SC627 - Assignment 1

Veejay Karthik J - 203230010

February 2022

1 Bug 1 Algorithm

The flowchart of the overall bug algorithm is as follows,

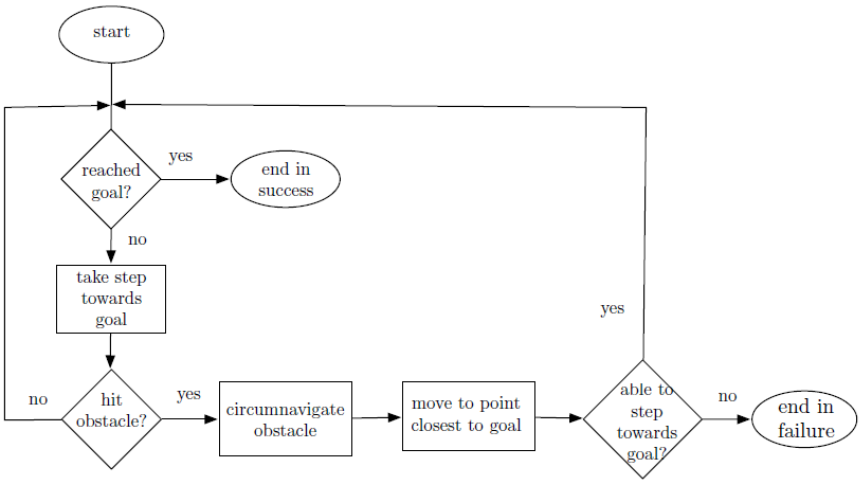

Figure 1: Bug 1 Algorithm

The function **MainScript()** drives the overall algorithm as illustrated in the flowchart above. To carry out the required functionalities to execute the bug 1 algorithm, the **MainScript()** calls the individual functions that are programmed for each specific functionality in an appropriate sequence as illustrated in the above flowchart.

1.1 Data Retrieval

The function **DataRetrievalText()** serves two purposes.

Firstly, it retrieves the necessary information for the working of the bug algorithm from an input text file. This information consists of the Start point, Goal point, step size and the set of vertices of the obstacles in the environment.

Further, it then passes this set of vertices into a function **VerticesSorter()** which sorts and rearranges in a cyclic order.

1.1.1 VerticesSorter()

To sort the vertices in a cyclic order, the following algorithm is employed. Given a set of $\{(x_i, y_i)\}$ representing the vertices of an obstacle, the centroid of this set is computed as (x_{cen}, y_{cen}). The co-ordinates of the vertices are then recomputed as $\{(x_i, y_i)\} - (x_{cen}, y_{cen})$ (Transformed Vertices).

Now, to arrange them in a cyclic order, the angle subtended by the transformed vertices with respect to the positive direction of $x-$ axis are sorted using the bubble sort technique in this function.

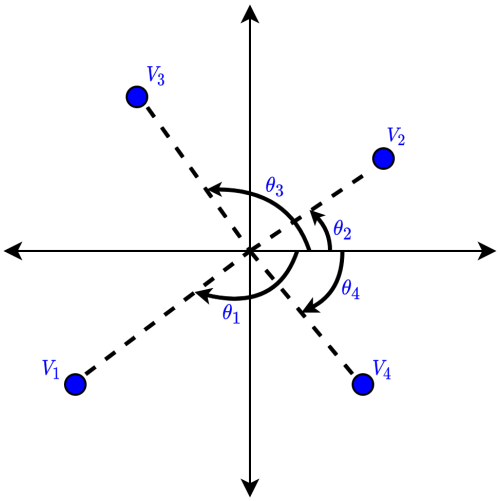

Figure 2: Transformed Vertices and their corresponding angles which are arranged in a cyclic order. Suppose if the set of vertices in the input text file are in a sequence given by $\{V_1, V_2, V_3, V_4\}$, this functions returns them in a cyclic sequence given by $\{V_1, V_4, V_2, V_3\}$ as shown in this figure.

1.2 checkCollision()

To check if a given point Q lies within a polygon P whose vertices are obtained as a set of co-ordinates in a cyclic order, the following algorithm is employed.

1. The edges of the polygon are represented as vectors as follows,

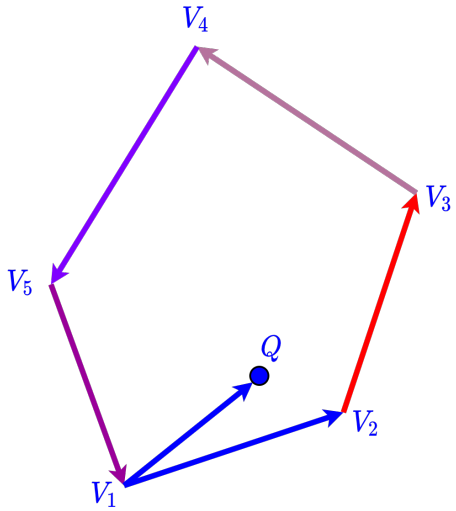

Figure 3: The polygon edges represented as vectors. Each set of edge vectors are represented as $\{\mathbf{V_i V_{i+1}}\}$ as illustrated in this figure

2. Suppose if the point Q lies inside the polygon P, the following condition holds true for each edge vector.

   The sign of the following cross product remains the same with respect to the all the edge vectors - $(\overrightarrow{V_iV_{i+1}} \times \overrightarrow{V_iQ})$

   Suppose if the cross product changes sign with any edge vector, it implies that the point Q is outside the polygon P. Here, since the workspace in which the motion planning is carried out is considered to be the XY plane, the cross product that is defined above is always either in $+Z$ or $-Z$ directions. Therefore the notion of positive and negative signs of the cross product are considered with respect to the Z axis.

1.3 Tangent_Computer()

During the circumnavigating phase of the motion planning algorithm, this function computes the unit direction vector in which in the next step is to be taken. The step size of these subsequent steps is obtained as an input from the input text file.

Based on the location of the robot around the obstacle, this function computes the unit tangent vector in two possbile cases.

1. **Case 1:** If the closest point on the obstacle polygon from the robot is any point on one of its edges excluding the vertices, the instantaneous unit vector in which the next step is to be taken is computed as the unit vector to the corresponding edge vector.
2. **Case 2:** If the closest point on the obstacle polygon from the robot is any of its vertices, the instantaneous unit vector in which the next step is to be taken is computed as the tangent to the circle drawn through closest point and the robot, at the robot's position.

These cases are as illustrated in Fig. (4)

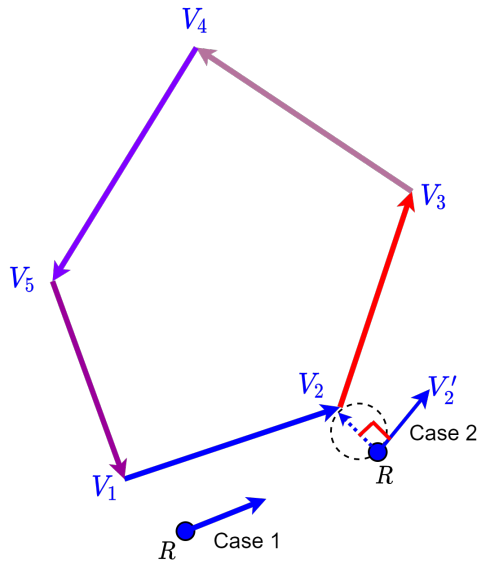

Figure 4: The unit vectors representing the direction in which the next step is taken during the different possible cases is illustrated in this figure. In case 2, $RV_2 \perp RV_2'$ (Tangent to the circle).

If the edge vectors encompass the obstacle polygon in anti-clockwise direction, the unit tangent vector for the subsequent step can be computed as follows, (Here $\mathcal{R}(\theta)$ represents the rotation matrix required to create a vector rotation in θ radians anti-clockwise)

$$\overrightarrow{RV_2'} = \mathcal{R}(-\frac{\pi}{2})\frac{\overrightarrow{RV_2}}{||\overrightarrow{RV_2}||}$$

1.4 closestPolygonEdgeComputer()

This function computes the closest point on the obstacle with respect to the robot's current position. Since the each polygonal obstacle is characterized by a set of line segments with the vertices given by a set of point P_i, the set of points within each line segment(P_λ) can be parameterized as follows,

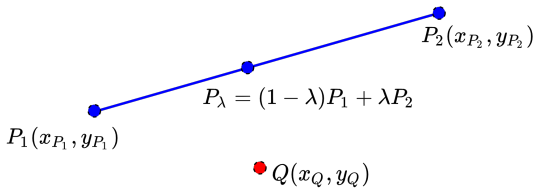

Figure 5: Parameterization of points in a line segment

$$P_\lambda = (1-\lambda)P_1 + \lambda P_2 \quad ; \quad \lambda \in [0,1]$$

Using this parametric representation of the line segment, the closest point on the same with respect to any arbitrary outside point can be computed as the solution to the following constrained optimization problem. (Here, $d(.,.)$ represents Euclidean distance function)

$$\begin{aligned} \min_{\lambda} \quad & d(Q, P_\lambda) \\ & 0 \leq \lambda \leq 1 \end{aligned} \tag{1}$$

This optimization problem can be solved as an unconstrained problem by employing the first-order condition for optimality and imposing bounds on the obtained value of the optimal λ^* as follows. Lets optimal value of λ obtained as a solution to the above optimization problem by solving it in the unconstrained paradigm be represented as λ_u^*.

$$\left(\frac{d}{d\lambda}(d(Q, P_\lambda))\right)_{\lambda=\lambda_u^*} = 0 \tag{2}$$

The solution to equation (2) is analytically obtained as follows,

$$\lambda_u^* = \frac{(x_Q - x_{P_1})(x_{P_2} - x_{P_1}) + (y_Q - y_{P_1})(y_{P_2} - y_{P_1})}{(x_{P_2} - x_{P_1})^2 + (y_{P_2} - y_{P_1})^2}$$

To optimal value of λ^* which is the solution to the actual optimization problem is computed from λ_u^* as follows,

$$\lambda^* = \begin{cases} 0 & \lambda_u^* < 0 \\ \lambda_u^* & 0 \leq \lambda_u^* \leq 1 \\ 1 & \lambda_u^* > 1 \end{cases}$$

Using this optimal value of λ^*, the closest line segment of the polygon P can be obtained by computing the optimal distances to each line segment of the polygon and determining that line segment which is at the least distance from an exterior point Q.

Suppose if the line segment under consideration is P_iP_{i+1}, based on the value of λ^* obtained, conclusions on the location of the closest point can be made.

1. Suppose if $0 < \lambda^* < 1$, it implies that the closest point lies in the interior of the line segment P_iP_{i+1} and the unit tangent vector in which the next step should be taken is computed as a unit vector parallel to $\overrightarrow{P_{i+1}P_i}$
2. The cases where $\lambda^* = 0$ or $\lambda^* = 1$ corresponds to the case where the closest point on the line segment is its vertices P_i and and P_{i+1} respectively. In this case, the unit tangent vector is computed as the tangent to the circular region as shown in the previous subsection.

1.5 Euclidean_distance()

This function computes and returns the Euclidean Distance between the points passed through it as its arguments.

2 Simulation Results

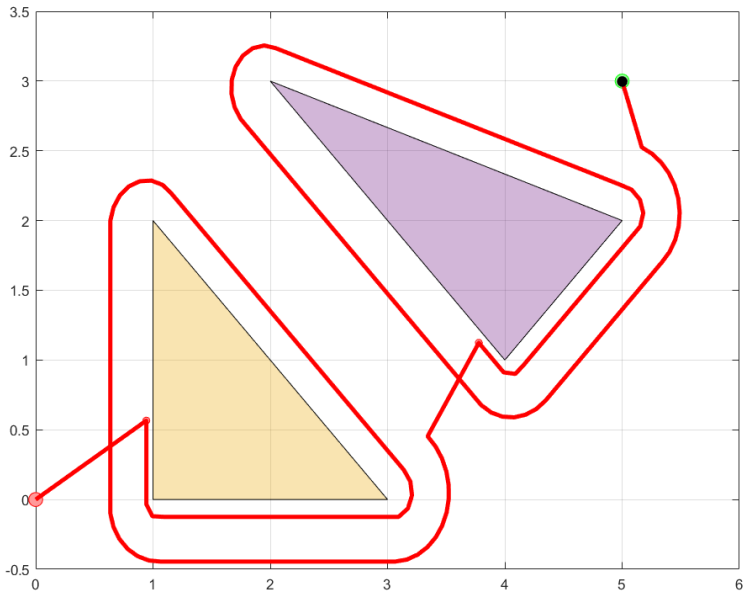

Figure 6: Simulation Output

The simulation output resulting from the motion planning carried through the Bug 1 algorithm is illustrated in the above figure. The inputs to the Bug 1 algorithm are read from the given input text file in the question and the simulation is carried out on the basis of the same.