Implementation of Bug 2 Algorithm

(Submitted by Pavan Poudel)

Motion planning is a fundamental task in robotics. Having different properties of robots, the form of effective motion planner may vary. Once the task and the robot system is defined, we can choose between algorithms based on how they solve the problem. Some algorithms consider global knowledge of the working space, and some not.

Bug algorithms assume only local knowledge of the environment and a global goal. These algorithms assume the robot is a point operating in a two-dimensional plane. The obstacles have bounded perimeters and are finite in number and robot has no prior knowledge of locations and shapes of the obstacles. These algorithms also assume that robot has small computational power and amount of memory but can compute the direction forward the goal from its current position as well as the distance between two points.

Bug 1 and Bug 2 algorithms are among the earliest and simplest sensor based motion planning algorithms with tactile sensor (zero range sensor), later Tangent Bug algorithm introduced with the finite sensing range. In Bug algorithms, success is guaranteed, when possible. The algorithms require two behaviors: move on a straight line towards goal or navigate the obstacle.

In this project, Bug 2 algorithm has been implemented using Python programming. Among Bug 1, Bug 2 and Tangent Bug algorithms, I preferred to implement this Bug 2 algorithm as I found it easier than other to implement.

In Bug 2 algorithm, robot is moving towards the goal unless an obstacle is detected, then navigates the obstacle until it reaches to the straight line joining the start position of robot and goal or navigation-start-point of the obstacle. The following steps describe the Bug 2 algorithm in detail:

Algorithm 2 Bug2 Algorithm

Input: A point robot with a tactile sensor

Output: A path to q_{goal} or a conclusion no such path exists

1: **while** True **do**
2: **repeat**
3: From q_{i-1}^{L}, move toward q_{goal} along m-line.
4: **until**
q_{goal} is reached **or**
an obstacle is encountered at *hit point* q_i^H.
5: Turn left (or right).
6: **repeat**
7: Follow boundary
8: **until**
9: q_{goal} is reached **or**
10: q_i^H is re-encountered **or**
11: m-line is re-encountered at a point m such that
12: $m \neq q_i^H$ (robot did not reach the hit point),
13: $d(m, q_{\text{goal}}) < d(m, q_i^H)$ (robot is closer), and
14: If robot moves toward goal, it would not hit the obstacle
15: Let $q_{i+1}^{L} = m$
16: Increment i
17: **end while**

Figure 1. Bug 2 Algorithm

In this project, two obstacles have been considered: one is rectangular closed obstacle and the other is a L-shaped solid obstacle. A rectangular working space of 960 * 720 has been considered. The figure 2 shows the complete view of the working space and obstacles. As shown in figure 2, there are many options available. The success scenarios are either both robot and goal positions are outside the obstacles or both are within the rectangular obstacle 1. And the failure scenarios are either one of them is inside the rectangular obstacle 1.

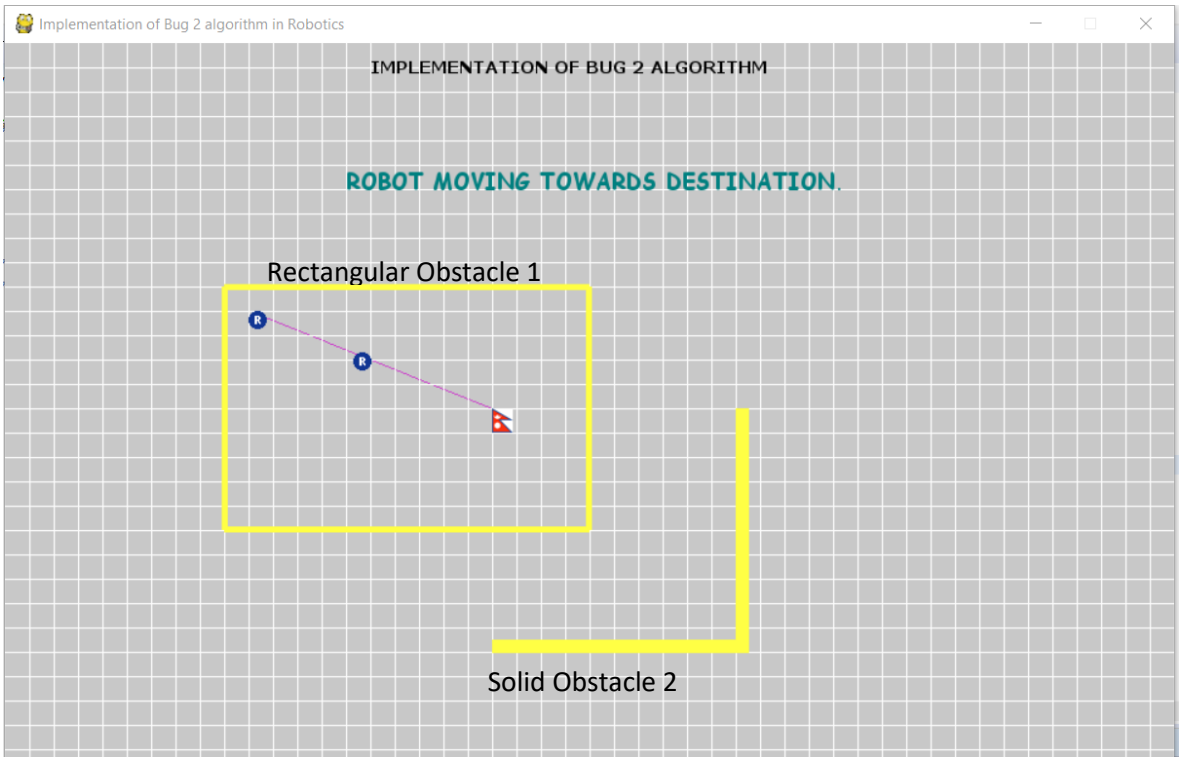

Figure 2. Working space of robot with obstacles

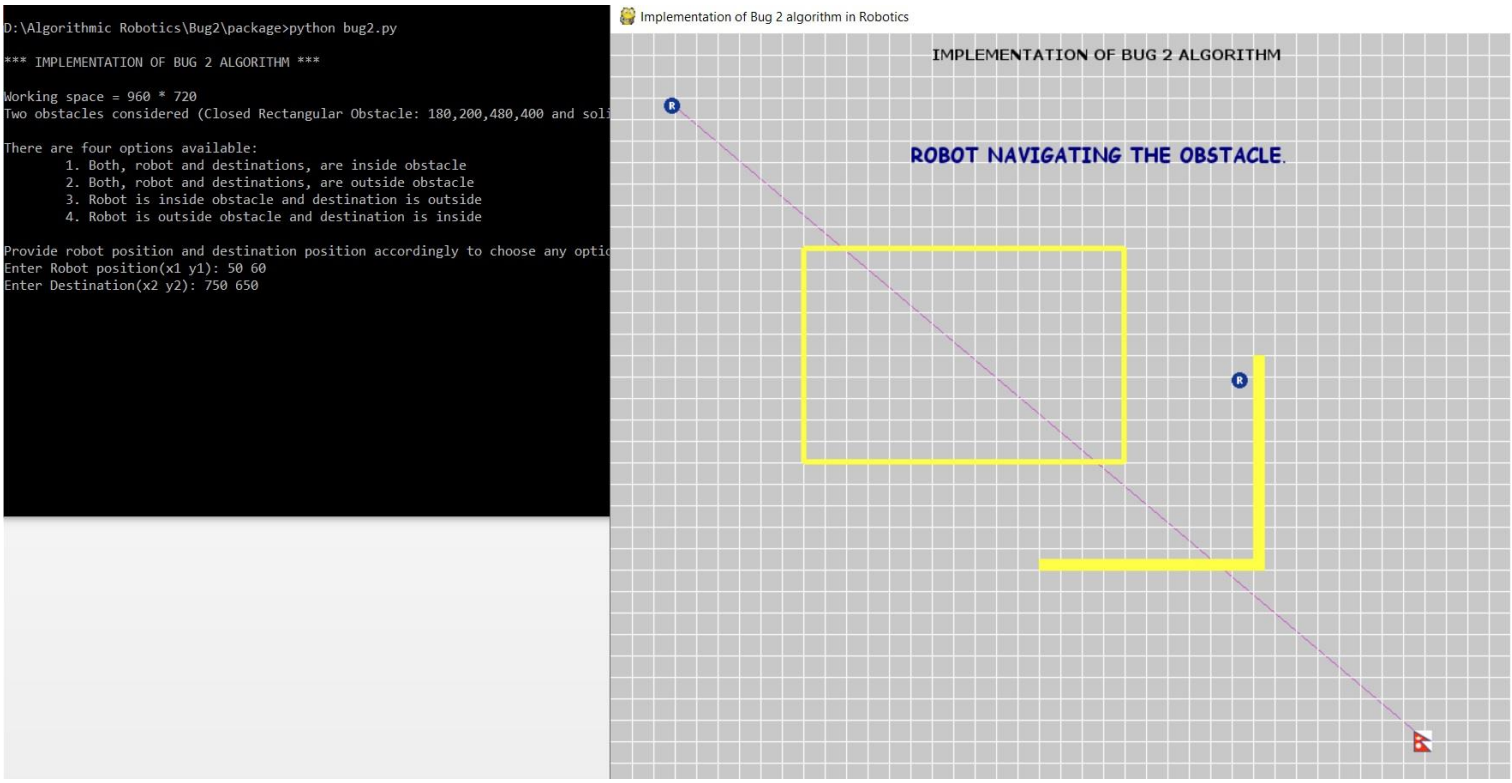

Figure 3. Output with both robot and goal outside obstacle 1

The following pseudocode describes in detail about the implementation:

1. Input robot and goal positions (R and G respectively): R(x1, y1), G(x2, y2)
2. Calculate slope (m) of the line joining R and G

$$m = \frac{y2 - y1}{x2 - x1}$$

3. Among different possible cases, execute one:
    a. If R and G both are inside closed rectangular obstacle 1:
        i. Move along the straight line towards goal until it reaches.

        ```
        while(x1 < x2){
                x1 += 1, y1 += m
        }
        ```

        ii. Return success message
    b. Else if R is inside and G is outside the obstacle 1:
        i. Calculate point of intersection (x_intersect, y_intersect) of straightline joining R & G and obstacle.
        ii. Move along straight line towards goal upto (x_intersect, y_intersect) (i.e until it collides with obstacle), navigate the whole obstacle (rectangle) from inside until it reaches again the point of contact (x_intersect, y_intersect).
        iii. Return unsuccessful message
    c. Else if R is outside and G is inside the obstacle 1:
        i. Calculate point of intersection (x_intersect, y_intersect) of straightline joining R & G and obstacle.
        ii. Move along straight line towards goal upto (x_intersect, y_intersect) (i.e until it collides with obstacle), navigate the whole obstacle (rectangle) from outside until it reaches again the point of contact (x_intersect, y_intersect).
        iii. Return unsuccessful message
    d. Else if both R and G are outside the obstacle 1:
        i. Calculate the hit point and leaving point by finding the point of intersections of straight line joining robot and goal position and the edges of obstacles.

ii. Move along straight line towards goal upto hit point, navigate the obstacle until it reaches the leaving point of the obstacle, then move again along straight line towards goal until it reaches or if another obstacle detected repeat the above process.
iii. Return success message

Implementing the algorithm, there raised many cases. To consider all the cases, it takes more time. That's why there are some bounded conditions listed below, but still it perfectly simulates the algorithm within that boundary condition:

1. Slope of the line joining R and G must be positive
2. Robot position should be smaller than the goal position i.e. (x1,y1) < (x2,y2)
3. Has not considered for both robot and goal at horizontal line or vertical line

In this way, Bug 2 algorithm has been implemented.

***