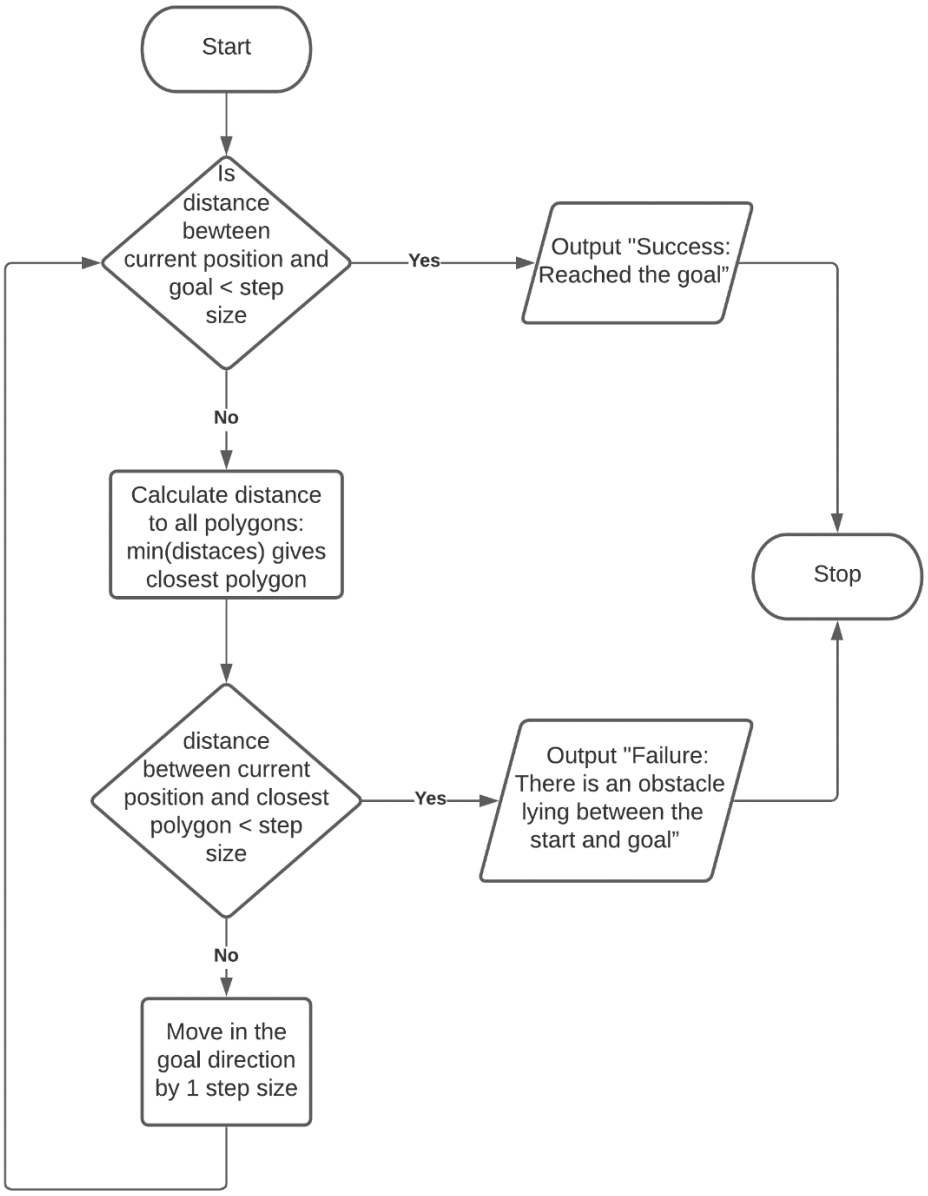
# Report File
# Assignment 1

Shashwat Shukla

213234001

**Q.Perform the following tasks and document them in a concise project report:**

**(i) Sketch a flowchart and implement the Bug-Base algorithm.**



**(ii) Describe in a paragraph how you will modify BugBase to implement the Bug 1 algorithm. Explain the roles of the geometric functions in Exercises (E1.6) and (E1.7) and what new logic will be needed.**

**Ans.** The pseudo code of the bug-1 algorithm is as follows:

**Bug 1 algorithm**

```
1: while not at goal :
2:     move towards the goal
3:     if hit an obstacle :
4:         circumnavigate it (moving to the left or right is unimportant). While cir-
           cumnavigating, store in memory the minimum distance from the obstacle
           boundary to the goal
5:         follow the boundary back to the boundary point with minimum distance to
           the goal
```

The bug base algorithm ends in failure when an obstacle is detected.

To modify to Bug 1, when an obstacle is detected, the program instead of stopping, should enter a loop. In this loop, the bot should circumnavigate the obstacle, moving along its boundary and record its path. When the robot returns to the point where it began the circumnavigation, it should calculate the point closest to the goal, and move towards the point, again circum navigating the obstacle.

Once the point closest to the goal is reached, it should be checked if moving towards the goal from that point is possible or not. If not, the program ends in failure. If it is possible to move, the above steps followed are repeated until the goal is within one step size.

The functions used in Exercise 1.7 , ***computeDistancePointToPolygon,*** helps to detect the obstacle. If the obstacle is within one step size moving towards the goal, it is assumed the obstacle is reached.

The function ***computeTangentVectorToPolygon***, gives a vector as output, which faces a direction tangent to the obstacle. This is the direction, we move in to circumnavigate the obstacle.

The functions described above from E1.7, use functions described in E1.6 to work.

The function ***computeLineThroughTwoPoints***, gives the parameters for the equation of line passing through any two given lines.

The function ***computeDistancePointToLine***, gives the distance from a line. The equation of the line is parameterised by the output of the ***computeLineThroughTwoPoints*** function.

The function ***computeDistancePointToSegment***, gives the distance from a line segment between two points. This function utilises the result of the function ***computeDistancePointToLine*** to calculate the distance. This function is then used by the function to find distance from the polygon by finding the minimum distance from each edge of the polygon.

**(iii) Implement a fully-functioning version of Bug 1, described as follows: computeBug1**

**Input: Two locations start and goal in Wfree, a list of polygonal obstacles obstaclesList, and a length step-size**

**Output: A sequence, denoted path, of points from start to goal or returns an error message if such a path does not exists. Successive points are separated by no more than step-size and are computed according to the Bug 1 algorithm.**

**(iv) Test your program on the following environment:**

**start = (0; 0) and goal = (5; 3)**

**step-size = 0:1**

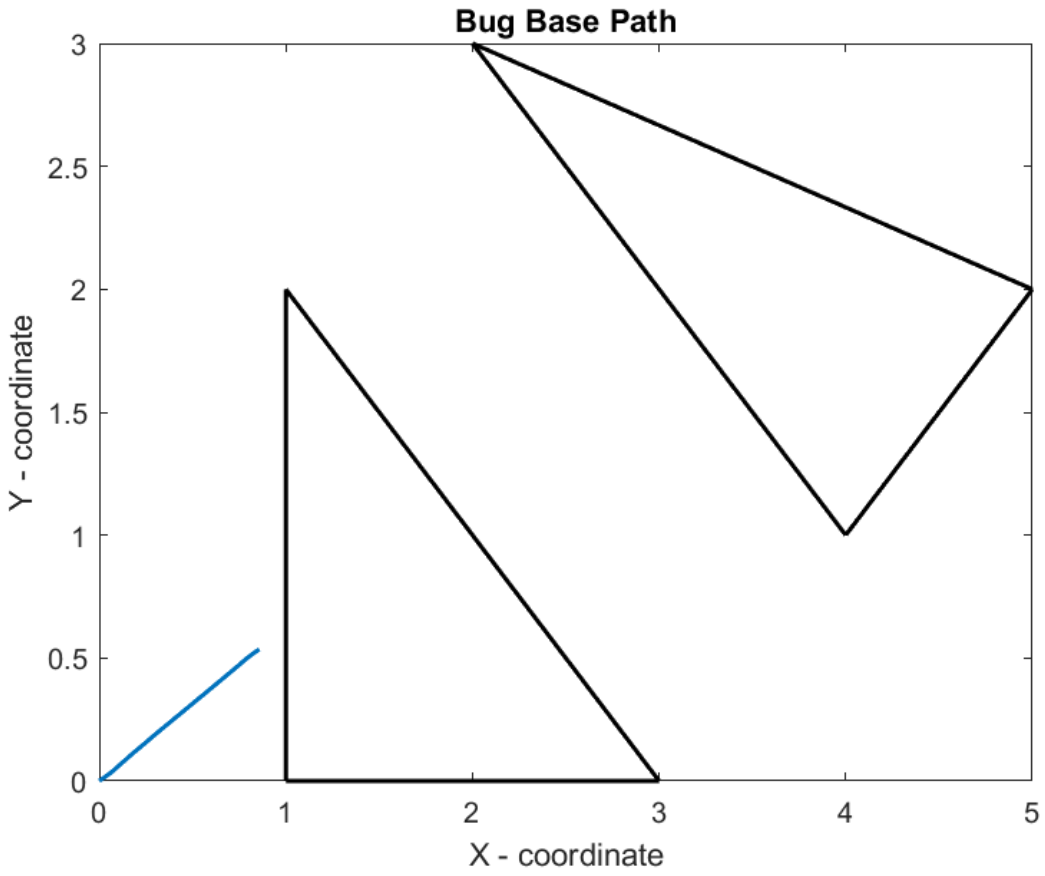
**obstaclesList = {(1; 2); (1; 0); (3; 0)}; {(2; 3); (4; 1); (5; 2)}**

**Include in your report a plot of the path taken by your bug from start to goal, the total path length and the computing time, and a plot of the distance from the bug's position to the goal as a function of time.**

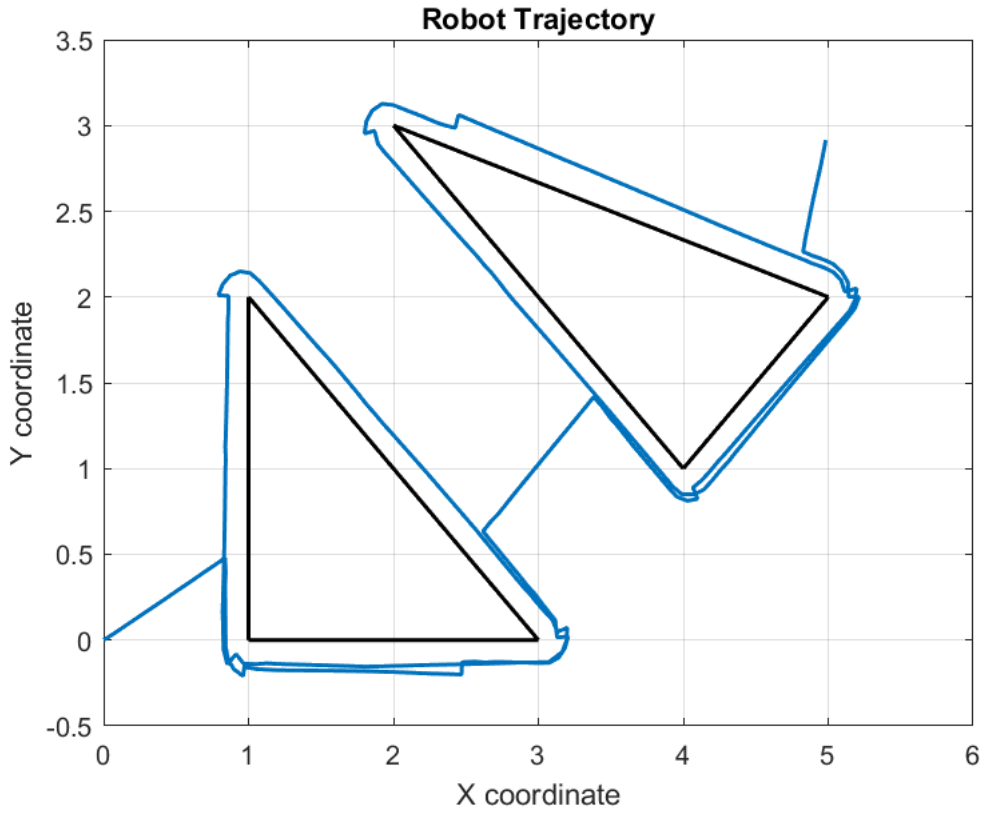
The bug base and bug 1 algorithms are applied to the scenarios mentioned in question (iv).

The results are shown as follows:

a) The path taken by the robot following BugBase algorithm.




b) The path taken by the robot using Bug 1 algorithm.




c) The distance to goal a function of time using the Bug 1 algorithm.: