Robot Motion Planning (ITCS 6152)

Project Report on

Motion Planning for Unicycle robot Using Rapidly Exploring Random Trees

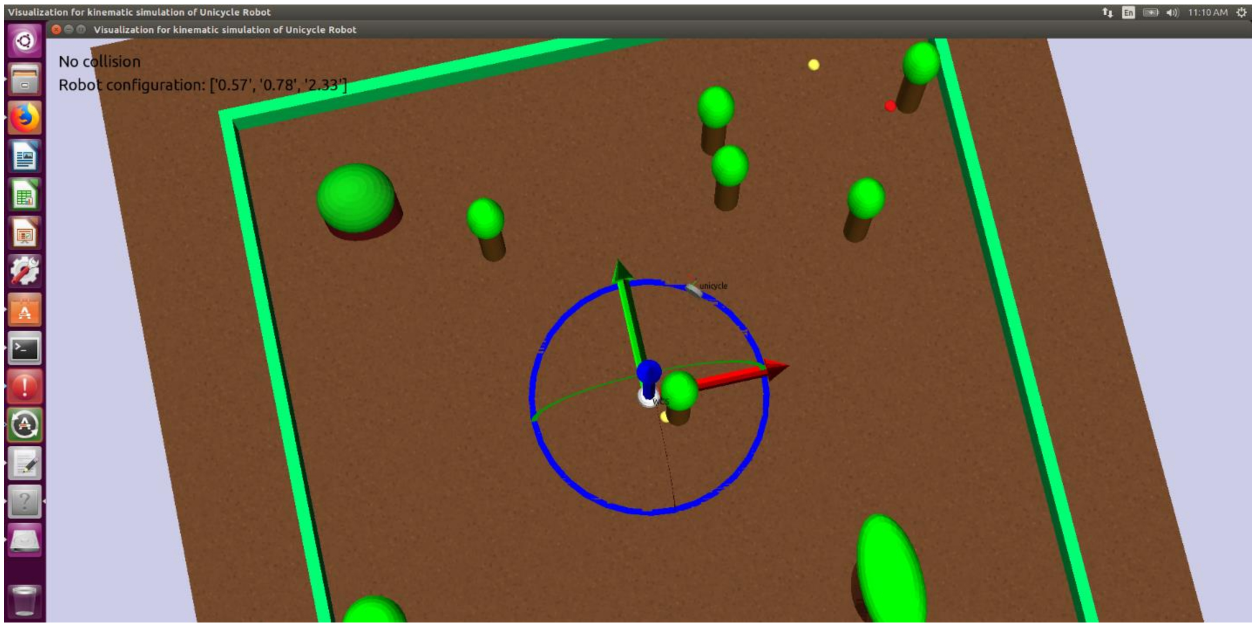
Student Name : Hima Bindu Sigili
Email : hsigili@uncc.edu
Student ID : 801023234

Contents:

Introduction:

Problem Statement:

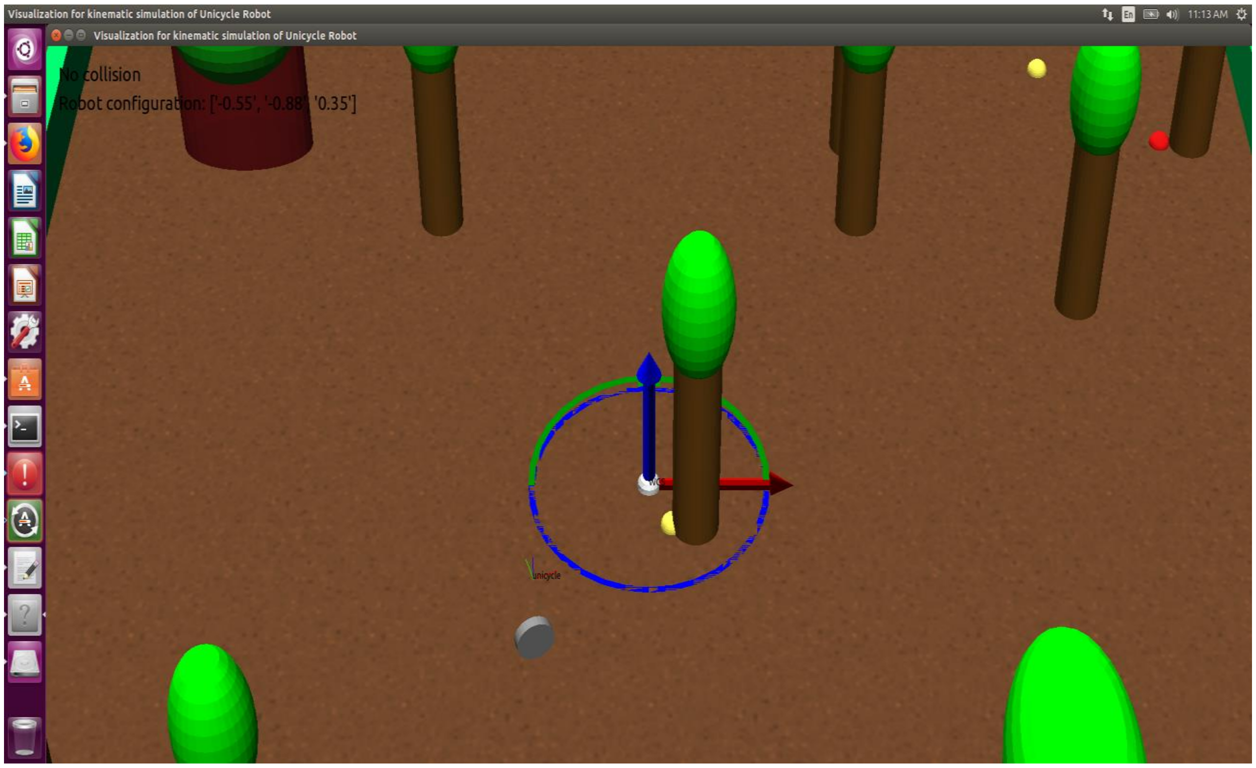
My Objective is to find a path for a non-holonomic single wheeled robot (Unicycle bot) moving in a fruit farm (static environment) with a lot of trees in between (circular/elliptical obstacles) such that my robot can effectively make its way towards ripened fruit (red fruit is my goal).


Visualization for kinematic simulation of Unicycle Robot
Visualization for kinematic simulation of Unicycle Robot
11:10 AM
unicycle

No collision
Robot configuration: ['0.57', '0.78', '2.33']

As my robot model should satisfy non-holonomic constraints it requires a kinodynamic planning which can be considered as motion-planning problem in a higher dimensional state space with both first-order differential constraints ($\dot{x}\sin\theta - \dot{y}\cos\theta=0$) and along with the obstacle-based global constraints. So, the goal is to design a feasible path that satisfies both global obstacle constraints and local differential constraints.

Robot Model:


Visualization for kinematic simulation of Unicycle Robot
11:13 AM
Visualization for kinematic simulation of Unicycle Robot
unicycle

No collision
Robot configuration: ['-0.55', '-0.88', '0.35']

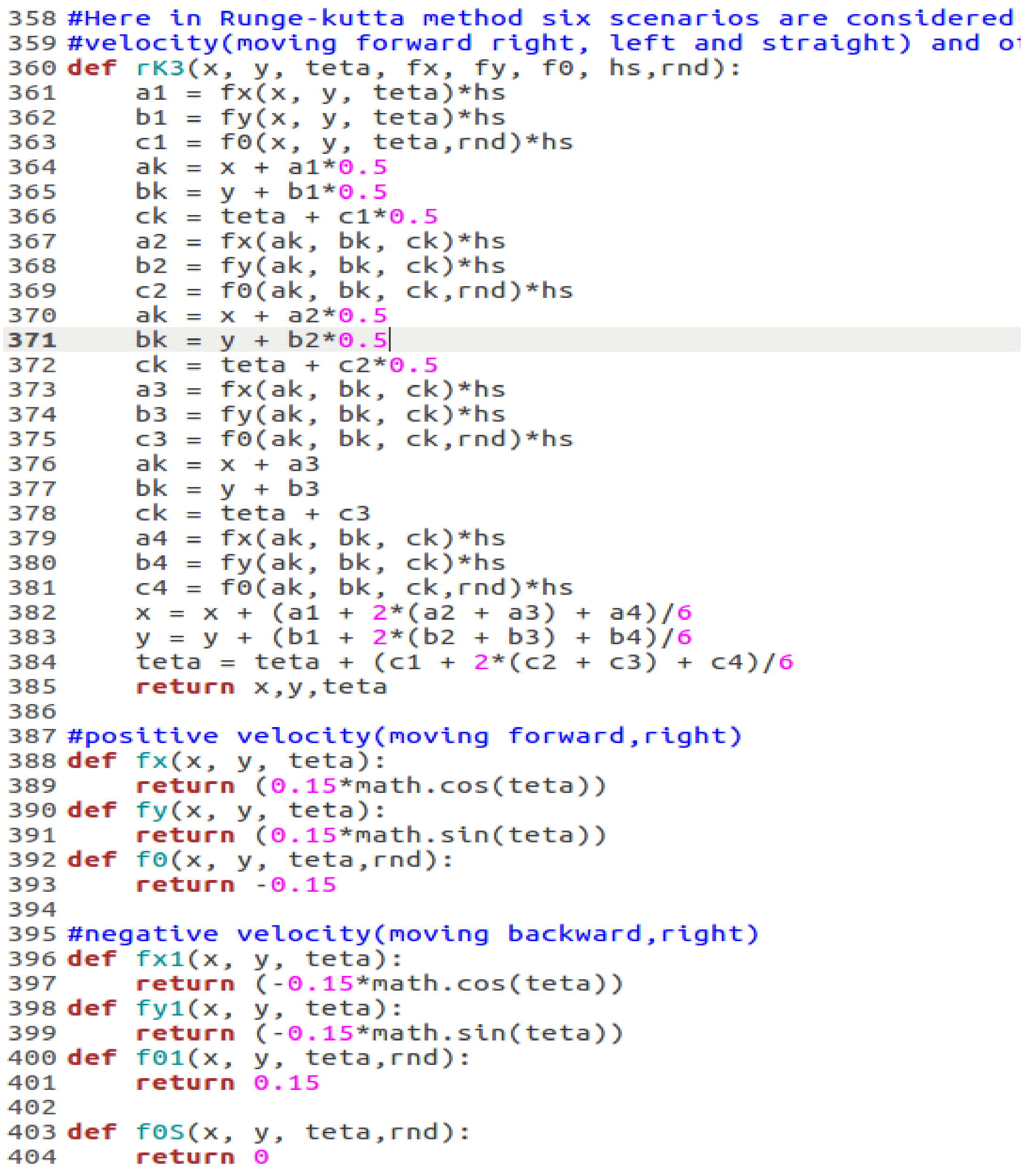
A unicycle type robot is a robot moving in a 2D world, having some forward speed but it cannot instantaneously move in every direction i.e., it must satisfy dynamic constraints. In other words, it is a non-holonomic system. Modeling unicycle type robots involves studying their kinematics as well as dynamics, similar with most of the physical systems. Kinematics modeling describes the trajectories that the mobile robots follow considering the speeds. The dynamics modeling accounts for the commanding forces and frictions defining the speeds.

Kinematic Model:

The configuration, $q = (x,y,\theta)$ is the robot position and orientation in world reference frame, and the pair (v,ω) is the input control encompassing the linear and angular velocities. The kinematic model of a unicycle type robot is usually described by a simple non-linear model:

$$\dot{x} = v\cos(\theta) \qquad \dot{y} = v\sin(\theta) \qquad \dot{\theta} = \omega$$

For simplicity I have considered linear velocity v to be a non-zero constant value i.e., v = 0.15 and for angular velocity ω as 0.15. By computing the nearest path point to the robot by applying six input controls i.e., I have used six cases for the input velocity v , +0.15 (positive for moving forward) and -0.15 (negative velocity for moving backward) and 0.15 angular velocity (moving counter clockwise) and -0.15 (moving clockwise) and 0 for moving in a straight path And I used Runge-Kutta method for interpolation with a step size of 0.05 for 100 iterations as shown in below figure.



```
#Here in Runge-kutta method six scenarios are considered
#velocity(moving forward right, left and straight) and o
def rK3(x, y, teta, fx, fy, f0, hs,rnd):
    a1 = fx(x, y, teta)*hs
    b1 = fy(x, y, teta)*hs
    c1 = f0(x, y, teta,rnd)*hs
    ak = x + a1*0.5
    bk = y + b1*0.5
    ck = teta + c1*0.5
    a2 = fx(ak, bk, ck)*hs
    b2 = fy(ak, bk, ck)*hs
    c2 = f0(ak, bk, ck,rnd)*hs
    ak = x + a2*0.5
    bk = y + b2*0.5
    ck = teta + c2*0.5
    a3 = fx(ak, bk, ck)*hs
    b3 = fy(ak, bk, ck)*hs
    c3 = f0(ak, bk, ck,rnd)*hs
    ak = x + a3
    bk = y + b3
    ck = teta + c3
    a4 = fx(ak, bk, ck)*hs
    b4 = fy(ak, bk, ck)*hs
    c4 = f0(ak, bk, ck,rnd)*hs
    x = x + (a1 + 2*(a2 + a3) + a4)/6
    y = y + (b1 + 2*(b2 + b3) + b4)/6
    teta = teta + (c1 + 2*(c2 + c3) + c4)/6
    return x,y,teta

#positive velocity(moving forward,right)
def fx(x, y, teta):
    return (0.15*math.cos(teta))
def fy(x, y, teta):
    return (0.15*math.sin(teta))
def f0(x, y, teta,rnd):
    return -0.15

#negative velocity(moving backward,right)
def fx1(x, y, teta):
    return (-0.15*math.cos(teta))
def fy1(x, y, teta):
    return (-0.15*math.sin(teta))
def f01(x, y, teta,rnd):
    return 0.15

def f0S(x, y, teta,rnd):
    return 0
```

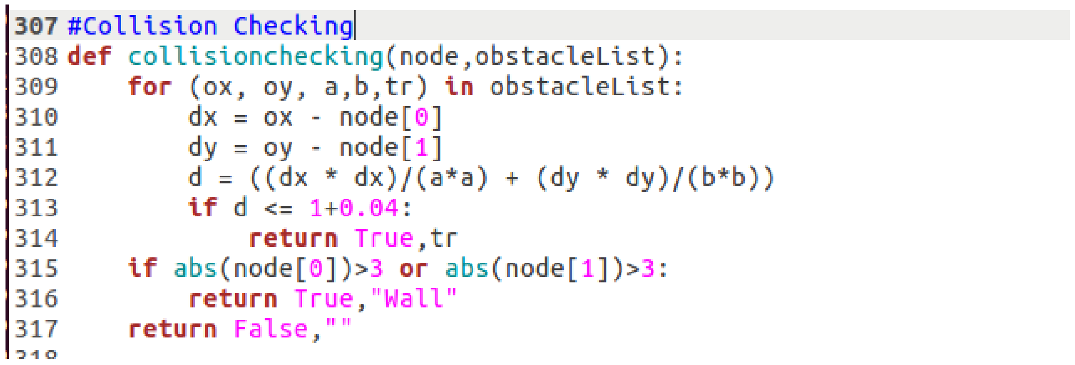
Figure 1: Runge-Kutta Implementation

Motion Planning Algorithm:

In my project I have implemented a sampling-based motion planning technique for the path planning problem. As robot is a non-holonomic robot I used RRT (Rapidly exploring random trees) motion planning algorithm. Since the environment is static, having the knowledge of the environment will be very useful in long run hence values (position of obstacles) are not provided dynamically. All the positions of the obstacles (trees and fruits) have been specified in the buildWorldnew.py which contains code to build the environment. Start and goal configurations can be changed dynamically, and simulation starts only if the start and goal configurations are collision free in c-space.

simpleWorld.xml contains the information on the robot that is being used and the size of the terrain. I have created my own robot model unicycle.rob by tweaking the already existing sphero.rob file (replacing the sphero.off file with cylinder_y.off file in the sphere.rob) After the robot model is ready we start executing MyRRT.py which is the main file that creates visualization for kinematic simulations. Inside this file RRTplanner method of RRT class is called to implement the algorithm that lies in a separate file called RRTUtil.py. Here in this class for 5000 samples we try to find a path from initial configuration to goal configuration using bidirectional RRT. User defined class called Node serves the purpose of a tree vertex and all the instances that are formed while finding the path are stored in a List data structure.

I have used my own collision detection method, as my obstacles are static I have stored their configurations as well as their names in a list so that if the robot collides with the obstacle this method checks if the sum of the radius of the circle or ellipse and the radius of the unicycle is less than the distance between the center of the obstacle and the center of the robot then return true indicating collision, if not less than then it returns false indicating no collision. Also, I should consider the walls (boundaries of the farm) as obstacles so I have checked the robot's x and y co-ordinates should be less than the x and y co-ordinates of the wall to check if the robot is collision free.

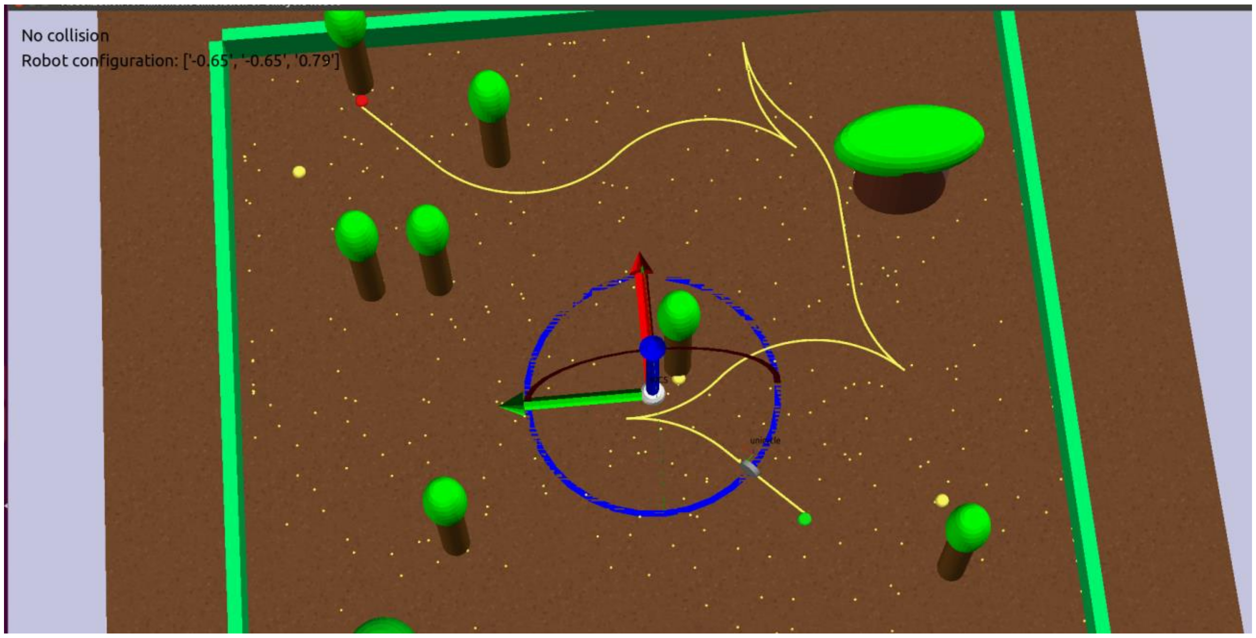

```
#Collision Checking
def collisionchecking(node,obstacleList):
    for (ox, oy, a,b,tr) in obstacleList:
        dx = ox - node[0]
        dy = oy - node[1]
        d = ((dx * dx)/(a*a) + (dy * dy)/(b*b))
        if d <= 1+0.04:
            return True,tr
    if abs(node[0])>3 or abs(node[1])>3:
        return True,"Wall"
    return False,""
```

To check the closeness of two configurations I have used a tolerance of 0.02 for position and 0.1 for orientation. To show the simulated motion of the robot from start to goal I have stored all the intermediate configurations in to a list.

Problems faced:

During the earlier stages of the project I have used single- directional RRT algorithm to find the path. But even after 5000 iterations it was unable to form a path from start to goal. So, I shifted to the Bi-directional RRT Implementation which took around 2500 iterations to reach the goal. Also getting used to Klampt took me a lot of time and effort. Even after finding the path I was unable to show the simulation. With the help of a research graduate in my class finally I was able to figure out and solve the issue.

Results:



```
No collision
Robot configuration: ['-0.65', '-0.65', '0.79']
```

videof.mp4

You can find a video showing the simulation in the home page of my website, link is mentioned below

Future Implementations:

As RRT algorithm will not promise an optimal solution I would like to go for improvised version of RRT i.e., RRT* a sampling-based algorithm for optimal motion planning. I will be implementing RRT* algorithm to my model to obtain an optimal path from start to goal configurations sooner. Also, I will do some research on the ball bot as well and will try to relate this ideology to my project. I will be extending my project to efficiently implement the path planning problem for a bicycle in future.

GitHub Link

https://github.com/HimaBinduSigili/FinalProject6152/

Website Link

https://himabindusigili.github.io/