SC 627 Motion Planning

first Assignment

Omar Kashmar

213011004

The pseudocode code of bug 1 Algorithm:

1.Import the necessary libraries:

```
# import the messges

from geometry_msgs.msg import Point

# the point messge conatins # This contains the position of a point in free space

#float64 x

#float64 y

#float64 z

from sensor_msgs.msg import LaserScan

from nav_msgs.msg import Odometry

from tf import transformations

#we need the transformation to get the euler or qurtanion

# import the gazebo messges

from gazebo_msgs.msg import ModelState

from gazebo_msgs.srv import SetModelState

# import ros service

from std_srvs.srv import *

#std_srvs contains two service types called Empty and Trigger, which are common service patterns for sending a signal to a ROS node. For the Empty service, no actual data is exchanged between the service and the client.

# The Trigger service adds the possibility to check if triggering was successful or not.

import math

# for calculation

import numpy as np
```

2. start the bug1 Algorithm: we need to pass the start location and obstacles and the velocity command

```
def computeBug1(start, goal, obstaclesList, step_size):
    def is_obstacle(point):
        for obstacle in obstaclesList:
            if point in obstacle:
                return True
        return False
```

3. calculate the Euclidean distance between every two points

```
    def compute_distance(point1, point2):
        return np.sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)
```

4.calculate the angle between every two points

```
 def compute_angle(point1, point2):
        return np.arctan2(point2[1]-point1[1], point2[0]-point1[0])

    path = [start]
    current_point = start
    heading = compute_angle(current_point, goal)
```

5.curicimnvigates around the obstacles

```
    while True:
        next_point = (current_point[0] + step_size * np.cos(heading),
                      current_point[1] + step_size * np.sin(heading))

        if is_obstacle(next_point):
            heading = compute_angle(current_point, goal)
            continue

        path.append(next_point)
        current_point = next_point
```

6.check if we are near the goal location as nearst goal and move

```
        if compute_distance(current_point, goal) <= step_size:
            return path
    return "Error: No path exists"
```

The Concept of Bug 1 Algorithm:

Bug 1 Algorithm is a basic robotic path planning algorithm that involves moving in a straight line toward a goal until an obstacle is encountered, then moving around the obstacle while keeping it on a particular side until the goal is reached. This process is repeated until the goal is reached. The algorithm is called "Bug 1" because it is the first of a series of Bug algorithms that were developed in the 1980s for autonomous robot navigation.

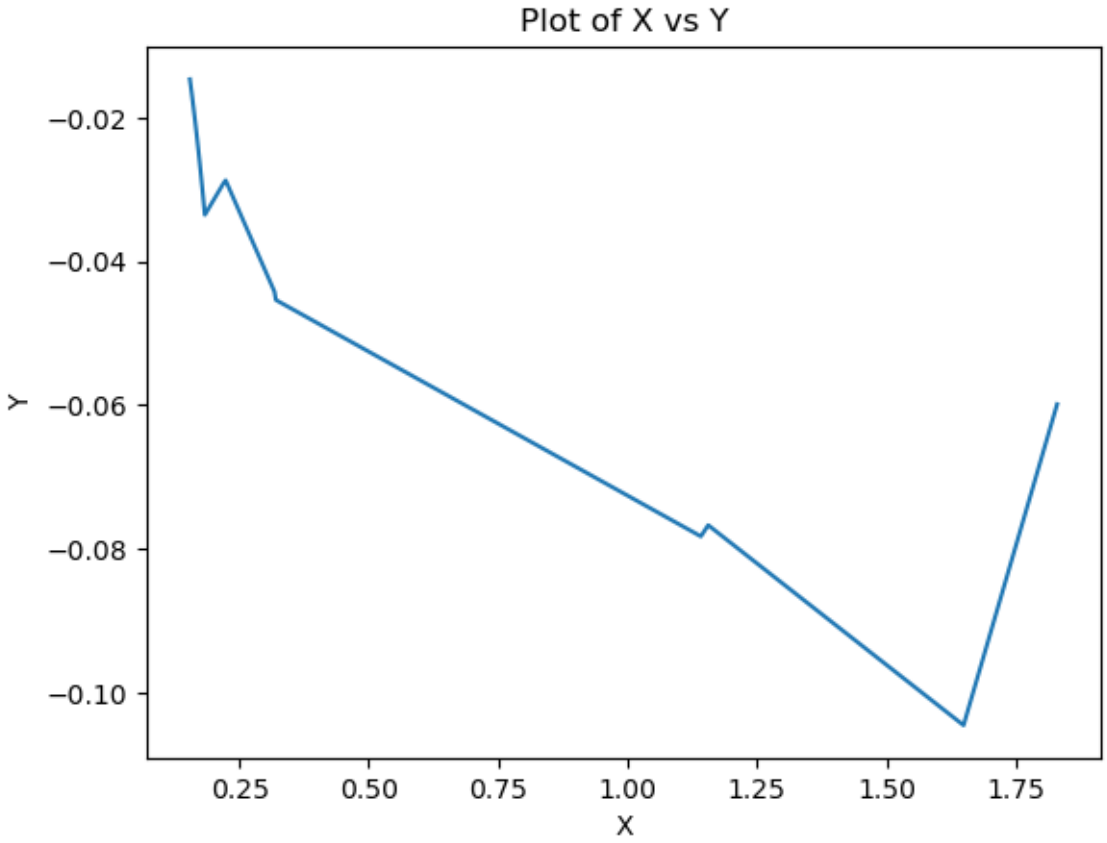
Bug 1 Algorithm has several limitations and challenges, some of which are:

1. It can get stuck in a loop while trying to navigate around an obstacle that it cannot pass through.
2. It is not optimal, which means that it may take longer than necessary to reach the goal.
3. It assumes that the robot has perfect knowledge of its environment, which is not always the case.
4. It can be difficult to implement in complex environments, where there are many obstacles and the robot needs to find the best path to the goal.
5. It may not work well in dynamic environments, where obstacles are moving or changing.

Overall, Bug 1 Algorithm is a simple and intuitive algorithm, but it has limitations and may not be suitable for all types of robotic navigation tasks.

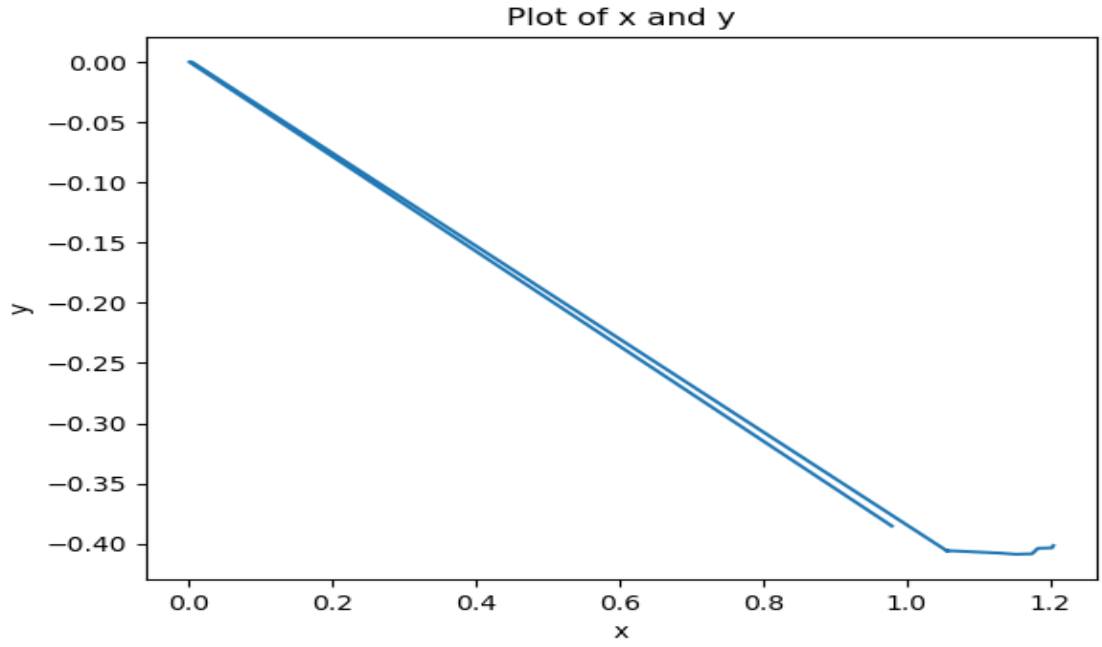
2.Output of the simulations, plotting of the Odom:

1. going the goal around the cylinder The Goal Location (1.85, -0,007)



2. going to the goal around the square:

The Goal Location (1.25,-0.35)



The difficulties The I have faced:

In the hardware implantation it is going to hit the obstacles.