Notes on TangentBug

October 7, 2015

1 Overview

The TangentBug algorithm consists of two distinct behaviours:

1. Move towards a goal
2. Obstacle following

Switching between behaviour 1 and 2 occurs when the robot detects that there is an impassable obstacle between it and the goal. Switching from behaviour 2 to behaviour 1 is slightly more complicated. TangentBug exits immediately if the robot has reached the goal. Otherwise, it begins in state 1.

2 Moving towards the goal

This behaviour is very simple, and operates as follows:

```
1: function MoveTowardsGoal( )
2:     if Robot is at goal then
3:         Stop and end algorithm
4:     else
5:         Turn towards goal and move forward, until and obstacle is detected between the robot and goal
6:         BoundaryFollow( )
7:     end if
8: end function
```

3 Boundary Following

The idea of boundary following is that the robot moves along the obstacle at some distance $d_desired$ until it can move towards the goal again. If it executes a complete loop around the obstacle without being able to move towards the goal, the algorithm ends in failure. At a high level, this behaviour is described by the following algorithm:

```
1: function BoundaryFollow( )
2:     leavingCondition ← false
3:     startLocation ← current location
4:     d_min ← distance to goal
5:     while current location is different to startLocation and leavingCondition=false do
6:         MoveAtAngle( )
7:         if d_min > current distance to goal then
8:             leavingCondition ← true
9:         end if
10:    end while
```

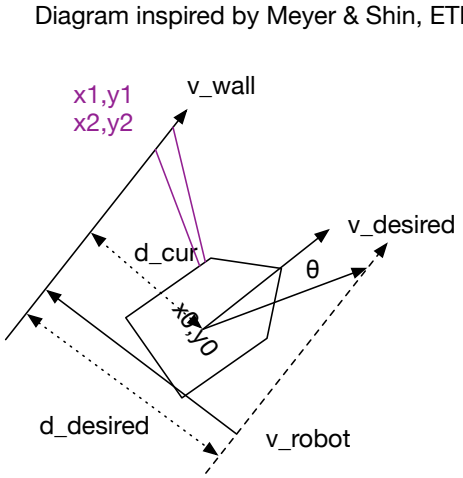

Figure 1: A robot and boundary.

```
11:     if current location = startLocation then
12:         exit algorithm with failure
13:     else
14:         MoveTowardsGoal( )
15:     end if
16: end function
```

The problem then is to identify what angle the robot should move at.

As an aside, when starting to follow a boundary, you have to choices with regards to which direction to turn. You should always turn in the direction which will minimise the distance to the goal. If equidistant, pick randomly.

4 Changing Angles

Consider the robot in Figure 1. The following table describes the various labels in the figure. For simplicity, we may assume that we are operating in the robot's frame of reference. Our problem is to determine the value of θ — how much the robot should turn to head in the appropriate direction. We assume that sensor 1 and 2 are two *adjacent* beams from our range finder[1].

$v_r = [x0, y0]^T = [0, 0]^T$	a vector representing the robot's origin. $[0, 0]^T$ in the robot's frame of reference.
$v_1 = [x1, y1]^T$	a vector representing the x and y position that sensor 1 has detected an obstacle at.
$v_2 = [x2, y2]^T$	a vector representing the x and y position that sensor 2 has detected an obstacle at.
v_{wall}	a vector pointing in the direction of the wall.
$d_{desired}$	the desired distance away from the wall that the robot should be at (prespecified)
v_{robot}	a vector perpendicular to the wall
d_{cur}	the current distance from the robot to the wall
$v_{desired}$	a vector representing our desired location (i.e., movement direction and distance from wall)
θ	the turning angle needed to reach the desired distance from the wall

The question is how to compute $v_{desired}$ and from this θ.

We begin by noting that $v_{wall} = v_2 - v_1 = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$. The vector perpendicular to the wall is then

[1] Try use the ones that report the closest distance to the obstacle.

$v_{robot} = \begin{pmatrix} y_2 - y_1 \\ -(x_2 - x_1) \end{pmatrix}$. The distance from the robot to the wall, d_{cur} can be computed as a dot product between a unit form[2] of v_{robot} and v_1, i.e.

$$d_{cur} = |\hat{v}_{robot} \cdot v_1| = \frac{x_1(y_2 - y_1) - y_1(x_2 - x_1)}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

.

Using the vectors v_{robot} and v_{wall} as an orthogonal coordinate system, we can now describe the vector $v_{desired}$ as

$$v_{desired} = [x_d, y_d]^T = d_{desired}\hat{v}_{wall} + (d_{current} - d_{desired})\hat{v}_{robot}$$

Finally, we can compute θ as follows:

$$\theta = arctan(y_d/x_d)$$

The `MoveAtAngle()` function then computes θ as above, turns, and moves forward for a distance $|v_{desired}|$ (i.e., until reaching the x, y coordinates of $v_{desired}$ in the robot's original frame of reference[3].

[2] given a vector $v = [x, y]^T$ we can obtain a unit vector of it as $\hat{v} = [x/(\sqrt{x^2 + y^2}), y/(\sqrt{x^2 + y^2})]^T$

[3] This assumes that the range sensing distance is much greater than $d_{desired}$, which is pretty much always the case.