SC 627 - Motion Planning and Coordination for Autonomous Vehicles
Assignment I: Bug 1 Algorithm

Gagan Jain | 180100043

Bug 1 Algorithm Flowchart

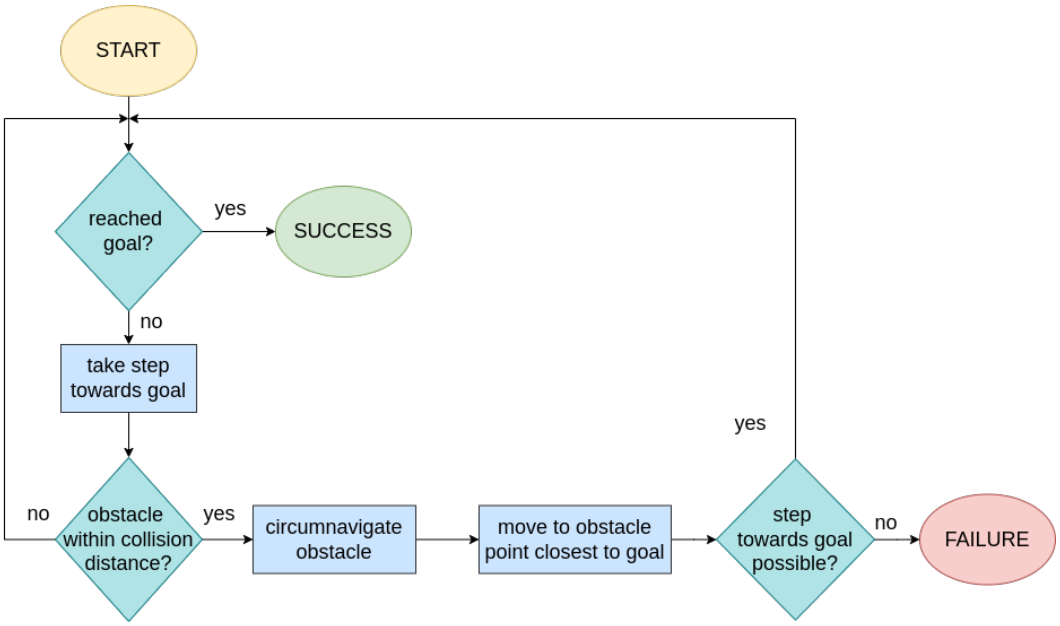

Figure 1: Bug 1 Algorithm

BugBase to Bug 1

My implementation for the BugBase Algorithm within the algorithm class goes below with appropriately defined functions. The functions developed in E.16 and E1.7 for finding the closest distance of a point from a polygon are used to find the closest obstacle to the bug at every step.

```
def bugBaseAlgorithm(self):
        while not self.goalCondition():
                self.findClosestObstacle()
                if self.isNearObstacle():
                        return self.getAlgoOutcome(fail = True)
                self.moveTowardsGoal()
        self.storeData()
        return self.getAlgoOutcome(fail = False)
```

It is quite evident that the BugBase algorithm just chooses to objectively move towards the goal at every step and if it encounters an obstacle in between, it just gives up and returns failure. This is not a complete algorithm, and is only guaranteed to find a existing path if the line joining the start and goal points is not blocked anywhere in between.

Now to shift to the Bug 1 algorithm, we need to work to develop the logic to go around the obstacles. Like the BugBase algorithm, this also tries to move in the direction of the goal until reached. But, when it encounters an obstacle in its path, it chooses to circumnavigate it and store the point on the obstacle which is closest to the goal point while doing so. This is where the functions developed in E1.6 and E1.7 come in handy.

The circumnavigation requires the knowledge of closest point on the obstacle to the bug and a tangent vector to move ahead. I have used the convexity of polygonal obstacles as an assumption for developing the termination condition. For the obstacle which is being circumnavigated, we compute the centroid and store the vector direction from the centroid to the position where the circumnavigation is started. The termination condition checks for the current vector direction and when it becomes parallel to the initial vector, it is assured that a complete 360 degree rotation has taken place, so the circumnavigation is terminated.

After completing the circumnavigation, the bug continues to move around the obstacle until it reaches the point which is closest to the goal on the obstacle. To check whether this has been reached, we check whether the bug is in proximity of the line segment joining that closest point to the goal point. Note that this could have been done by just checking the proximity to the closest point but due to environment randomness, the bug drifts sometimes. So, for added robustness, we chose to check proximity to the line.

Now, it is important to check whether a step towards the goal leads to hitting the obstacle or is in free space. For this, we check the angle between the lines joining the current position to the centroid and to the goal point. It can be shown that this angle has to be obtuse for a convex polygon. So, in case this angle is acute, we report failure, otherwise we take a few steps away from the obstacle towards the goal to get out of the obstacle proximity condition, in order to avoid a second circumnavigation around the same obstacle. Once out, the bug again starts heading towards the goal until it reaches there or encounters another obstacle. For the latter, the entire circumnavigation process is repeated. Finally, if the bug reaches the goal, it reports success and returns the followed path, goal distance at every step and total path length.

The Bug1 code snippet under the Bug1AlgoClass in my repository has been shown below.

```
def bug1Algorithm(self):
        while not self.goalCondition():
                self.findClosestObstacle()
                if self.isNearObstacle():
                        self.circumNavigation()
                        self.moveToObstaclePointClosestToGoal()
                        if not self.isGoalFeasible():
                                return self.getAlgoOutcome(fail = True)
                        self.moveLittleAwayFromObstacle()
                self.moveTowardsGoal()
        self.storeData()
        return self.getAlgoOutcome(fail = False)
```

Simulation results

The given test case produced the following results upon simulation of the Bug1 Algorithm:

Result of the motion planning algorithm: Success
The total length of path taken: 27.346575435580032 metres
Time taken by the algorithm is: 798.4340562820435 seconds

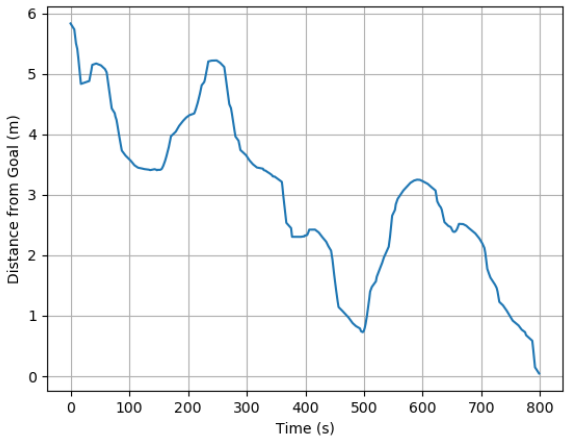

Figure 2: Bug to Goal Distance vs Time

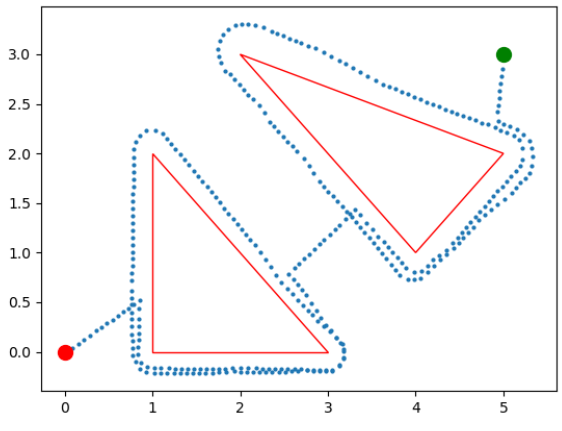

Figure 3: Path visualization