SC627: Motion Planning and Coordination of Autonomous Vehicles

Assignment-1 Report

February 2023

1 Introduction

The bug-1 algorithm is a simple motion planning algorithm that allows a robot to navigate safely to a given goal point while avoiding any detected obstacles in the way. The algorithm makes the bot move toward the goal unless there is an obstacle in the way. If an obstacle is detected, the algorithm makes the bot circumnavigate the boundary of the obstacle while storing the point on the boundary nearest to the goal. Once the circumnavigation is over, the algorithm makes the bot move to the point on the boundary closest to the obstacle and then makes it move in the direction of the goal again.

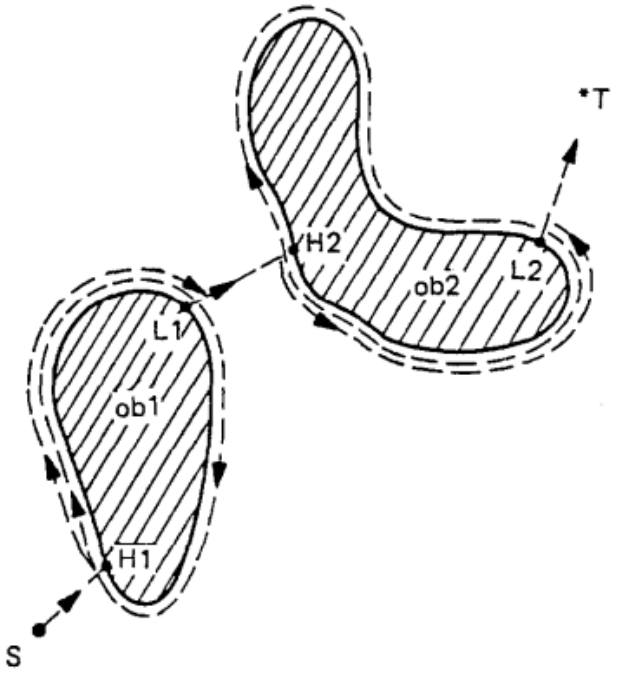

Figure 1: Path generated by Bug-1 algorithm; *Source: Lumelsky, V.J., Stepanov, A.A. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica 2, 403–430 (1987).*

2 Pseudocode

Algorithm 1 Bug-1 Algorithm

```
entry ← None
exit ← None
didNotReachGoal ← True
exitPointNotFound ← True
circumnavigating ← False
while didNotReachGoal do
  if currentPosition = Goal then
    didNotReachGoal ← False
  else if No Obstacle Detected and not circumnavigating then
    Move towards the goal
  else if Obstacle Detected then
    circumnavigating ← True
    entry ← currentPosition
    exit ← currentPosition
    while exitPointNotFound do
      Circumnavigate the boundary
      if distanceOfGoalFromCurrentPosition < distanceOfGoalFromExit then
        exit ← currentPosition
      end if
      if Rearrival at entry then
        exitPointNotFound ← False
      end if
    end while
    Move towards the exit point along the boundary
    if Reached exit point then
      circumnavigating ← False
      Move towards goal
    end if
  end if
end while
```

3 Simulation Results

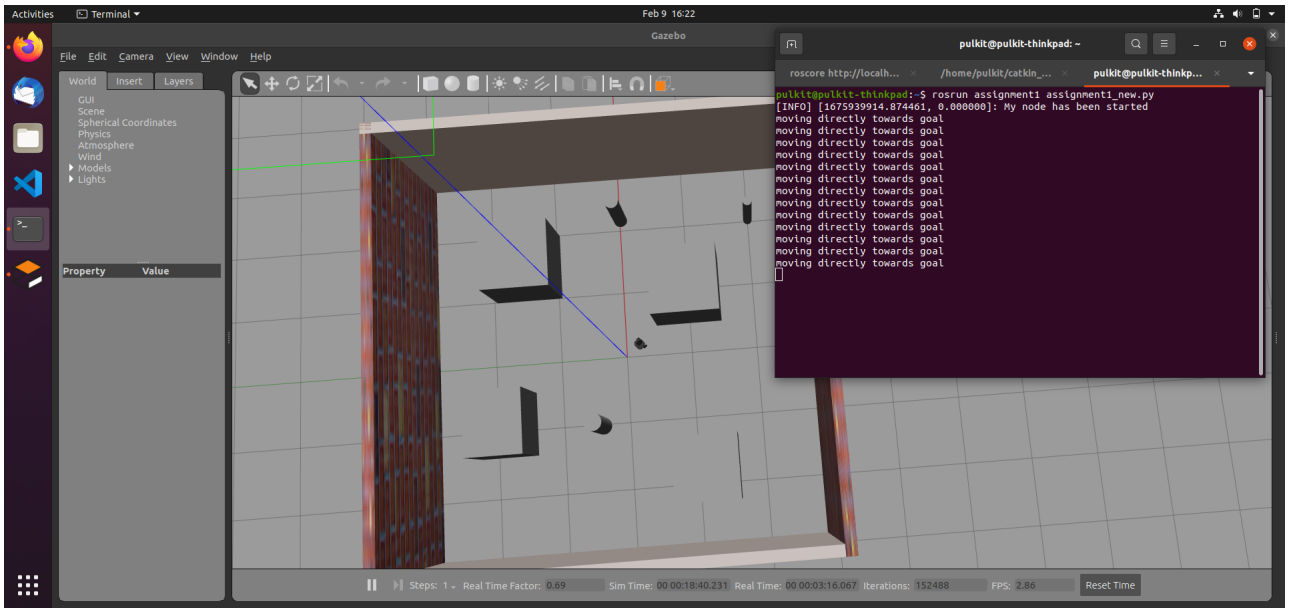

Figure 2: Moving directly towards goal in absence of obstacles, goal at (2.5, -2.5)

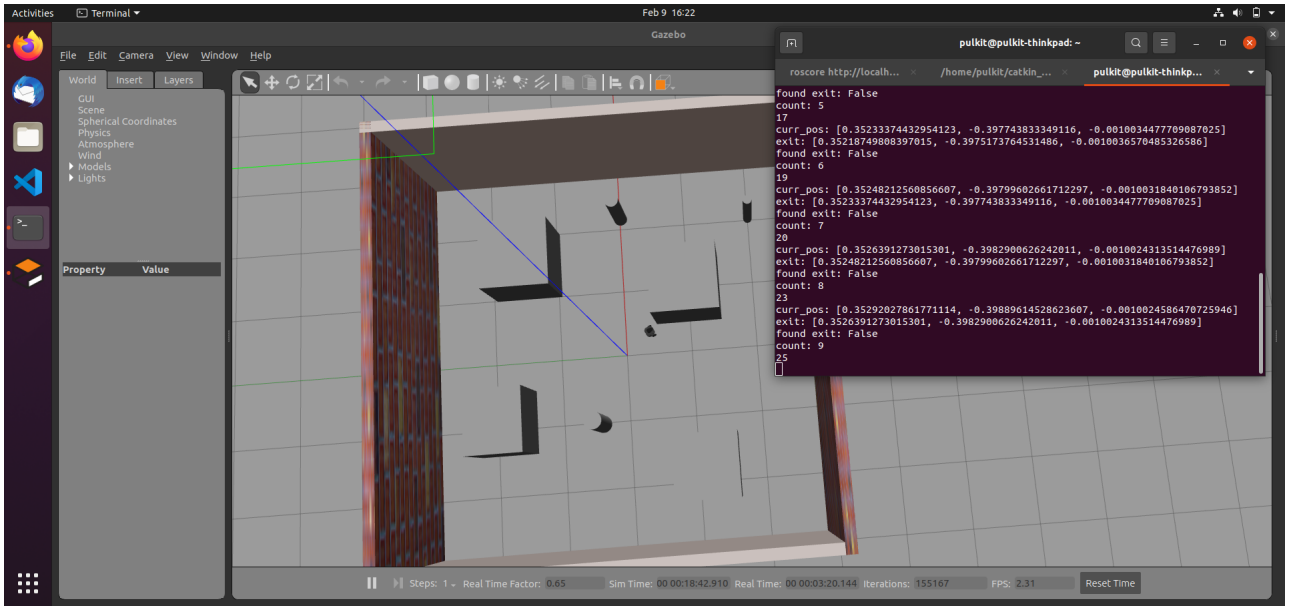

Figure 3: Encountering obstacle (corner of a square), goal at (2.5, -2.5)

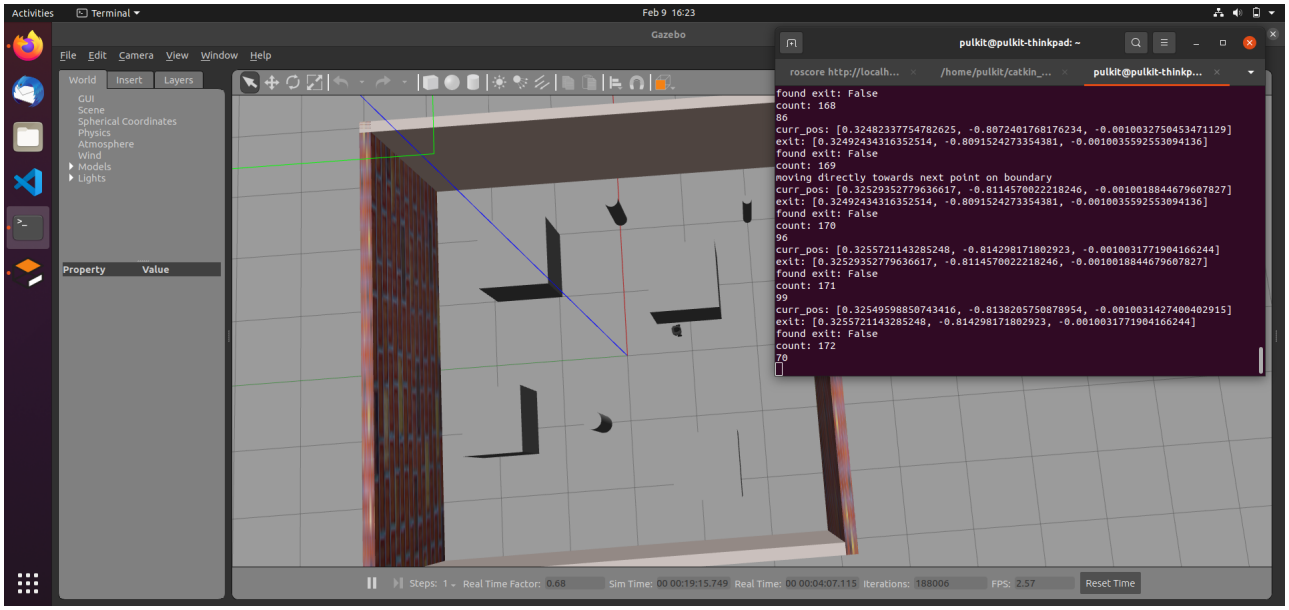

Figure 4: Circumnavigating boundary of a square obstacle, goal at (2.5, -2.5)

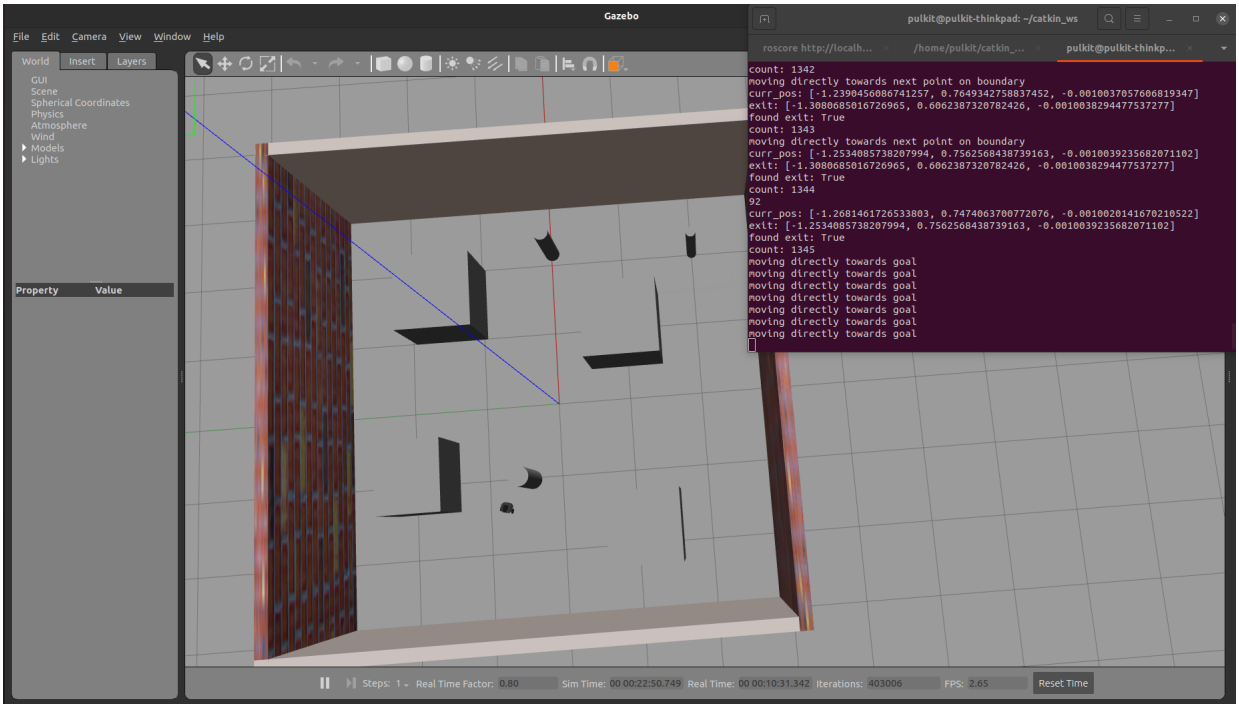

Figure 5: Exiting after circumnavigating a cylindrical object, goal at (-2,1)

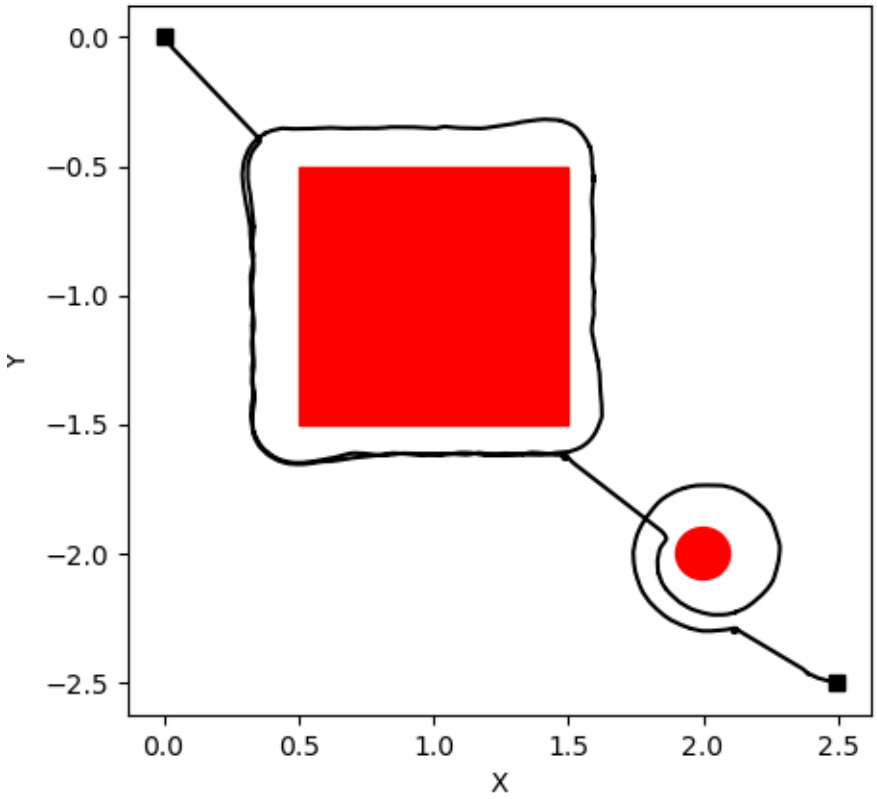

Figure 6: Odometry data extracted and plotted on matplotlib

The code implemented makes the bot move at an angle of 90 degrees to the closest obstacle point while circumnavigating. To avoid the bot from deviating too far from the obstacle boundary, the angle is maintained at 84 degrees with some tolerance for the nearest obstacle point. The threshold for detecting obstacles is 0.2, the threshold for closing on entry and exit points while circumnavigating the points is 0.1. For circular objects, the bot might deviate in a larger radius during its second revolution, so it might cross a point closer to the goal than the one initially computed in the first revolution. Hence the code allows for the exit point to be updated in the bot's second revolution - allowing the bot to either arrive close to the exit point and then move towards the goal, or let it move out of the circumnavigation from a new exit point closer than the older exit point (which is very close to the previous one, but not within the threshold 0.1 distance of the previously decided exit point).

4 Hardware Results

The code was appropriately modified so that it can be implemented properly on hardware. The angular and linear speeds were firstly brought down within the hardware constrained speed of 0.15 m/s. The threshold distance was increased to 15cm and the angle at which the bot was maintained from the nearest point on boundary was kept at 81 degrees with an allowed error of $\pm$ 10 degrees. Moreover, the laserscanner data was taken to be an array of dimension 360 in the simulation code, however, on hardware, the dimension of the data need not be constant, hence the angle between the bot and the nearest point on the obstacle was calculated by appropriately scaling the index of the minimum value from the data array. The bot was directed to move from its start (0,0) towards its goal at (2,-2), which it successfully reached upon the execution of the code. The path taken by the bot has been shown in Fig. 7.

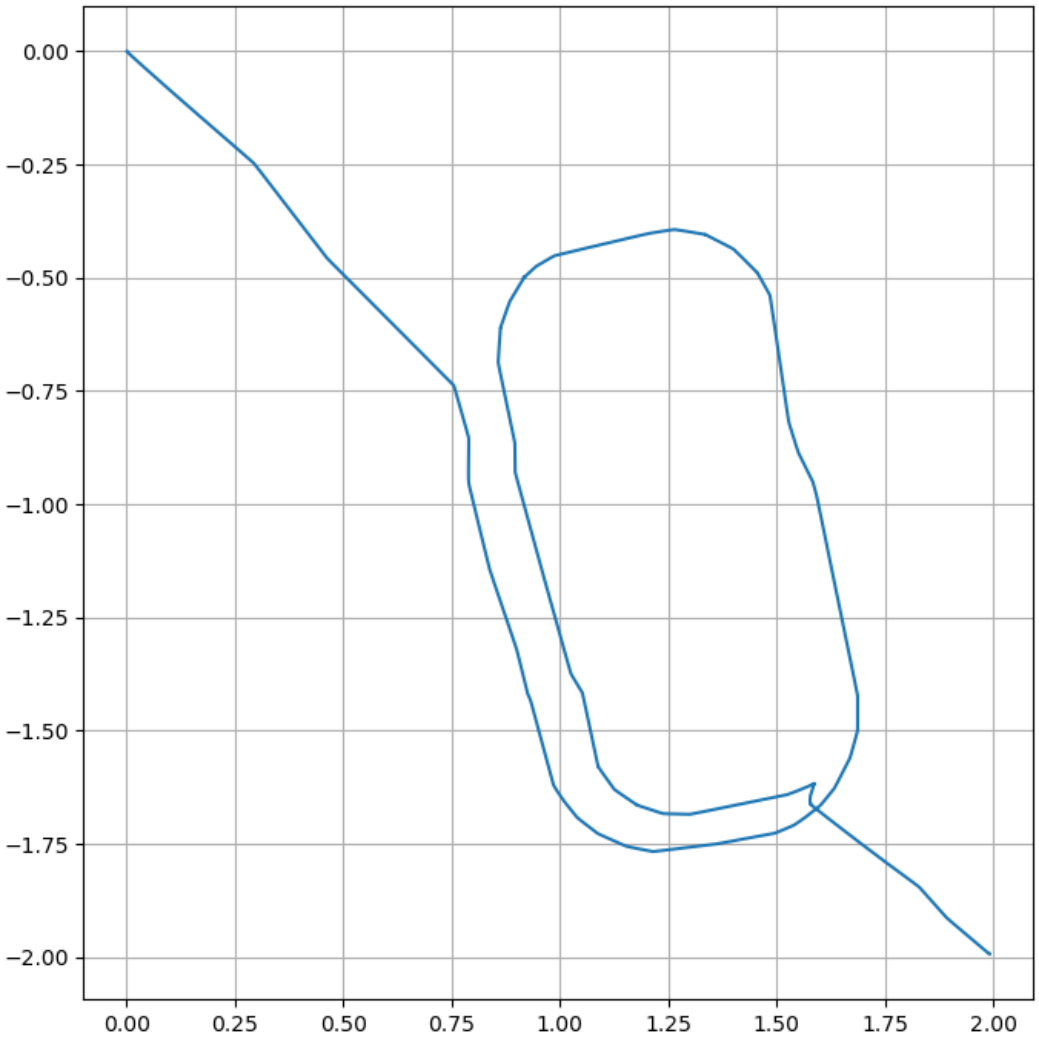

Figure 7: Plot of the odometry data obtained after the bot moved from its start to its goal position using the bug-1 algorithm