4/13/2022

AuE8240 Automotive Driving Technologies

Homework-4

Siddharth Thorat
\CU-ICAR

1. Trapezoidal Decomposition based motion planning?

- The simplest methods cell-decomposition is the trapezoidal decomposition.
- This method divides a free-space into trapezoids by imagining vertical upper and lower edges emanating from each obstacle vertex.

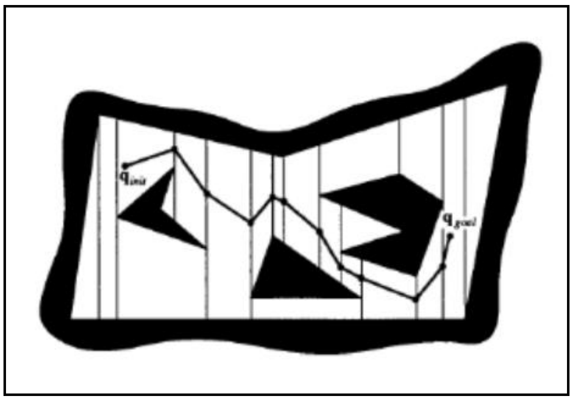


Figure 1

Problem for trapezoidal decomposition-based motion planning:

a. Trapezoidal decomposition is exact and complete, but they are not optimal.

How to address this problem?

- We need to address this problem using trapezoidal decomposition with visibility graph.
- We need to Construct a robust algorithm to perform this task is far from trivial.
- One approach would be that to use the standard polygon fill algorithm, except that only scan-line at the polygon vertices would be considered.
- This has the advantage that it is relatively easy to implement (some modifications of the standard polygon fill), but also has the disadvantage that it will produce a larger set of trapezia than necessary.
- We thus require an adaptation of the polygon fill algorithm in order to avoid constructing the unnecessary trapezia.
- The similar data structures are used, but with some important modifications.

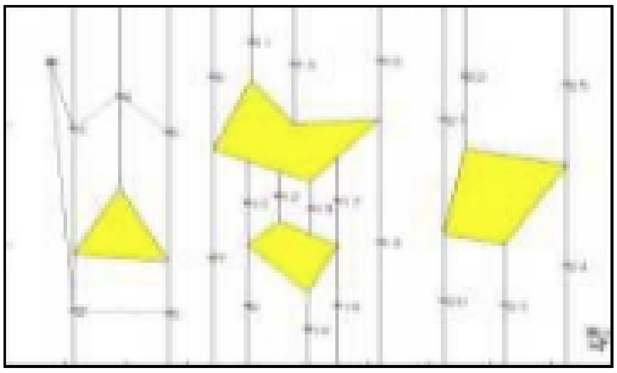


Figure 2

2. Describe the problem of RRT and how RRT* address the problem.

RRT:

- Rapidly-exploring Random Tree (RRT) is an algorithm designed to efficiently search nonconvex, high dimensional space by randomly building a space-filling tree.
- The tree is constructed incrementally from samples drawn randomly from the search space.

Problem of RRT:

- They cannot find the optimal solution for the optimal path from initial state to every state in the planning domain. Thus, this behaviour is not consistent.
- The RRT's are not asymptotically optimal because the existing state graph biases future expansion.

Addressing the problem by RRT*:

- For the above point mentioned, the RRT* overcomes this by introducing incremental rewiring of the graph.
- The advantages of the presented sampling technique are demonstrated with a new algorithm, Informed RRT*.
- This method retains the same probabilistic guarantees on completeness and optimality as RRT* while improving the convergence rate and final solution quality.
- We present the algorithm as a simple modification to RRT* that could be further extended by more advanced path-planning algorithms.
- We can show experimentally that it outperforms RRT* in rate of convergence, final solution cost, and ability to find difficult passages while demonstrating less dependence on the state dimension and range of the planning problem.

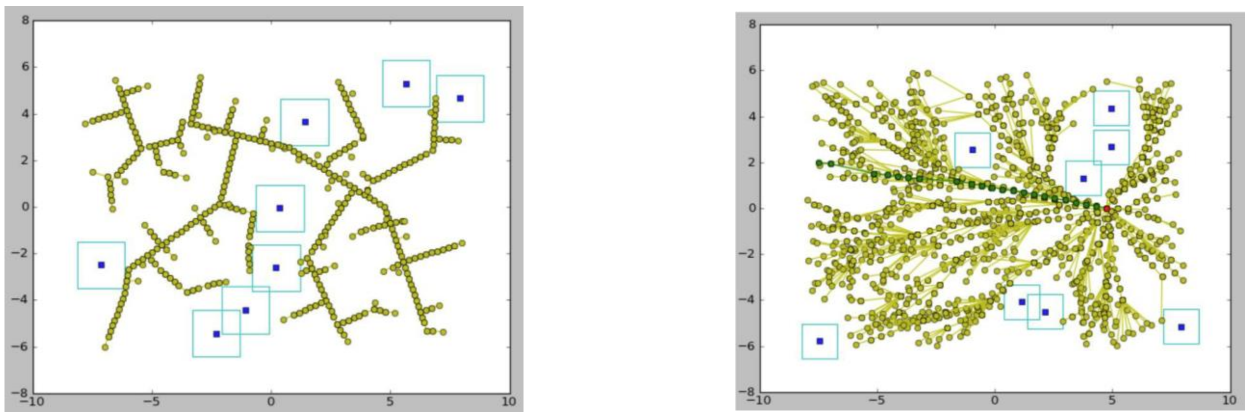
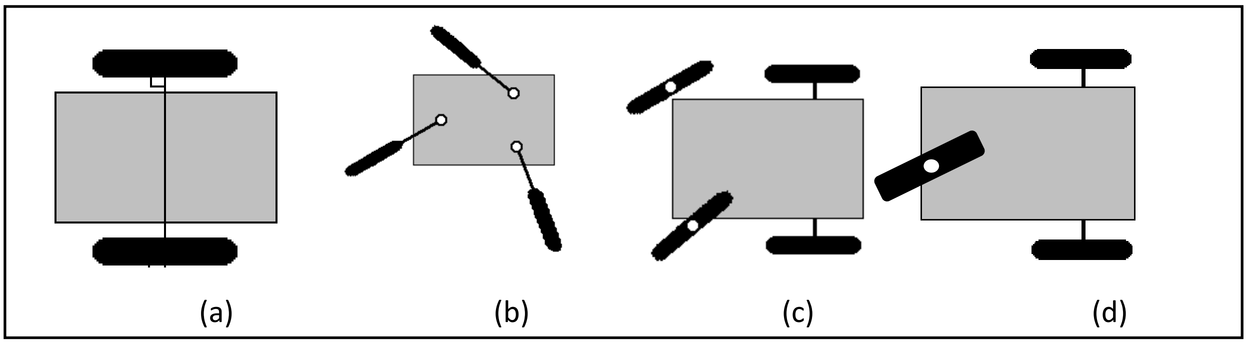
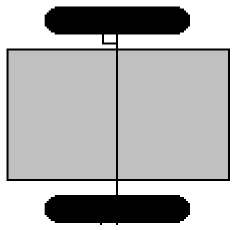


Figure 3

3. Give the degree of Manoeuvrability, degree of mobility and degree of steerability of the following vehicles.

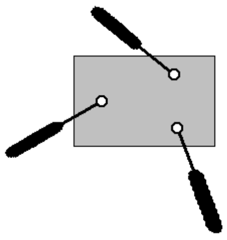


a.



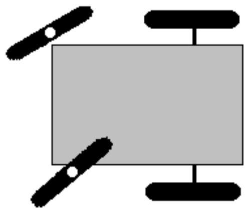
a. Degree of mobility -2
b. Degree of Steerability-0
c. Degree of Maneuverabilty-2

b.



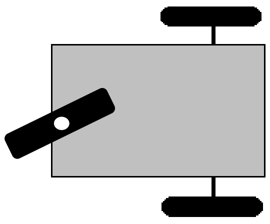
a. Degree of mobility -3
b. Degree of Steerability-0
c. Degree of Maneuverabilty-3

c.



a. Degree of mobility -1
b. Degree of Steerability-1
c. Degree of Maneuverabilty-2

d.



a. Degree of mobilty-1
b. Degree of Steerability-1
c. Degree of Maneuverabilty-2

4. Why is dynamic control better but harder than kinematic control?

- The kinematic control model will use speeds as their control signal to model and to control the systems.

Where,

Model: $x' = f(X,u)$

Control: $u = U(X^d,X)$ For the passenger car $u = (v, \varphi)^T$

- A dynamic control model will use the torque to control and model the systems. They tend to be more accurate and have good concern.
- We can ideally manipulate the speeds of the vehicle but we need to deal with the inertia of the body. Thus, dynamic control model is more difficult and harder but better than kinematic control.

Where,

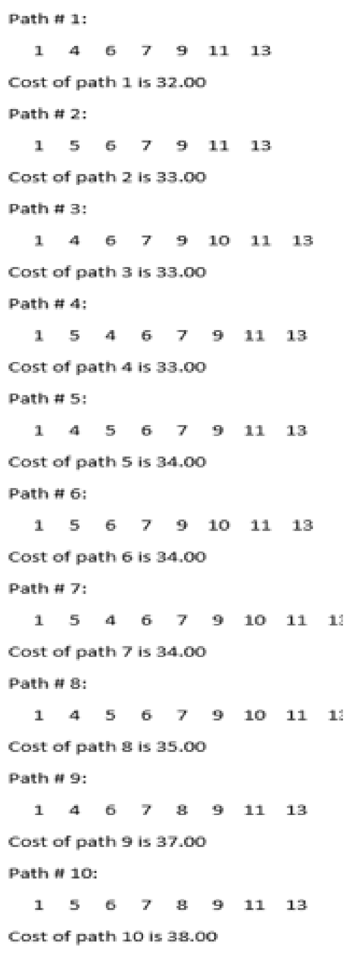
Model: $x' = f(X,u)$

Control: $u = U(X^d,X)$ For the passenger car $u = (\tau_d, \tau_s)^T$

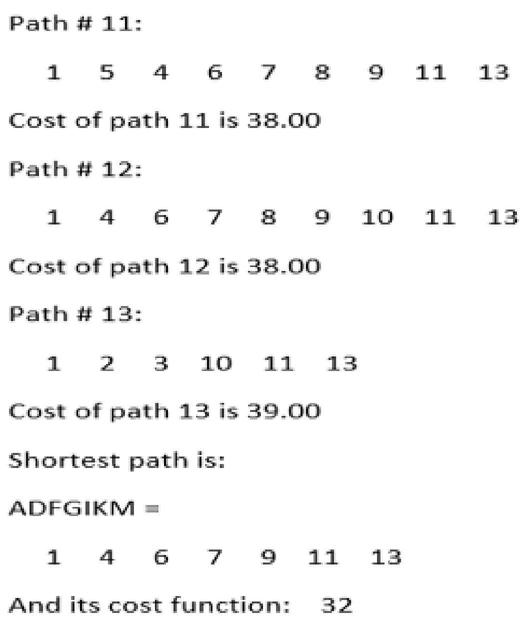
Problem 2

Build a graph to represent the map including Clemson (main campus) and CU-ICAR, where the vertices include Clemson, CU-ICAR, and intersections of roads (only considering roads: 85, 25, 185/29, 385, 123, 93, 178, 76) between Clemson and CU-ICAR, and the edges are road segments connecting each two vertices.

(1) Assign weights to each edge by the section road distance between two vertices, and derive the detailed process (including every step and their cost functions) of using Dijkstra's algorithm to find the route of shortest distance from Clemson to CU-ICAR. Modify the given A* MATLAB code to Dijkstra's algorithm and use it to solve this motion planning problem, print out the result, and compare with your derived result.



```
Path # 1:
   1    4    6    7    9   11   13
Cost of path 1 is 32.00
Path # 2:
   1    5    6    7    9   11   13
Cost of path 2 is 33.00
Path # 3:
   1    4    6    7    9   10   11   13
Cost of path 3 is 33.00
Path # 4:
   1    5    4    6    7    9   11   13
Cost of path 4 is 33.00
Path # 5:
   1    4    5    6    7    9   11   13
Cost of path 5 is 34.00
Path # 6:
   1    5    6    7    9   10   11   13
Cost of path 6 is 34.00
Path # 7:
   1    5    4    6    7    9   10   11   1
Cost of path 7 is 34.00
Path # 8:
   1    4    5    6    7    9   10   11   1
Cost of path 8 is 35.00
Path # 9:
   1    4    6    7    8    9   11   13
Cost of path 9 is 37.00
Path # 10:
   1    5    6    7    8    9   11   13
Cost of path 10 is 38.00
```

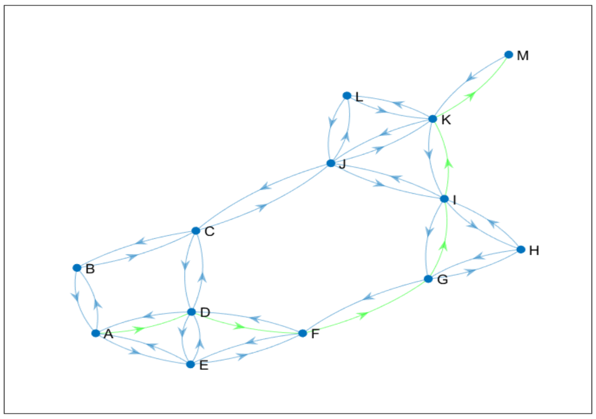


```
Path # 11:
   1    5    4    6    7    8    9   11   13
Cost of path 11 is 38.00
Path # 12:
   1    4    6    7    8    9   10   11   13
Cost of path 12 is 38.00
Path # 13:
   1    2    3   10   11   13
Cost of path 13 is 39.00
Shortest path is:
ADFGIKM =
   1    4    6    7    9   11   13
And its cost function:    32
```

(2) Assign weights to each edge by the driving time (e.g., section road distance/speed limit) between two vertices, assign heuristic time to each node (e.g., Euclidean distance to the destination/max. speed), and derive the detailed process (including every step and their cost functions) of using A* algorithm to find the route with shortest time from CU-ICAR to Clemson. Use the given A* MATLAB code to solve this motion planning problem, print out the result, and compare with your derived result.

path = {'A'}   {'D'}   {'F'}   {'G'}   {'I'}   {'K'}   {'M'}

cost = 32



Plot 1

Problem 3: (Motion Control)

A vehicle is driving along a straight lane which center line is represented by y=1. The vehicle needs to switch to an adjacent straight lane which center line is represented by y=2. The units are all meters. A simplified discrete vehicle model is given by

$$\begin{bmatrix} x(i+1) \\ y(i+1) \\ \theta(i+1) \end{bmatrix} = \begin{bmatrix} x(i) \\ y(i) \\ \theta(i) \end{bmatrix} + \begin{bmatrix} v * cos\theta(i) \\ v * sin\theta(i) \\ tan\varphi * \Delta t * v/L \end{bmatrix} \Delta t$$

where the vehicle baseline L=1 m, the control sampling time is chosen as 0.01 second, and the vehicle speed is a constant v=1.2 m/s.

1. Use Pure Pursuit Method and Stanley Method to design a lane switching controller respectively to calculate the vehicle steering angle ϕ.

From the given equation:

$$\begin{bmatrix} x(i+1) \\ y(i+1) \\ \theta(i+1) \end{bmatrix} = \begin{bmatrix} x(i) \\ y(i) \\ \theta(i) \end{bmatrix} + \begin{bmatrix} v * cos\theta(i) \\ v * sin\theta(i) \\ tan\varphi * \Delta t * v/L \end{bmatrix} \Delta t$$

a. By using Pure Pursuit Method:

- **Goal:** Make the vehicle track a desired path $y = 2$

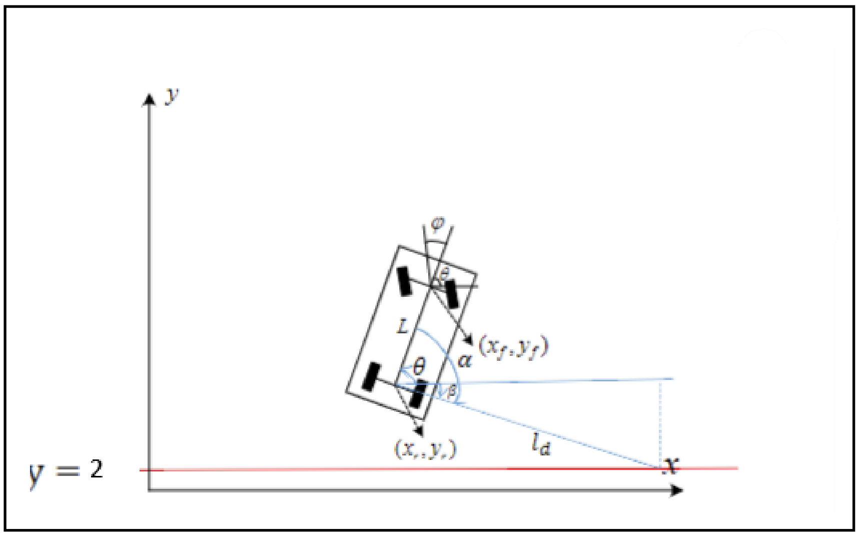


Figure 4

α: angle from vehicle orientation to look-ahead distance l_d
β: angle from x axis to look-ahead distance l_d
θ: vehicle orientation angle (from x axis to vehicle orientation)

From above figure:

$$\beta = \sin^{-1}\left(\frac{2 - y_R}{l_d}\right)$$

$$\alpha = \beta - \theta$$

$$\alpha = \sin^{-1}\left(\frac{2 - y_R}{l_d}\right) - \theta$$

ontroller -

$$\varphi = \tan^{-1}(\kappa L) = \tan^{-1}\left(\frac{2L \sin \alpha(t)}{l_d}\right)$$

where $\alpha(t) = \sin^{-1}\left(\frac{2 - y_R(t)}{l_d}\right) \theta(t)$

$$l_d = k v(t)$$

b. By using Stanley Method:

- **Goal:** Make the vehicle track a desired path $y = 2$

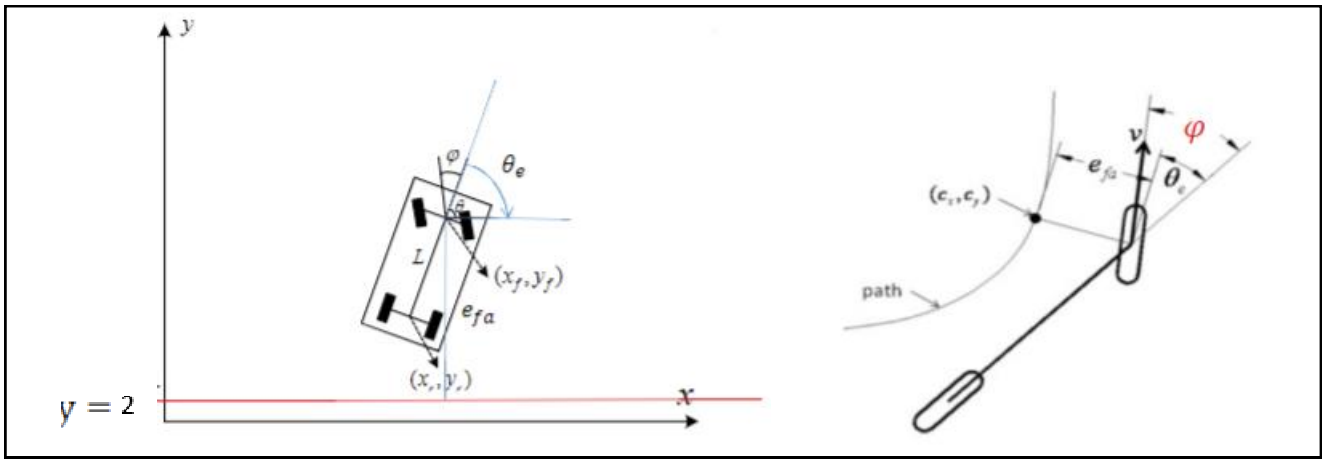


Figure 5

Where,

θ_e: angle from vehicle orientation to desired orientation
e_{fa}: distance from front axel to desired path (direct to left-hand side of vehicle->positive; direct to right-hand side of vehicle-> negative)

From the above figure:

$$\theta_e(t) = 0 - \theta(t) = -\theta($$

$$e_{fa} = 2 - y_f(t)$$

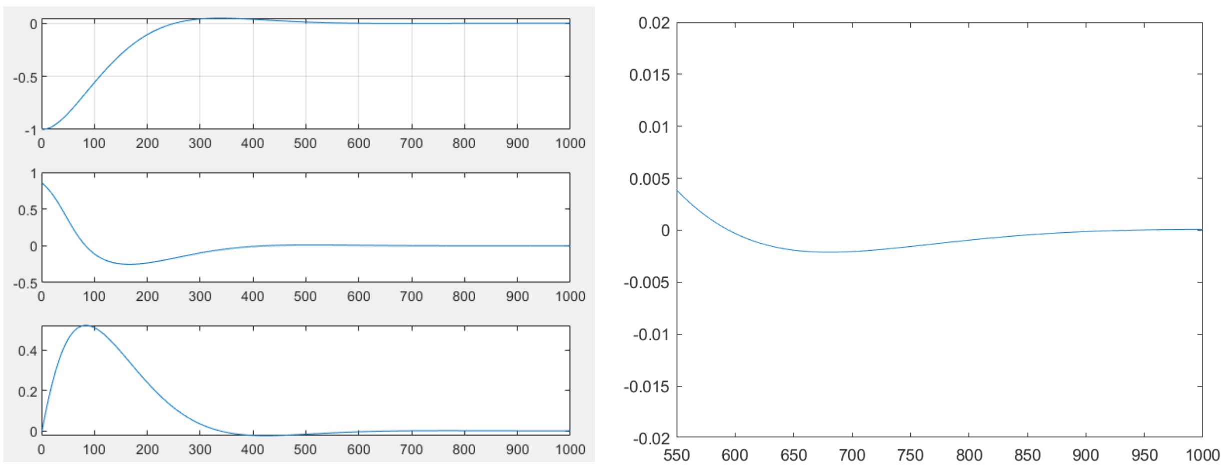
Controller →

$$\varphi(t) \quad \theta_e(t) + \tan^{-1}\left(\frac{k e_{fa}(t)}{v(t)}\right)$$

here $\theta_e(t) = -\theta(t)$

$$e_{fa} = 2 - y_f(t)$$

2. Implement the Pure Pursuit method using MATLAB and tune the control parameter k to make the lane changing finished (lane tracking error < 0.01 m) with seconds and the overshoot is within 0.05 m. Give the parameter and plot the desired-lane tracking errors, steering angles and vehicle orientations during the lane switching process.



Plot 2

```
MATLAB Code:

% HW4: Pure Pursuit control for lane switching
clear all;
yd = 2;                  % desired path
x0 = 0;                  % initial x
y0 = 1;                  % initial y = 1,2
theta0 = 0;          % initial theta
N = 1000;                % calculate 1000 steps
x = zeros(1,N);      % creat x vecotor with all 0 elements
y = zeros(1,N);      % creat y vecotor with all 0 elements
theta = zeros(1,N); % creat theta vecotor with all 0 elements
phi = zeros(1,N);    % creat phi vecotor with all 0 elements
x(1) = x0;               % record the first x value
y(1) = y0;               % record the first y value
theta(1) = theta0;   % record the first theta value
T = 0.01 ;                 % sampling time
```

```
v = 1.2 ;           % speed
k = 1.1 ;           % control gain (need to tune this)
L = 1.0;            % vehicle baseline
for i=1:1:N-1
    ld = k*v;                                        % calculate lookaead distance
    alpha = asin((yd-y(i))/ld)-theta(i);         % calculate alpha angle
    phi(i) = atan(2*L*sin(alpha)/ld);              % calculate steering angle phi
    x(i+1) = x(i)+v*cos(theta(i))*T;             % calculate next x based on current 
control input
    y(i+1) = y(i)+v*sin(theta(i))*T;             % calculate next y based on current 
control input
    theta(i+1) = theta(i) + (v*tan(phi(i))/L)*T; % calculate next theta based on current 
control input
end
figure(1)
subplot(3,1,1)
plot(y-yd);                 % plot lane switching errors
grid on;
subplot(3,1,2)
plot(phi);                  % plot steering angles
subplot(3,1,3)
plot(theta)                 % plot vehicle orientations
figure(2)
plot(x,y);
max(y-yd)           % overshoot
plot(y-yd)          %tracking error after 6 seconds
ylim([-0.02 0.02])
xlim([550 1000])
```

References:

1. P.J. From, J.T. Gravdahl, K.Y. Pettersen
   Vehicle-manipulator systems: modeling for simulation, analysis, and control
   Advances in industrial control, Springer London, London (2014)

2. http://motion.cs.illinois.edu/RoboticSystems/GeometricMotionPlanning.html#:~:text=One%20of%20the%20simplest%20cell,emanating%20from%20each%20obstacle%20vertex.

3. https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378#:~:text=The%20robotic%20path%20planning%20problem,which%20is%20not%20necessarily%20discretized

4. https://clemson.instructure.com/courses/156553/files/folder/Lectures/Supplementary%20Slides%20for%20Exam%202?preview=13004035

5. https://clemson.instructure.com/courses/156553/files/folder/Lectures?preview=12898911