OELP Report

on

Path Planning for wheeled mobile robot in partially known uneven terrain.

Name: C. Pavan Kumar Reddy Mentor: Sneha Gajbhiye

Objectives:

1. Information about the Robot kinematic model.
2. Open-loop control of the kinematic model.
3. Graphs of x(t) vs t and y(t) vs t.
4. Path planning Using the A* algorithm.
5. Path for the fixed obstacle environment or known environment.
6. Path for the random obstacle environment or unknown environment.

Current Work:

Information Extraction of a Partially Known Uneven Terrain Environment:

This section models the partially known uneven terrain environment considering its geomorphic features such as slope, step, and unevenness. Slope describes the inclining degree of the uneven terrain, where a greater slope value implies a more inclining uneven terrain that is more difficult for a wheeled mobile robot to pass. Step indicates the change rate of the vertical height of the uneven terrain within a certain horizontal distance. The larger the step value is, the more likely the wheeled mobile robot will overturn. The unevenness describes the jitter degree of the terrain's elevation, where a greater unevenness value would lead to a more unstable motion of the wheeled mobile robot.

The terrain's slope actually represents the angle between the terrain plane and the horizontal plane.

Robot Kinematic Model:

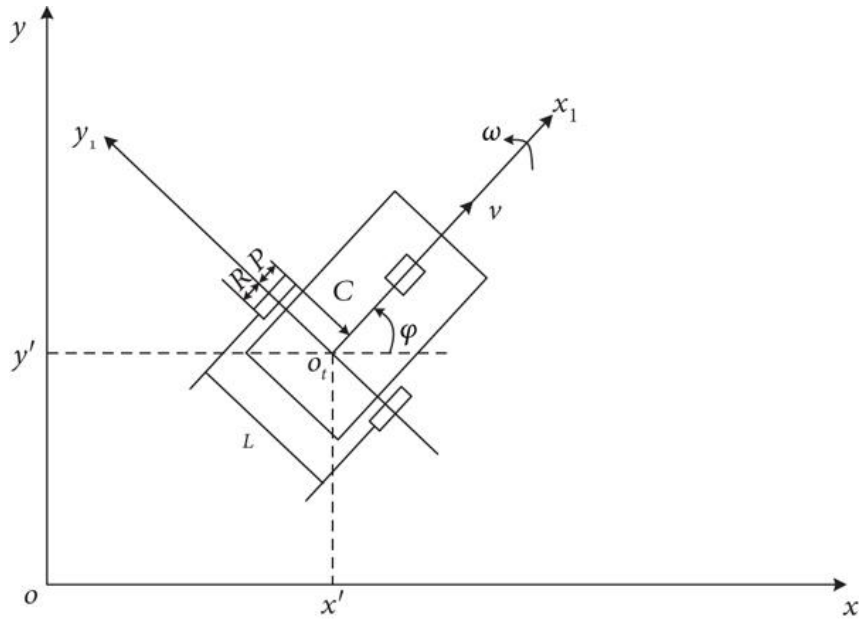


Figure-1 – mobile robot coordinate system

[https://www.hindawi.com/journals/wcmc/2021/2974839/]

As , we assumed that the kinematics of the four-wheel mobile robot is:

$dx/dt = v*\cos(\theta(t))$ --------(1)

$dy/dt = v*\sin(\theta(t))$ --------(2)

$d\theta/dt = w$ -------(3)

Where, v, w, x, y, and $\theta \in [-\pi, \pi]$ respectively represent the robot's tangential velocity, angular velocity, horizontal displacement, vertical displacement, and yaw angle.(In figure-1 θ is represented by φ)

In the simplified differential car model, the right and left wheel velocities are defined as vR and vL. Then, v and m in Equation can be calculated as $v = (vL + vR)/2$, $w = (vR - vL)/\lambda$,

where λ is the distance between the two wheels and $v \le vmax$, $|w| \le 2vmax/\lambda$. vmax is the robot's maximum speed.

Open-loop control of kinematic model:

And here we look into the scenario where the linear and angular velocities of the mobile robot is given and assumed to be constants. Therefore, it can be called as open-loop kinematic model.

Assuming the robot initial position is known that is , (x(0), y(0) , θ(0)) = (0.5,0.1,pi/2).

We apply this initial position to the robot in the form of differential equations (1) – (3) and applying velocities (v,w). By solving the differential equations for (x(t), y(t) , θ(t)) w.r.t time(t) and we get the plots of x(t) vs t and y(t) vs t.

As w=1, => θ = t+C (As, at t=0 , θ =pi/2)

We get, θ = t+pi/2

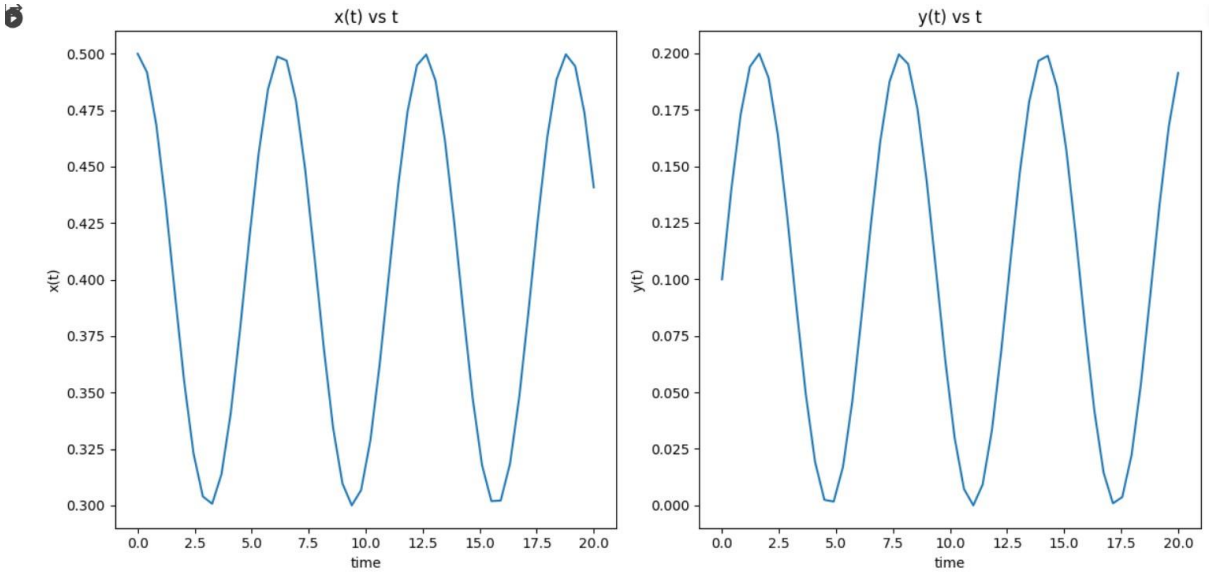
Therefore, dx/dt = v*cos(t+pi/2)= -vsin(t)

dy/dt = v*sin(t+pi/2)= vcos(t)

Therefore, through integration, we get x(t) vs t as a cosine graph and y(t) vs t as a sine graph. Below is the code for the plots of x(t)vs t and y(t) vs t :

```python
# the final plots of x(t) and y(t) using the kinematic equation
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

v = 0.1
w = 1
X0 = [0.5, 0.1, np.pi/2]

t = np.linspace(0, 20)

def model(x, t, v, w):
    dxdt = v * np.cos(x[2])
    dydt = v * np.sin(x[2])
    dthetadt = w
    dX = [dxdt, dydt, dthetadt]
    return dX

Y = odeint(model, X0, t, args=(v, w))

# Create two subplots, one for x(t) and one for y(t)
plt.figure(figsize=(12, 6))

plt.subplot(121)  # Left subplot for x(t)
plt.plot(t, Y[:, 0])
plt.xlabel("time")
plt.ylabel("x(t)")
plt.title("x(t) vs t")

plt.subplot(122)  # Right subplot for y(t)
plt.plot(t, Y[:, 1])
plt.xlabel("time")
```

The output plots are as follows:



We can clearly, see the above output plots as we have calculated theoretically.

Path Planning using A* algorithm:

Now I have worked upon creating an environment with obstacles and the robot should cross those obstacles and reach the specified goal state from the specified start state using the A* algorithm.

For that I have created 2 types of environment.

1)Robot reaching the specified goal state from the specified start state in the known obstacle environment.

2)Robot reaching the specified goal state from the specified start state in the unknown obstacle environment.

Case1:

In the type 1 , we already know the obstacles and we have a path that avoids the obstacles and the robot follow that path .(it is shown as the red dotted lines in the python implementation of code).

```
Enter the start node (row column):  0 1
Enter the goal node (row column):  2 2
Number of nodes explored are 4
[(0, 1), (1, 1), (1, 2), (2, 2)]
```

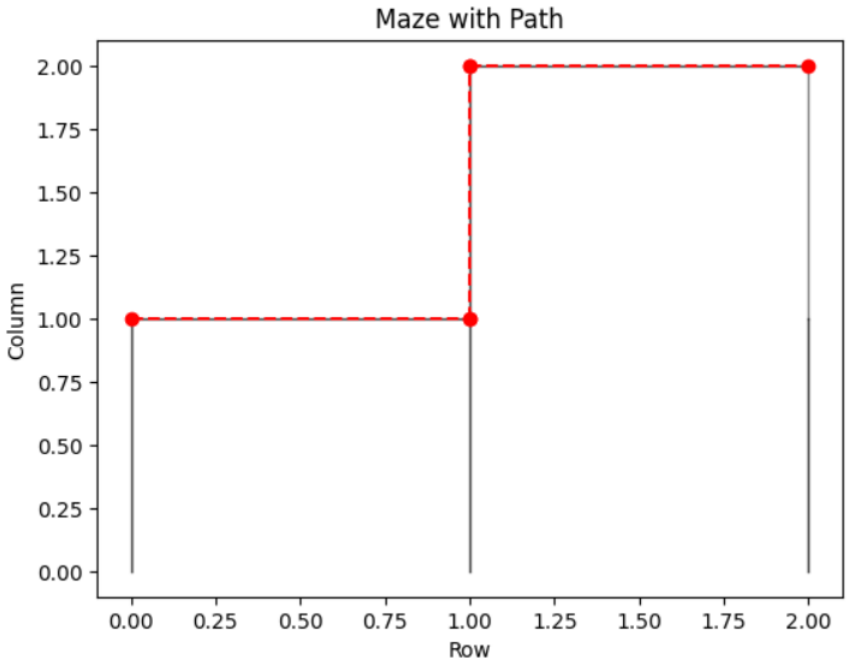


Figure-2- path for the fixed obstacle environment

The above is the output of the code implemented for the known/fixed maze of 2*2 and the start node and goal nodes are specified by me and we have the path shown in the diagram and also in the nodes explored list.

Case 2: In this case we have a randomly generated maze and we don't know the obstacles before and therefore it thinks rationally using A* algorithm using the Manhattan distance and reaches the goal state by avoiding the obstacles on the way of the path and during the travel.

(The path travelled by the robot is mentioned in the red dotted lines.)

```
Enter the value of n:  4
(0, 0) : [[0, 1]]
(0, 1) : [[0, 0], [0, 2]]
(0, 2) : [[0, 1], [0, 3]]
(0, 3) : [[0, 2], [1, 3]]
(1, 0) : [[2, 0], [1, 1]]
(1, 1) : [[1, 0], [2, 1]]
(1, 2) : [[2, 2]]
(1, 3) : [[0, 3], [2, 3]]
(2, 0) : [[3, 0], [1, 0]]
(2, 1) : [[1, 1], [2, 2]]
(2, 2) : [[2, 1], [1, 2]]
(2, 3) : [[1, 3], [3, 3]]
(3, 0) : [[3, 1], [2, 0]]
(3, 1) : [[3, 2], [3, 0]]
(3, 2) : [[3, 3], [3, 1]]
(3, 3) : [[2, 3], [3, 2]]
+--+--+--+--+
|           |
+  +--+--+  +
|  |     |  |
+  +--+  +  +
|  |     |  |
+  +  +--+  +
|  |        |
+--+--+--+--+
Enter the start node (row column):  2 2
Enter the goal node (row column):  1 3
Number of nodes explored are 12
[(2, 2), (2, 1), (1, 1), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (3, 3), (2, 3), (1, 3)]
```

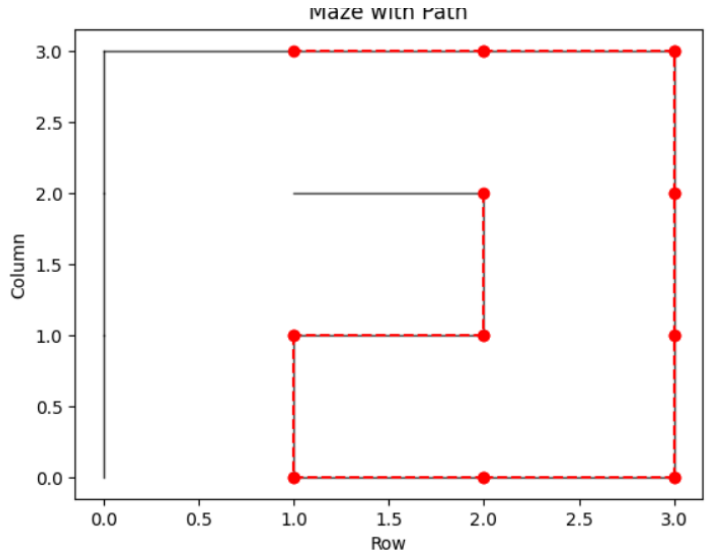


Figure-3- Mobile robot path in the random obstacle environment

The above is the output of the python code implemented with a random maze and it is specified as the input of n=4 by me , so a 4*4 random maze is created by me with the random obstacles and it rationally crosses those obstacles and reach the goal state from the start state which are also specified by me and the path travelled by it is mentioned in the red dotted lines.

If I again run the code and specify n=4 as input , it doesn't create the same maze . It creates a random maze with different obstacles. And if we specify the start and the goal states, It reaches the goal state if there is a path situated for sure.

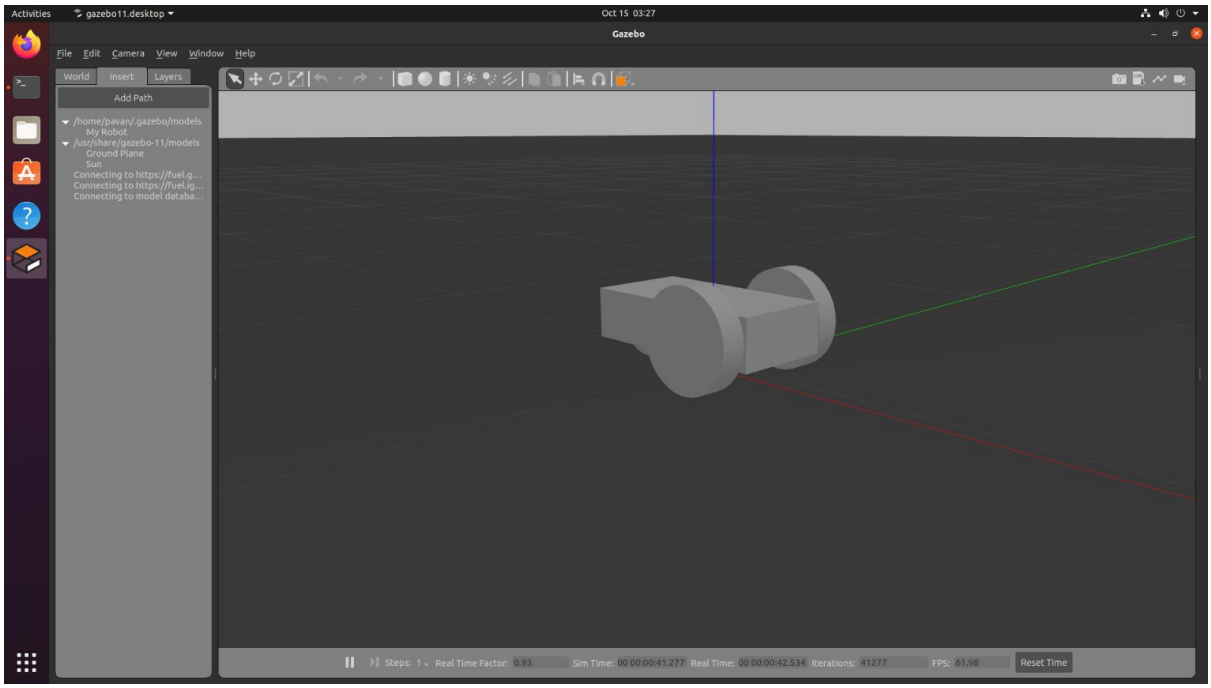
Hence , the motion is stated as the inertial motion w.r.t Earth. And it can be anywhere in the room if the maze is considered as a room.

So, we have defined the path travelled by the robot in a partially known uneven terrain which are the obstacles here.

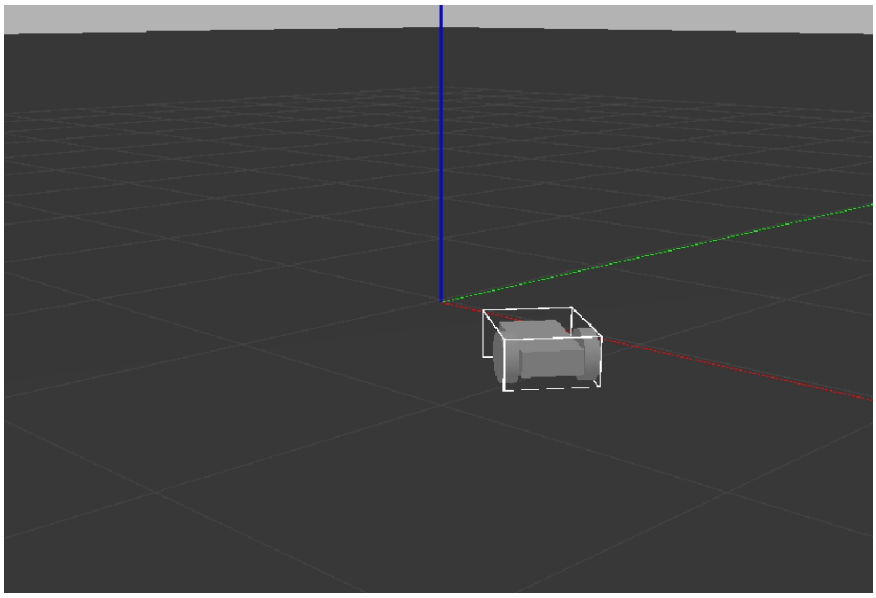
Future Work:

I also worked on the gazebo implementation.

I have installed the gazebo on the system and then simulated a mobile robot in it.



Above is the mobile wheeled robot generated in the gazebo and it moves as we give a force or velocity based or given the coordinates change too it moves a shown in below figure.



We can also integrate our python files with the gazebo using the ROS integration and we can see the 3D environment and clearly see how it works in the real life.

Signature of the Mentor