A study on Tangent Bug Algorithm

Kadir Firat Uyanik
KOVAN Research Lab.
Dept. of Computer Eng.
Middle East Technical Univ.
Ankara, Turkey
kadir@ceng.metu.edu.tr

***Abstract*— The ability to traverse/navigate without colliding to the obstacles around is one of most essential requirements for an autonomous mobile robot that is supposed to realize a particular task by itself. Assuming that the robot is provided with it's current global position and orientation with the global goal position, the problem of enabling a robot to reach it's goal position in a resonable amount of time, though seems simple, may present intriguing and difficult issues. Bug algorithms, namely the Bug1 and Bug2 [1], are known to be one of the earliest and simplest sensor-based planners, minimizing the computational burden on the robot while still guaranteeing global convergence to the target if reachable. However, these algorithms do not make the best use of the available sensory data to produce relativly shorter paths by utilizing range data. Although VisBug [2] algorithm uses the range sensor data, it can only utilize this data to find shortcuts that connects points on the trajectory found by Bug2 algorithm containing no obstacles between. In this document, our focus will be on the TangentBug algorithm(TBA) [3]. This algorithm uses range data to produce a local tangent graph so as to choose locally optimal direction while keeping approaching to the target position. TBA produces paths, in reasonably simple environments, that approach the globally optimal path as the sensor's maximal detection range increases.**

I. INTRODUCTION

TBA, similar to the other bug algorithms, assumes that the robot is a point on a 2D planar surface, and the obstacles in the environment are stationary. TBA initializes by getting the goal position as input. It is provided with the range sensor data at each step, and up-to-date position of the robot. Range data is consisted of the directed distance values of the rays emanating from the robot's current position and being scattered in a radial fashion as it is shown in the figure 1. This range sensor is modeled with the raw distance function $\rho = \mathbb{R}^2 \times S^1 \rightarrow \mathbb{R}$ where $\rho(x, \theta)$ is the distance to the closest obstacle along the ray from x at an angle θ. More formally:

$$\rho(x,\theta) = \min_{\lambda \in [0,\infty]} d(x, x + \lambda[cos(\theta), sin(\theta)]^T), \quad \text{where } x + \lambda[cos(\theta), sin(\theta)]^T \in \bigcup_i WO_i \tag{1}$$

For real sensors, sensor readings saturate at the maximum sensing range and it becomes $\rho(x, \theta) < R$ for the points in the range, and R for all the points that are out of range of the sensor.

Having updated the state of the environment and current position of the robot, TBA executes two main actions; move-to-goal and follow-boundary by taking into consideration some heuristics and boundary conditions.

This document is organized as follows: Section 2 gives the overall algorithm, Section 3 briefly presents the experimental framework. Section 4 discuesses about the results obtained during the experiments, and lastly Section 5 concludes with the possible improvements in the heuristics, robot's structure and corresponding changes in the algorithm.

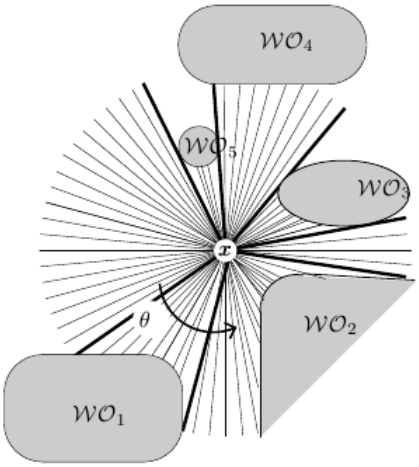

Fig. 1. Range data acquisition of a point robot

II. ALGORITHM

Definitions of the terms and equations that are used in the algorithm are as follows:

x : is the current position of the robot.

q_{goal} : is the goal position.

O_i : is the i^{th} discontinuity point as it is shown in the figure 2.

d : is the estimated distance from the current position to the goal through the local tangent point with the heuristic function $d(x, q_{goal}) = d(x, n) + d(n, q_{goal})$.

d_{reach} : is the output of the heuristic function taking into consideration the best local tangent point.

$d_{followed}$: is the minimum heuristic value calculated while following the boundary.

Here is how TBA works shortly;

1) Move towards the goal. Either directly through obstacle-free space or by following the wall if the

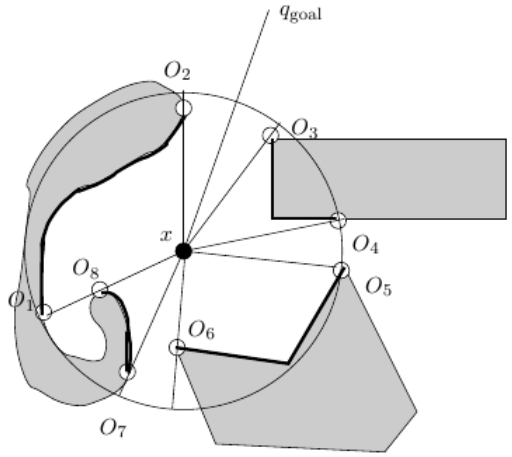

Fig. 2. Discontinuity points are obtained by taking difference between the consecutive sensor reading

distance to the wall is less than a safety measure until the robot get caught in a local minimum. This procedure is a simple gradient-descent in which robot moves in the direction so that it's distance to the goal becomes less and less. If goal is reached, programs exits.

2) Follow the boundary. Get the shortest ray's direction which implies the shortest distance to the wall, and move perpendiculat to this direction which is supposed to be in the similar direction to the latest moving direction of the robot. While following the boundary record the position of the points on the wall being followed, and the minimum of the distances of the points. If d_{reach} is less than $d_{followed}$ robot stops following the boundary and switches back to the state of moving to the goal. However, if robots encounter a point that it has followed before, stops the execution and exits with the *no solution* signal.

III. Experimental Framework

In this study, I have heavily used *Webots* commercial robotics simulator. The robot design via VRML (Virtual reality modeling language) which has three omniwheels enabling robot to translate and rotate at the same time while following a linear motion. I have used range-finder camera model with the configuration of *spherical*, 1° *resolution* and varying *maxRange* values.

Below is the kinematic equation for the robot that is used navigate robot as if it is a point robot but having a specific radius:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} sin(\alpha) & -cos(\alpha) & -r \\ sin(\beta) & cos(\beta) & -r \\ -sin(\beta) & cos(\beta) & -r \\ -sin(\alpha) & -cos(\alpha) & -r \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}$$

Robots wheels are specially designed so that they do not cause so much trobule while moving sideways. Each wheel has 15 rollers and this is pretty much enough to realize smooth movements.

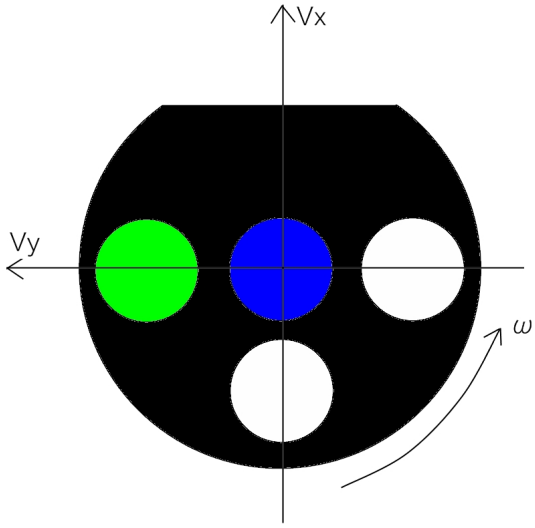

Fig. 3. Robot's reference frame

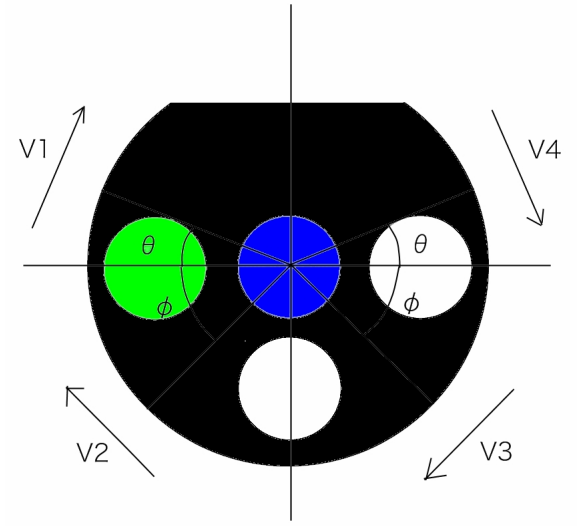

Fig. 4. Wheel enumaration and velocity directions are indicated. The kinematics equation given in this report considers the wheel angles from the vertical line, opposite to the angles given in this figure which are from the horizontal line.

IV. Discussion

TBA algorithm is tested for several scenarios in this study. For the videos of the experiment please visit following websites:

1) Simple move-to-goal behavior, `http://www.youtube.com/watch?v=-4pWHpzQlX0&NR=1`
2) Avoiding local minimum, `http://www.youtube.com/watch?v=JOvpn6uN47o`
3) Avoiding concave obstacle, `http://www.youtube.com/watch?v=62DFlIq5RxI`
4) Reporting no solution, `http://www.youtube.com/watch?v=ZpKmI_hMcog`

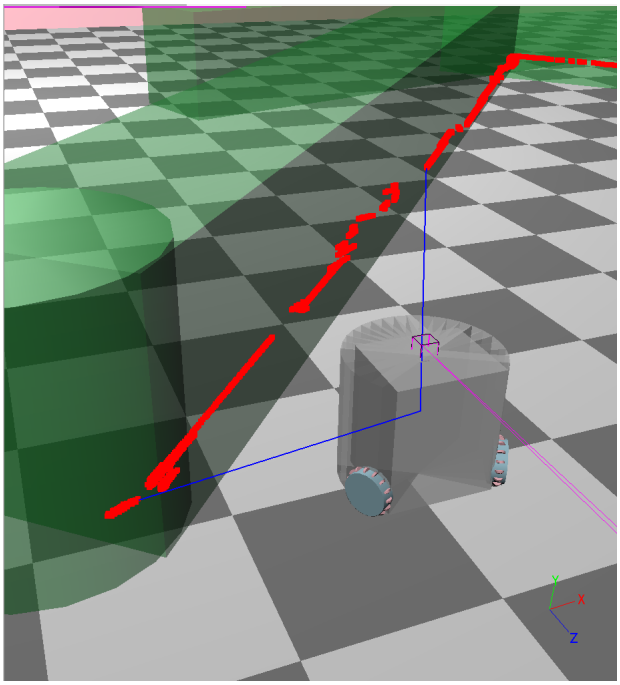

Fig. 5. Simualted robot taking the boundary following action almost sideways.

In the videos, red points indicate the local tanget points and the point that are on the way to the goal. Sometimes, while turning around the rectangular edges, the algorithm may not find local tangents. One can add a hyserisis behavior here to avoid such situations, meaning that robot will keep going although it doesn't find a local tangent to follow, but it will find one as soon as the edge point is passed.

Another issue is simulation time takes a lot, in the order of one fourth of the speed that it is supposed realize. During programming I haven't consider too much about the optimization issues, there might be the cases in which unnecssary ray calculations are performed.

V. CONCLUSIONS AND FUTURE WORK

In the current version of the implementation, I didn't consider the cases when local tangent that is to be followed is not in the direction of the latest velocity command direction. This happens when traversing around the outside of a triangular obstacle and while turning around corners. One can make a change in the implementation so that robot keeps moving to pass the corner and finds a local tangent in the other way around, moves in that direction and continues following the boundary of the triangle more correctly.

Several optimization can be done while ray tracing. Sometimes it is not necessary to trace the rays at the back of the robot with respect to the direction it moves.

As a visualization improvement, in the physics file of the simulation, one can add several openGL drawing functions -instead of one- which discriminates the points on the tangential points, the point that is on the way to the goal, and the points that robots center has passed through. Besides, an object would be added to the goal position to make things more easier to understand for the people watching the demo videos.

VI. ACKNOWLEDGEMENT

All the source code; simulation manager, agent, physics, and simulation design can be found, here `http://kovan.ceng.metu.edu.tr/~kadir/academia/courses/grad/cs548/hmws/hw1/src/`

This document can also be downloaded in pdf format, here `http://kovan.ceng.metu.edu.tr/~kadir/academia/courses/grad/cs548/hmws/hw1/report/tba2.pdf`.

REFERENCES

[1] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. Algoritmica, 2:403-430, 1987

[2] H. Choset and J. W. Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. In IEEE International Conference on Robotics and Automation, Nagoya, Japan, May 1995.

[3] I. Kamon, E. Rivlin, E.Rimon. A new range-sensor based globally convergent navigation algorithm for mobile robots. Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on In Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on, Vol. 1 (1996), pp. 429-435 vol.1.