https://www.overleaf.com/3572957318ybzgqspyrysx

RT-RRT*
https://dl.acm.org/doi/pdf/10.1145/2822013.2822036
https://users.aalto.fi/~hamalap5/FutureGameAnimation/p113-naderi.pdf

Paolo F. et. al Motion Planning inDynamic EnvironmentsUsing Velocity Obstacles
https://journals.sagepub.com/doi/epdf/10.1177/027836499801700706

Reif, J., and Sharir, M.1985. Motion planning in the presence of moving obstacles
https://dl.acm.org/doi/pdf/10.1145/179812.179911

ST-RRT*
https://arxiv.org/pdf/2203.02176.pdf

From MATLAB on A* and RRT
Path Planning with A* and RRT | Autonomous Navigation, Part 4

Random tree, RRT and RRT*
https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

[RRT-star & (various path planning algorithms)]
Python:
https://github.com/zhm-real/PathPlanning ⇐ selected!
C++:
https://github.com/vss2sn/path_planning
https://github.com/rishabh1b/RealTimePathPlanning (RT-RRT*) ⇐⇐⇐
rrt*
https://github.com/motion-planning/rrt-algorithms

Our Git repo: https://github.com/rahul-murthy/Sem-Robo_Project.git

[TODO]
1. choose github repo & Choose language
-> test those out and see if they provide testing env.
1'. Each of us should clone the repository on our local devices,
Test it out.
---
2. make the obstacles move [Sid]
2'. Generate obstacles at given location and velocity [Sid & Don]

#if professor approves ==========
3. set the robot shape(circle),
3'. Collision check with moving obstacles(circle to circle)
3''. Collision check with maze (circle to rectangle)
#else ==========
3. set the robot shape(rectangle), turning radius [Rahul]
3'. Modify collision checking
⇒ Rectangle(robot, can rotate) - Circle collision(moving obstacle) [Sid]
⇒ Rectangle(robot, can rotate) - Rectangle(static obstacles) [Don]
#endif ==========

4. make the simulation fail when collision happens. [Don]
(2'. set the obstacle shapes [Sid])

4' Check what we need for predictive collision checking. [Sid]
Variables: time variable for Tree Nodes, velocity for the moving obstacles,
Initial location of the moving obstacles, etc.
Methods: need some function to pass the obstacles' velocity information to the robot.
Etc.
Any other things that need to be added.

Algorithm 1 predictiveCollisionCheck(C,V,L,S)

Input: C(robot's configuration state),
V(a set of moving obstacles' velocity vectors),
L(initial locations of the moving obstacles),
S(shapes of the moving obstacles)
Output: $IsCollision$ (collision happens or not)

```
procedure PREDICTIVECOLLISIONCHECK(C, V, L, S)
    t = getTimeFromStartTo(C)
    for each obstacles O_i do
        find a time interval [t1:t2] that O_i collides with C
        if (( t1 ≤ t ) & ( t ≤ t2 )) then
            return true
        end if
    end for
    return false
```

// void RTRRTstar::expandAndRewire(std::list<Nodes>& nodes, const std::list<obstacles*>& obst)

5. create predictive collision checking
6. Implement Phase1 (path planning before the robot moves)
   6'. Implement Phase1 with rewiring.

---

7. Make the obstacles change their velocity at random time.
8. Start working on rewiring based on 7.
(End of thanksgiving)

Git install:
https://github.com/git-guides/install-git
Git clone repository:
https://www.educative.io/answers/how-to-clone-a-git-repository-using-the-command-line
Github
https://github.com/rahul-murthy/RT-RRTstar-algorithm.git
*Collision Checking:
https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169

https://www.youtube.com/watch?v=MvlhMEE9zuc

[Nov 19] Sid & Don
Separate Axis Theorem (theory)
https://www.gamedev.net/tutorials/_/technical/game-programming/2d-rotated-rectangle-collision-r2604/
(Code) https://gist.github.com/nyorain/dc5af42c6e83f7ac6d831a2cfd5fbece

[RT-RRT* Sampling and expanding(collision checking) for Phase1]
1. Theta is dependent on x and y.
2. we sample x_j, y_j ==> theta_j (not the theta itself, but new velocity)
[Need code for this] in RTRRTstar::expandAndRewire
Get new velocity for j,
||v|| / (sqrt((x_j-x_i)^2 +(y_j-y_i)^2)) * <x_j - x_i, y_j - y_i>
Additionally, every Node needs velocity
2D rotation: https://www.cuemath.com/algebra/rotation-matrix/

3. find the nearest neighbor x_i, y_i which is already in the tree.
calculate euclidean dist 'len' from x_i, y_i to x_j, y_j
delta t = len / constant speed value
t_j = (t_i + delta t).
[Need code for this]
Every Node needs a time variable.
Function for calculating delta t is needed

4. collision checking
1) rotation. (the collision checked area is a circle.)
center == robot current center
radius == 1/2 diagonal of the robot (constant)
for all moving obstacles (circles)
get t_j
get moving_obstacle_position at time (t_j)
[Need code for this]
First, we need obstacles' initial location in the moving obstacle object.
(which would get updated when the obstacle's velocity changes <= needed in phase 2)
Second, with the initial location information & t_j value, A function that passes temp obstacles for predicted location of the obstacle at time t_j is needed.
do (circle to circle collision checking)
[Need code for this]
With radius information of (½ diagonal of the robot) & (moving obstacle's radius)
Check the distance between the robot's center and the obstacle's center is larger or equal to the sum of (½ diagonal of the robot) and (moving obstacle's radius).
for all static maze (rectangles)
do (circle to rectangle collision checking)

[Need code for this]
Maybe the existing code that calculates perpendicular distance between a line and the center of a circle could be used. (Need some more time to think)

2) moving forward (the collision checked area is a rectangle.)
get the path area
(Left Rear = Left Front of Current robot
Right Rear = Right front of the current robot
Left front = Left Front when the robot's center reaches candidate position
Right front = Right Front when the robot's center reaches candidate position)
[Need code for this] function that returns the rectangle shaped collision area. (four points)

for all moving obstacles (circles)
get t_j
get moving_obstacle_position at time (t_j)
[Need code for this] same as 4. 1)
do (rectangle to circle collision checking)
[Need code for this] same as 4. 1)

for all static maze (rectangles)
do (rectangle to rectangle collision checking)
[Need code for this]
Separating axis theorem.
Calculate for four axes.

5. Add the sampled point

6. the robot will rotate to match the theta2 <= This is done automatically when velocity of the robot changes
#if openframework changes orientation immediately <===== need to check this.
no need to consider t_rot
#else
// set the rotation time as contant (t_rot)
// need to wait some time to make all the rotation take the same amount of time. (t_rot)
#endif

7. move forward until the center of the robot reaches x2, y2
#if openframework changes orientation immediately <===== need to check this.
The speed of the robot is constant.
[Need code for this] I think the current code sets this as a constant, but I need to check.
#else
//The robot stops when rotating (speed == 0).
//when the robot moves, the speed is constant.

#endif

Nov 20 [Rahul, Sid]

Check the path for both goal points [Need to change the RT-RRT* algorithm to accept two goals]

1. Robot samples the points in the workspace.
   If it finds path for one goal region,
   It continues to sample the points. [Need a code for this]
2. Find the path for the 2nd goal region.
   If it finds the path for 2nd goal, select the goals
3. Select the goal point based on the length of the path found. [Need a code for this]
4. Compare the length of path to both goals (not Euclidean Distance).
   If the len_G1 is < len_G2 then the robot selects the path to the goal region 1.
   else if len_G1 > len_G2 then the robot selects the path to goal region 2.
5. Optimize the path when the robot moves.

Nov 25 [Rahul, Sid, Don]

<Phase II.>
Moving Obstacles:
[Need a code for this]
Need to change the velocity of the obstacles.
{Instantly change velocity.}

Option1. Set time and velocity for changes when creating the moving obstacle object

```
Struct changes{
        Double time;
        ofVec2f velocity;
}
class movingObstacle{
        …
        Changes arrChanges[MAX_CHANGE_COUNT] =
        {       // time // velocity
                { 10,    {1, 0}},
                { 20,    {0, 1}},
                { 200,   {0, -1}},
        }
}
// somewhere with possibly a function with a different name
updateObstacle(){
        If (current time >= nextChangeTime)
        {
```

```
                    ofVec2f newVelocity = getNextVelocity(arrChanges);
                    movingObstacle->UpdateVelocity(newVelocity);
                    movingObstacle->UpdateInitialLocation(current location);
                }
                nextChangeTime = getNextChangeTime(arrChanges);
            }

        // Implement Option2 after debugging is done using option1.
        Option2. Randomly change the velocity once in a time interval that is a constant.
            updateObstacle(){
                If (current time >= nextChangeTime)
                {
                    ofVec2f newVelocity = getRandomVelocity();
                    movingObstacle->UpdateVelocity(newVelocity);
                    movingObstacle->UpdateInitialLocation(current location);
                }
                nextChangeTime = getNextChangeTime();
            }
        // The robot can retrieve this information by going through obstacles' velocity variables.
```

[Rewiring]

[Need a code for this] Need to update nodes when it gets obstructed by moving obstacles

[Need a code for this] Turning radius

limit the sampled theta value (-PI: PI]=> (-theta_limit:theta_limit]