

# Robotic Navigation and Exploration

## Lab1: Path Planning

Min-Chun Hu [anitahu@cs.nthu.edu.tw](mailto:anitahu@cs.nthu.edu.tw)  
CS, NTHU

# A\* Algorithm

- Cost Function:  $F(v) = g(v) + h(v)$
- Consider both previous and future parameters:
  - $g(v)$  : **calculated** path cost from start to  $v$  (Dijkstra)
  - $h(v)$  : heuristic path **estimation** from  $v$  to target (BFS)
- Special condition of A\*
  - $h(v) = 0$ 
    - degrade to Dijkstra's Algorithm
  - $g(v)=0$  or  $h(v) \gg$  Edge weight
    - degrade to Best-First Search

## [Practice] A\* Algorithm

```
def planning(self, start=(100,200), goal=(375,520), inter=None, img=None):
    if inter is None:
        inter = self.inter
    start = (int(start[0]), int(start[1]))
    goal = (int(goal[0]), int(goal[1]))
    # Initialize
    self.initialize()
    self.queue.append(start)
    self.parent[start] = None
    self.g[start] = 0
    self.h[start] = utils.distance(start, goal)
    while(1):
        # TODO: A Star Algorithm
        break
```

# RRT Algorithm

```
def planning(self, start, goal, extend_len=None, img=None):
    if extend_len is None:
        extend_len = self.extend_len
    self.ntree = {}
    self.ntree[start] = None
    self.cost = {}
    self.cost[start] = 0
    goal_node = None
    for it in range(20000):
        print("\r", it, len(self.ntree), end="")
        samp_node = self._random_node(goal, self.map.shape)
        near_node = self._nearest_node(samp_node)
        new_node, cost = self._steer(near_node, samp_node, extend_len)
        if new_node is not False:
            self.ntree[new_node] = near_node
            self.cost[new_node] = cost + self.cost[near_node]
        else:
            continue
        if utils.distance(near_node, goal) < extend_len:
            goal_node = near_node
            break
```

```
G.initialize( $x_{start}$ ) # Initialize Graph
for i=1 to max_iter do
     $x_{rand} \leftarrow \text{sample}()$  # Sample the points in 2d space.
     $x_{nearest} \leftarrow \text{nearest}(x_{rand}, G)$  # Find the nearest point in graph.
     $x_{new} \leftarrow \text{steer}(x_{rand}, x_{nearest}, \text{step\_size})$  # Collision detection.
    G.add_node( $x_{new}$ )
    G.add_edge( $x_{new}, x_{parent}$ )
    if  $\|x_{new} - x_{goal}\| < \text{threshold}$  then
        return G
```

# Steer

- Check if the line between two nodes is available.

```
def _steer(self, from_node, to_node, extend_len):
    vect = np.array(to_node) - np.array(from_node)
    v_len = np.hypot(vect[0], vect[1])
    v_theta = np.arctan2(vect[1], vect[0])
    if extend_len > v_len:
        extend_len = v_len
    new_node = (from_node[0]+extend_len*np.cos(v_theta), from_node[1]+extend_len*np.sin(v_theta))
    if new_node[1]<0 or new_node[1]>=self.map.shape[0] or new_node[0]<0 or new_node[0]>=self.map.shape[1] or
self._check_collision(from_node, new_node):
        return False, None
    else:
        return new_node, utils.distance(new_node, from_node)
```

# RRT\* Algorithm

```
G.initialize( $x_{start}$ )  # Initialize Graph
for i=1 to max_iter do
     $x_{rand} \leftarrow \text{sample}()$   # Sample the points in 2d space.
     $x_{nearest} \leftarrow \text{nearest}(x_{rand}, G)$   # Find the nearest point in graph.
     $x_{new} \leftarrow \text{steer}(x_{rand}, x_{nearest}, \text{step\_size})$   # Collision detection.
     $X_{near} \leftarrow \text{near\_node}(x_{new}, G)$ 
     $x_{parent} \leftarrow \text{best\_parent}(x_{parent}, X_{near})$   # Re-Parent
    G.add_node( $x_{new}$ )
    G.add_edge( $x_{new}, x_{parent}$ )
    G.rewire( $x_{new}, X_{near}$ )  # Rewire
    if  $\|x_{new} - x_{goal}\| < \text{threshold}$  then
        return G
```

## [Practice] RRT\* Algorithm

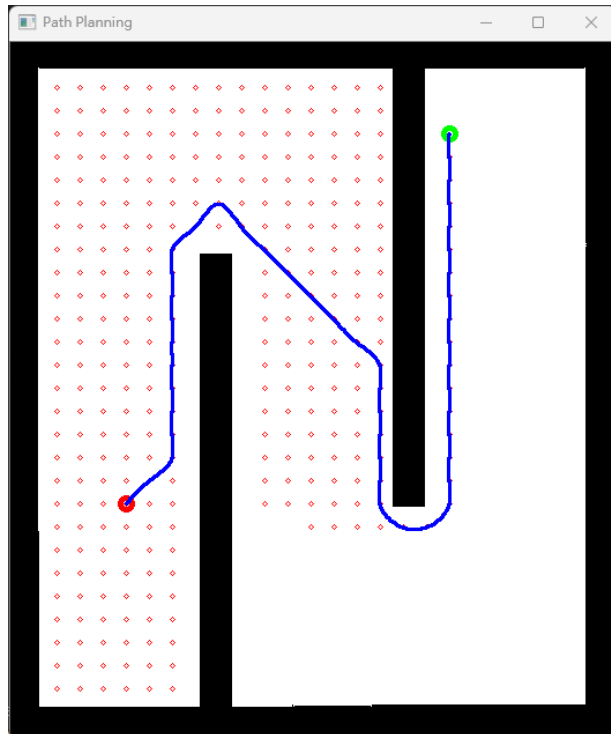
```
def planning(self, start, goal, extend_len=None, img=None):
    if extend_len is None:
        extend_len = self.extend_len
    self.ntree = {}
    self.ntree[start] = None
    self.cost = {}
    self.cost[start] = 0
    goal_node = None
    for it in range(20000):
        #print("\r", it, len(self.ntree), end="")
        samp_node = self._random_node(goal, self.map.shape)
        near_node = self._nearest_node(samp_node)
        new_node, cost = self._steer(near_node, samp_node, extend_len)
        if new_node is not False:
            self.ntree[new_node] = near_node
            self.cost[new_node] = cost + self.cost[near_node]
        else:
            continue
        if utils.distance(near_node, goal) < extend_len:
            goal_node = near_node
            break

    # TODO: Re-Parent & Re-Wire
```

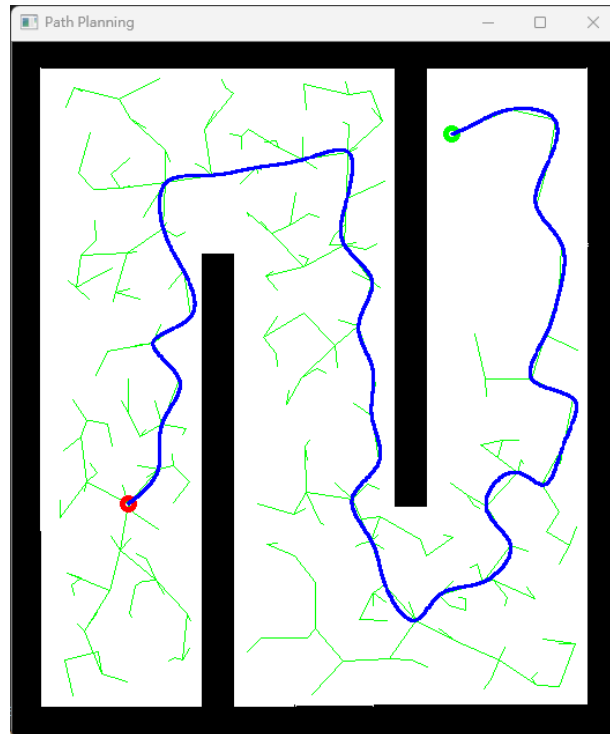
# [Run] Path Planning

```
python 03_path_planning.py -s [a_star/rrt/rrt_star] [--smooth]
```

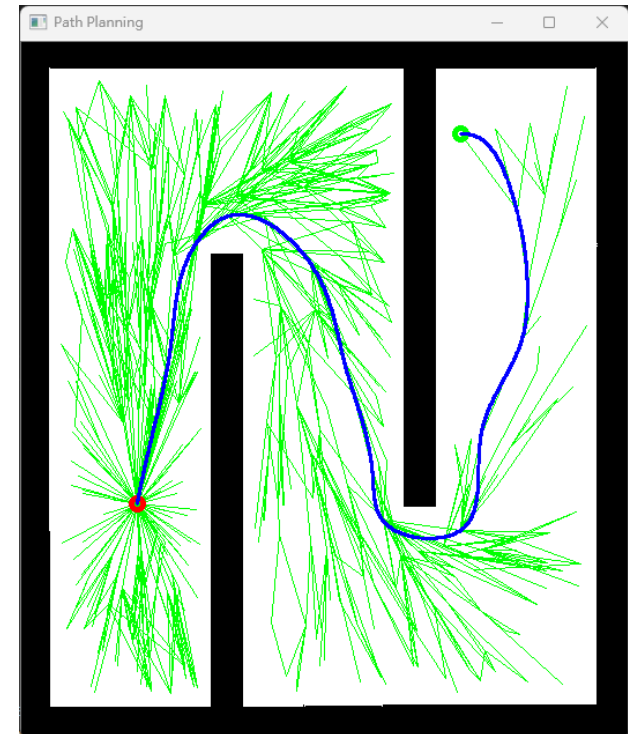
A\*



RRT



RRT\*





# Remind

- Check if you complete the “TODO” in the following files:
  - [./PathPlanning/planner\\_a\\_star.py](#)
  - [./PlathPlanning/planner\\_rrt\\_star.py](#)

Q&A