p.e(t) + k_d.\frac{\partial e(t)}{\partial t} + k_i.\int_0^t e(t) \qquad (3)$$

4 Experiments and Results:

Simulation videos illustrating our implementation of Bug 0 on e-puck in Webots can be found at :
Web address 1: https://www.youtube.com/watch?v=-hTFp8UpcRo
Web address 2: https://youtu.be/R6BD9uD3BnU

We notice in the simulation videos that the robot starts moving in a clockwise arc when it can no longer find a wall to its right. This is the 'clear obstacle' behavior of the robot.

It can be noticed in the simulation that the wall following behavior is not robust. This is intentional since we would like the robot to stop following wall whenever the wall curves sharply (as opposed to when open space is available toward goal), try to clear the wall, then pursue the goal direction.

We also noted that greater the distance of the goal from the robot is and higher the number of obstacles on the path, the more inaccurate is the final position of the robot. This is because of odometry errors accumulating over a longer path of the robot.

We frequently faced problems due to situations similar to one illustrated in the figure below. This, again, has to do with non-zero size of the real robot (the real robot is not a point) and the robot not being a tactile robot (it does not touch the wall when it follows it).

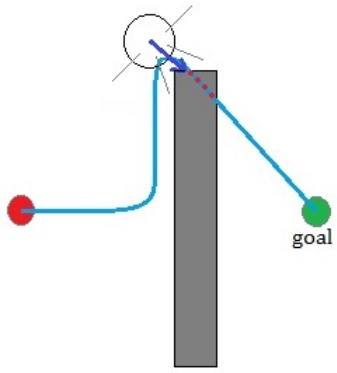

Figure 4: E-puck running into a corner due to the corner being in its 'blind spot'.

5 Conclusion:

We conclude that in presence of robust odometry and accurate sensor skirt around the robot, the Bug 0 algorithm can be a useful strategy to navigate a robot reactively to goal. The implementation becomes tractable when we design the 'go to goal' strategy as a composite of smaller and simpler behaviors.

However, on a robot like the e-puck, the main challenges arise due to:
1. Monotonically increasing error in odometry as the robot moves
2. Imperfect sensor skirt of the robot; the infrared proximity sensors are very sensitive to environmental conditions
3. Robot not being a 'point robot', which is an assumption in design of the Bug 0, as well as invalid assumption of tactile sensors.

Odometry easily fails when the robots wheels slide due a variety of reasons. Also, odometry errors are large when speed of the robot is big, so moving the robot slowly helps to track it more accurately using odometry

Also, the bug 0 can easily fail when the configuration of obstacles is tricky like in the figure below:

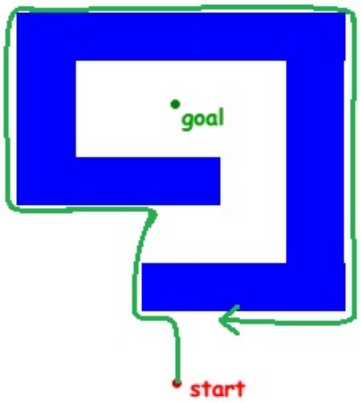

Figure 5: Bug 0 algorithm fails in this map if we decide to always turn left when we come across an obstacle.

Finally, different reactive algorithm, like the bug 2, given the challenges with the real robot, could counter intuitively be less effective since it relies on odometry being more robust.

6 References:

[1] Lecture slides of Dr. Xevi Cufi.

[2] Slides on Robotic motion planning using bug algorithms by Howie Choset, G.D Hager and Z. Dodds

[3] Report for lab P0, Wall Following Behavior on Epuck, Pamir Ghimire, Gopikrishna Erabati