RRT

October 29, 2020

1 RRT Algorithm

[ ]:

```
import argparse
import numpy as np
import plotly.graph_objects as go
import networkx as nx

def demo():
    np.random.seed(0)
    # States
    x = np.zeros(2)
    x_min = -5
    x_max = -x_min
    v_min = -1
    v_max = -v_min

    # Actions
    a = 0

    # System Dynamics
    A = np.array([
        [1, 1],
        [0, 1]
        ])
    B = np.array([
        [1/2],
        [1]
        ])

    def step(s, u):
        return A @ s + B.dot(u)

    # Goal State
    s_f = np.array([[0],[0]])
    print(s_f)
    # Goal Space
    allowable_dx = 0.5
```

```
allowable_dv = 0.1
# def goal_reached(s_curr):
#         if (np.abs(s_curr[0,0] - s_f[0,0]) < allowable_dx and np.abs()):

def goal_reached(s, s_f, dx=0.5, dy=0.1):
    return np.linalg.norm(s - s_f) <= np.linalg.norm(np.array([dx, dy]))

# Initial State
s_i = np.zeros((2,1))

s_i[0,0] = np.random.uniform(x_min, x_max)
s_i[1,0] = np.random.uniform(v_min, v_max)
print(s_i)

# Random Sample
def random_sample(p=0.05):
    theta = np.random.sample()
    if theta < p:
        return s_f
    s_rand = np.zeros((2,1))
    s_rand[0,0] = np.random.uniform(x_min, x_max)
    s_rand[1,0] = np.random.uniform(v_min, v_max)
    return s_rand

V = [s_i]
E = []

def nearest_vertex(v, V):
    return min([(i, x, np.linalg.norm(v - x)) for i, x in enumerate(V)],␣
↪key=lambda x: x[2])

# # Construct the graph
# def find_nearest_vertex(s_rand):
#     s_near = None
#     nearest_distance = np.inf()
#     for i in range(len(V)):
#         curr_distance = np.linalg.norm(V[i] - s_rand)
#         if curr_distance < nearest_distance:
#             nearest_distance = curr_distance
#             s_near = V[i]
#     return s_near

def drive_to(v, u):
    """Drive from v to u."""
    delta_p = v[0] - u[0]
    if delta_p > 0:
        a = -1
        elif delta_p < 0:
            a = 1
        else:
            a = 0
        return step(v, a)



    print("Starting simulation.")
    while True:
        s_rand = random_sample()
        nearest_index, s_near, distance = nearest_vertex(s_rand, V)
        s_new = drive_to(s_near, s_rand)
        V.append(s_new)
        E.append(( nearest_index, (len(V)-1) ))
        if goal_reached(s_new, s_f):
            return V,E




#if __name__ == "__main__":
 #   main()
```

2 Plot the resulting Graph

[27]:

```
from main import demo
import matplotlib.pyplot as plt
```

[2]:

```
V,E = demo()
```

```
[[0]
 [0]]
[[0.48813504]
 [0.43037873]]
Starting simulation.
```

[4]:

```
import networkx as nx
```

[61]:

```
#convert V ndarrays to tuples so format is hashable for G library
V_tuple=[]
for i in range(len(V)):
    x=V[i][0][0]
    v=V[i][1][0]
    x=round(x,2)
    v=round(v,2)
    Tuple=(x,y)
    V_tuple.append(Tuple)
```

[63]:

```
G=nx.Graph()
#Add the list of nodes:
G.add_nodes_from( V_tuple)
#print(G.nodes())
```

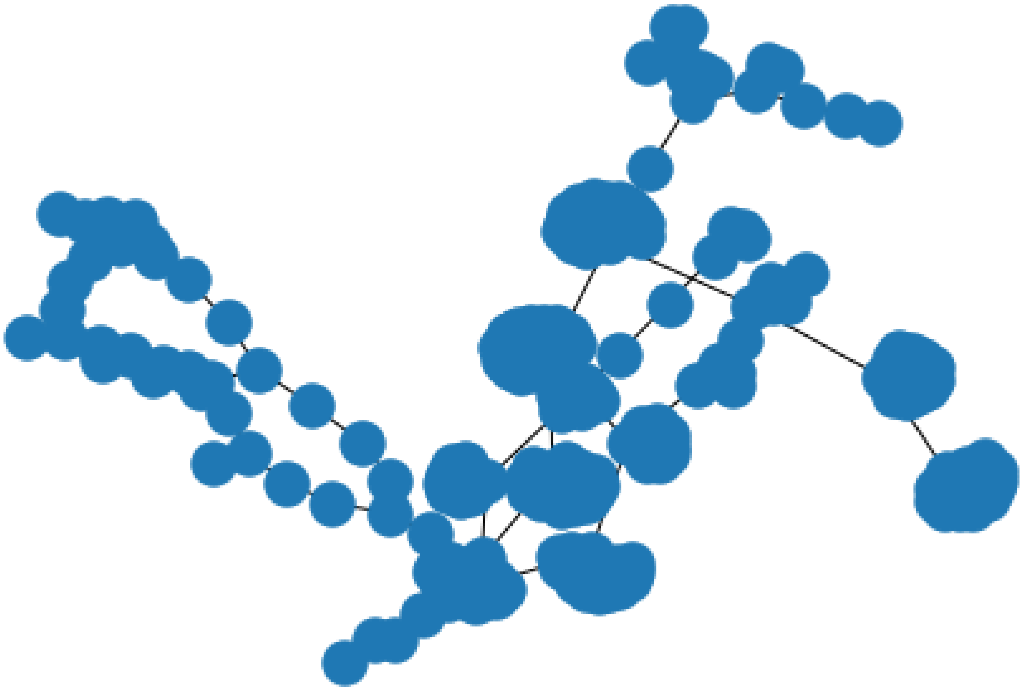
[77]:

```
# Add edges from the list of the edge tuples

for i in range(len(E)):
    s0=E[i][0]
    s1=E[i][1]
    edge=(V_tuple[s0],V_tuple[s1])
    G.add_edge(*edge)
```

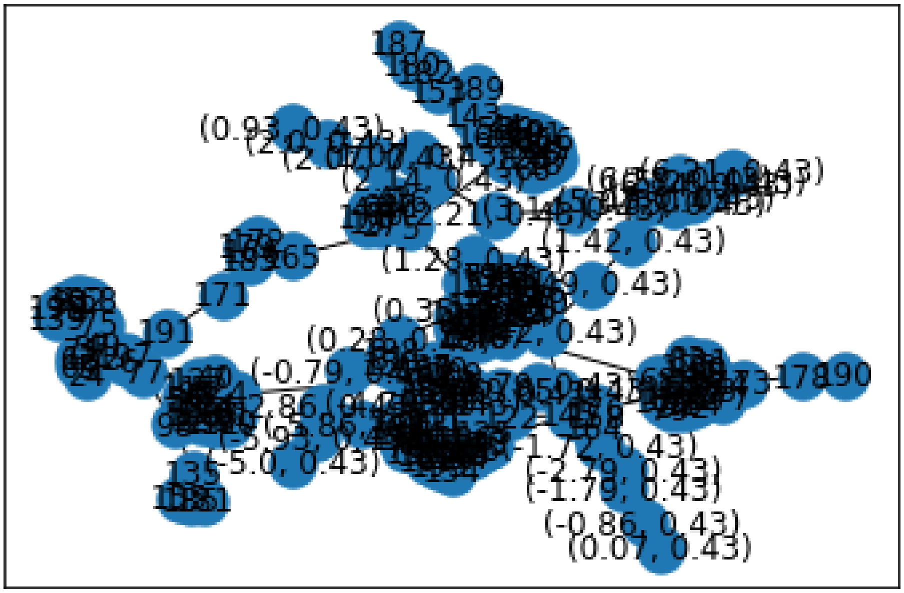
[79]:

```
#Plot the graph
nx.draw(G)
plt.savefig("RRT_Week4_Problem4.png") # save as png
plt.show() # display
```



[80]:

```
nx.draw_networkx(G)
plt.savefig("RRT_Week4_Problem4_rev2.png") # save as png
plt.show() # display
```


187
180
152
189
(0.93, 0.43)
(1.28, 0.43)
(1.42, 0.43)
178
165
171
191
77
24
135
(-0.79, 0.43)
(-5.0, 0.43)
(-1.72, 0.43)
(-2.79, 0.43)
(-1.79, 0.43)
(-0.86, 0.43)
(0.07, 0.43)
173
178
190

[ ]: