

# Searching based path planning

- RRT

- Rapidly exploring random tree

- 시작점에서 목표점까지 빠르게 도달하는 경로 생성

- 알고리즘 순서

- ① 랜덤 노드 생성
- ② 랜덤 노드로부터 가장 가까운 노드 선택
- ③ 랜덤노드로 향하는 방향(직선) 으로 step 만큼 늘려 새로운 노드 생성
- ④ 장애물과의 충돌 체크 후 (Euclidean distance 기반) 새로운 노드로 트리에 추가
- ⑤ ① ~ ④ 목적지에 도달할때까지 진행

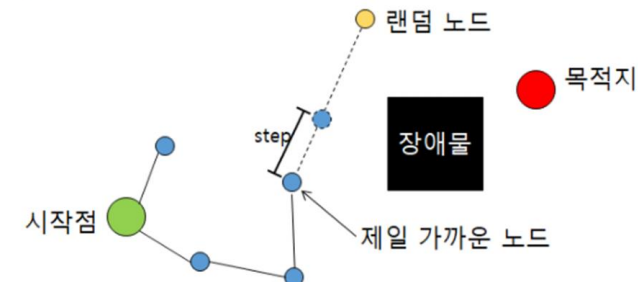
[1] 랜덤노드 설정



[2] 제일 가까운 노드 선택



[3] 새로운 노드 선정



[4] 충돌 확인 후 트리에 추가



- RRT

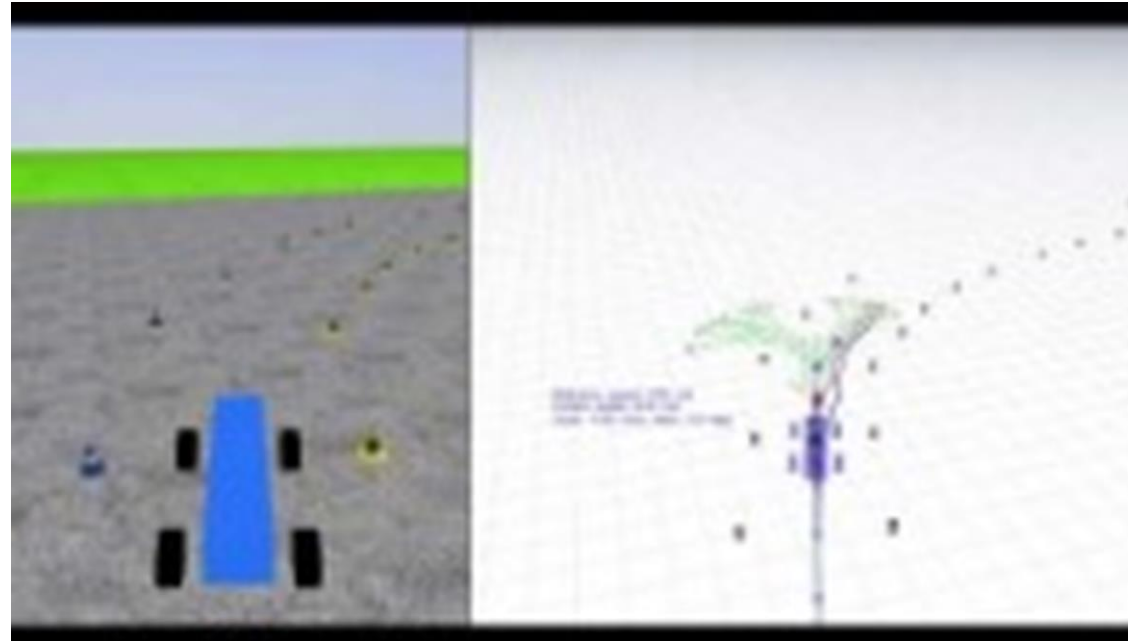
- RRT 알고리즘을 이용한 예시 결과
- Green line: 트리에 추가된 node, edge
- Blue circle: 장애물
- Red line: optimal path



# Searching based path planning

---

- RRT
  - Example
    - Video: <https://youtu.be/kjssdifs0DQ>
    - Code: [https://github.com/MaxMagazin/ma\\_rrt\\_path\\_plan](https://github.com/MaxMagazin/ma_rrt_path_plan)



# Searching based path planning

- RRT\*

- RRT 개선 - Rewiring

- RRT\* 는 RRT 와 다르게 트리내의 노드를 대체하여 cost 를 줄일 수 있는 경우 기존 노드를 대체하여 트리를 구성
    - 따라서 기존 RRT 보다 최적의 경로 생성 가능

- 알고리즘 순서

- ① 랜덤 노드 생성
- ② 랜덤 노드로부터 가장 가까운 노드 선택
- ③ 랜덤노드로 향하는 방향으로 step 만큼 늘려 새로운 노드 생성
- ④ 장애물과의 충돌 체크 후 (충돌이 없을 시)  
새로운 노드로부터 가까운 노드들 선택 (특정 반경 내), (c)  
가까운 노드들 중 lowest cost 를 가진 노드를 새로운 노드와 연결하며 새로운 노드의 부모 노드로 설정, (d)  
가까운 노드들 중 새로운 노드를 부모 노드로 할 때 더 낮은 cost 를 갖는 노드가 있는지 탐색, (e)  
있다면 트리 재구성 (rewiring) 진행, (f)
- ⑤ 목적지에 도달할 때까지 ① ~ ④ 진행

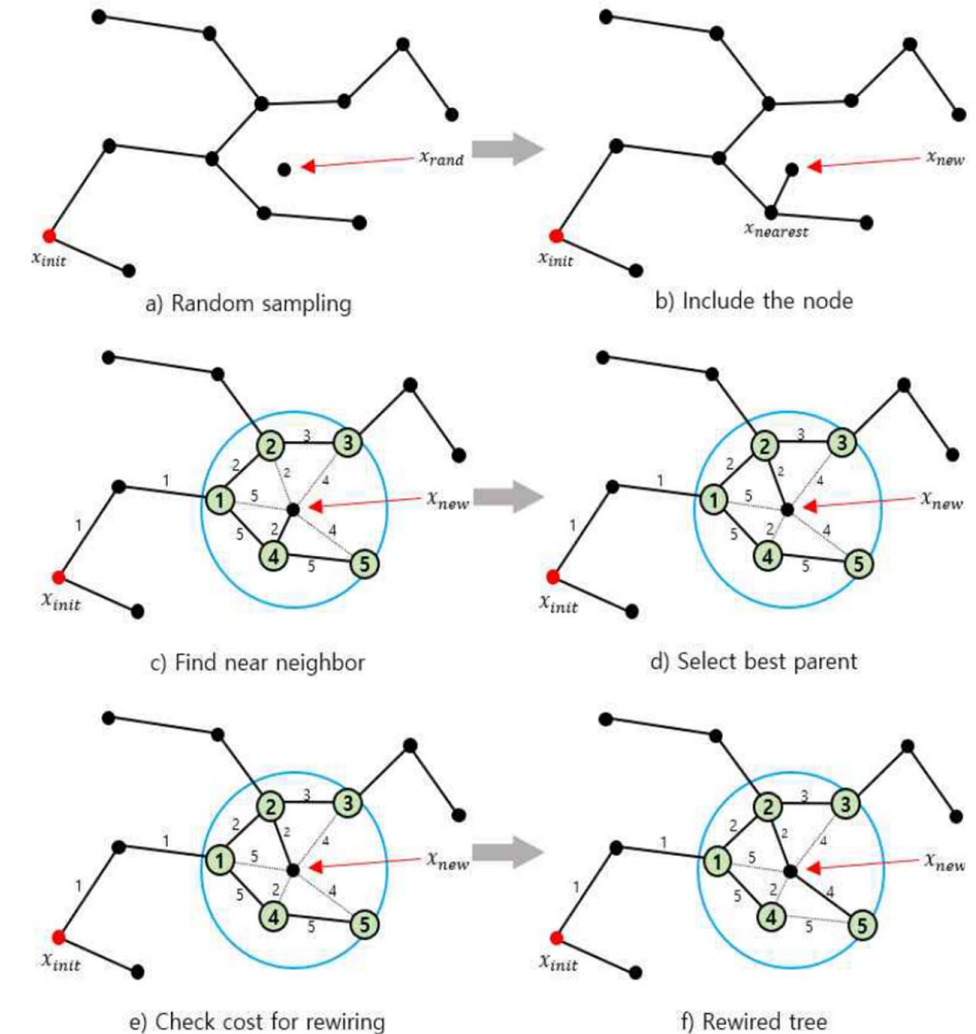


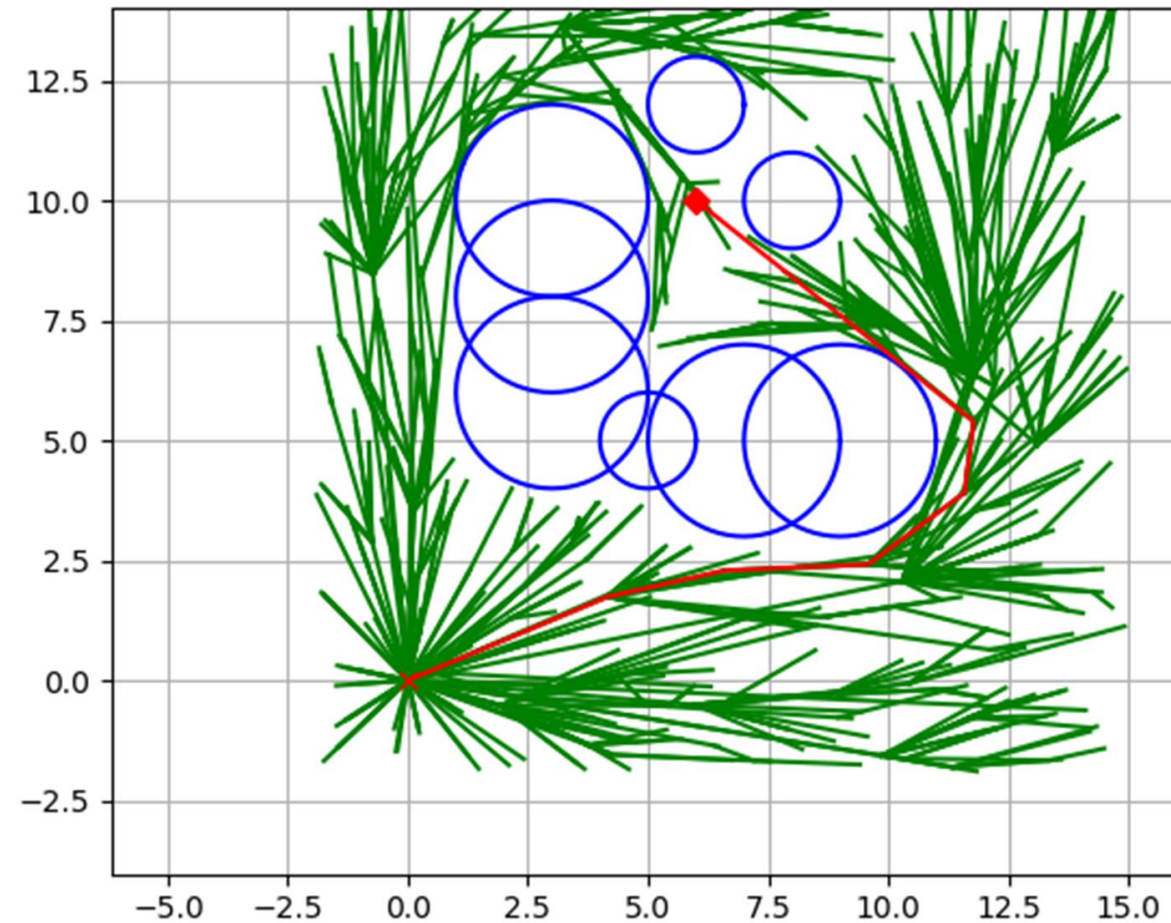
Figure 1. RRT\* 알고리즘 과정

# Searching based path planning

- RRT\*

- RRT\* result

- Green line: 트리에 추가된 node, edge
    - Blue circle: 장애물
    - Red line: optimal path
    - 트리 재구성 파트를 통해 path 가 반듯하게 펴진 형태



# Searching based path planning

- Connection method : Dubis path

- Dubins path

- 두 점 (A, B) 가 주어졌을때, curvature constraint 를 고려하여 두 점을 잇는 shortest curve 를 말함
    - Forward 방향으로만 이동 가능하며 right, straight, left 의 세 조합으로 구성
    - Optimal path 로 총 6 type 존재 (RSR, RSL, LSR, LSL, RLR, LRL)
    - Ref : [https://en.wikipedia.org/wiki/Dubins\\_path](https://en.wikipedia.org/wiki/Dubins_path)

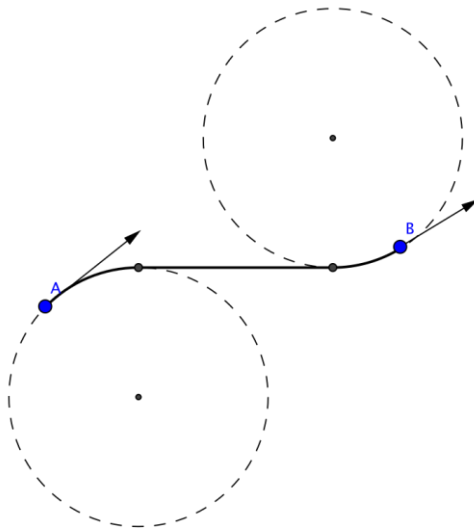


Figure 1. RSL Dubins path 예시

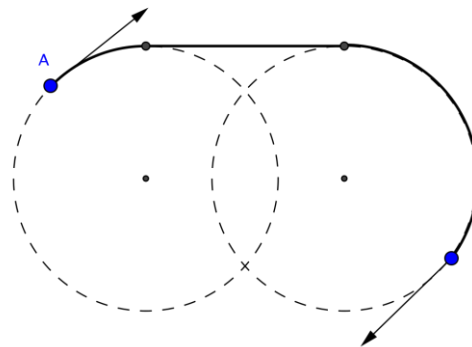


Figure 2. RSR Dubins path 예시

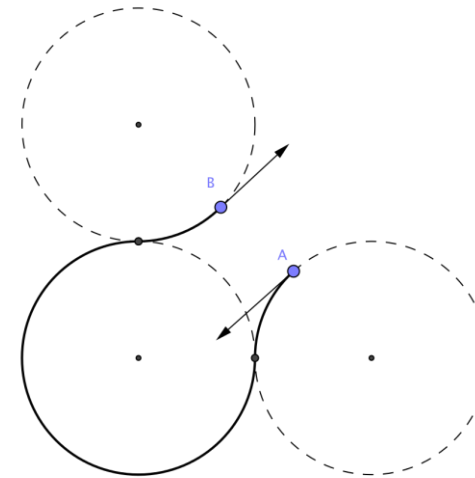
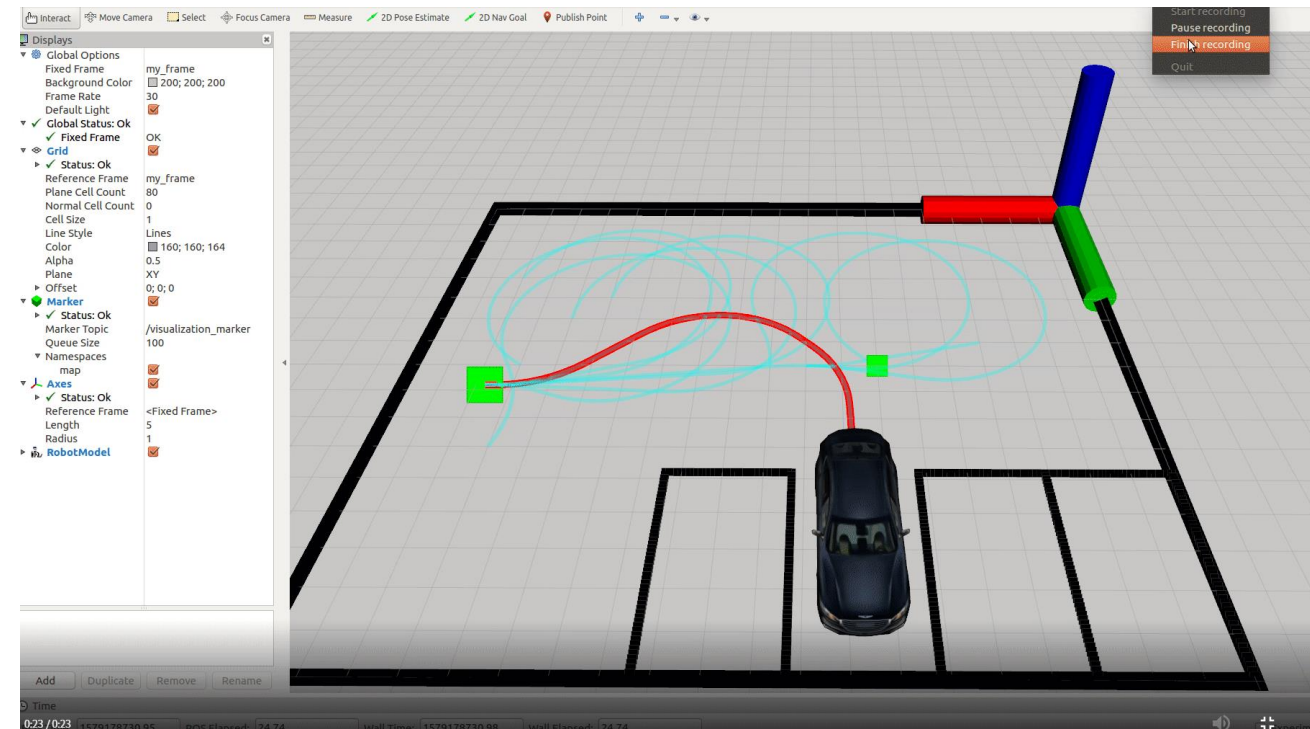


Figure 3. LRL Dubins path  
예시

# Searching based path planning

- RRT\* with Dubins path
  - Example
    - 전진 주차 시나리오
    - RRT\* 알고리즘에서 expansion 시 단순한 직선이 아니라 Dubins path 이용
    - 따라서 차량이 전진주행을 통해 tracking 가능한 경로가 생성됨





# Searching based path planning

- Connection method : CCRS

- Reeds-sheep path

- 원과 직선의 조합으로 path 를 생성
    - Forward 와 reverse 방향 모두 가능
    - Optimal path 로 총 46 type 존재
    - 조합의 사이사이에서 차량이 정지한 후 조향각을 바꾸고 다시 이동해야 함 (curvature가 연속적이지 않기 때문)

- Continuous curvature reeds-sheep path

- Fraichard et al. From Reeds and Shepp's to continuous-curvature paths. IEEE T-R. 논문 참조
    - Continuous한 curvature 을 갖는 Reeds-Sheep path 생성
    - gear 변속이 없는 한 조향을 바꾸기 위해 차량을 멈춰야 할 필요가 없음 (curvature가 연속적이기 때문)

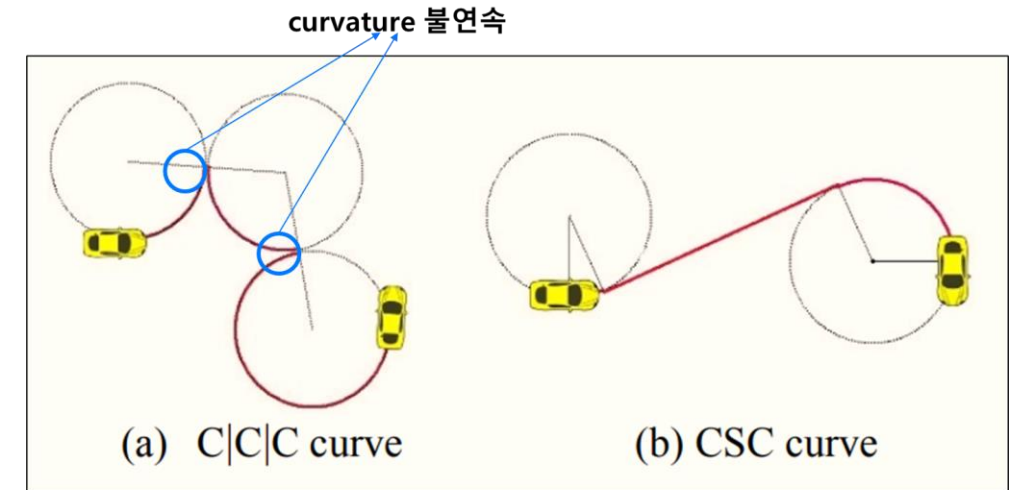


Figure 1. Reeds-Sheep path 예시

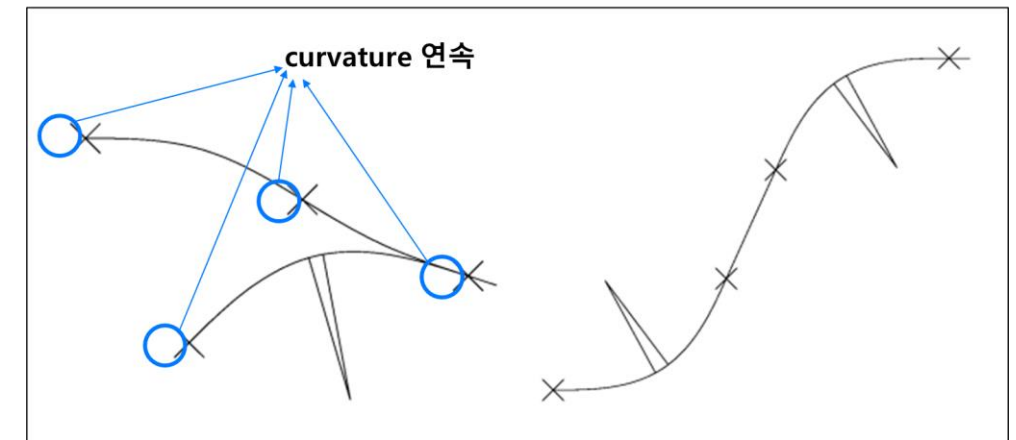
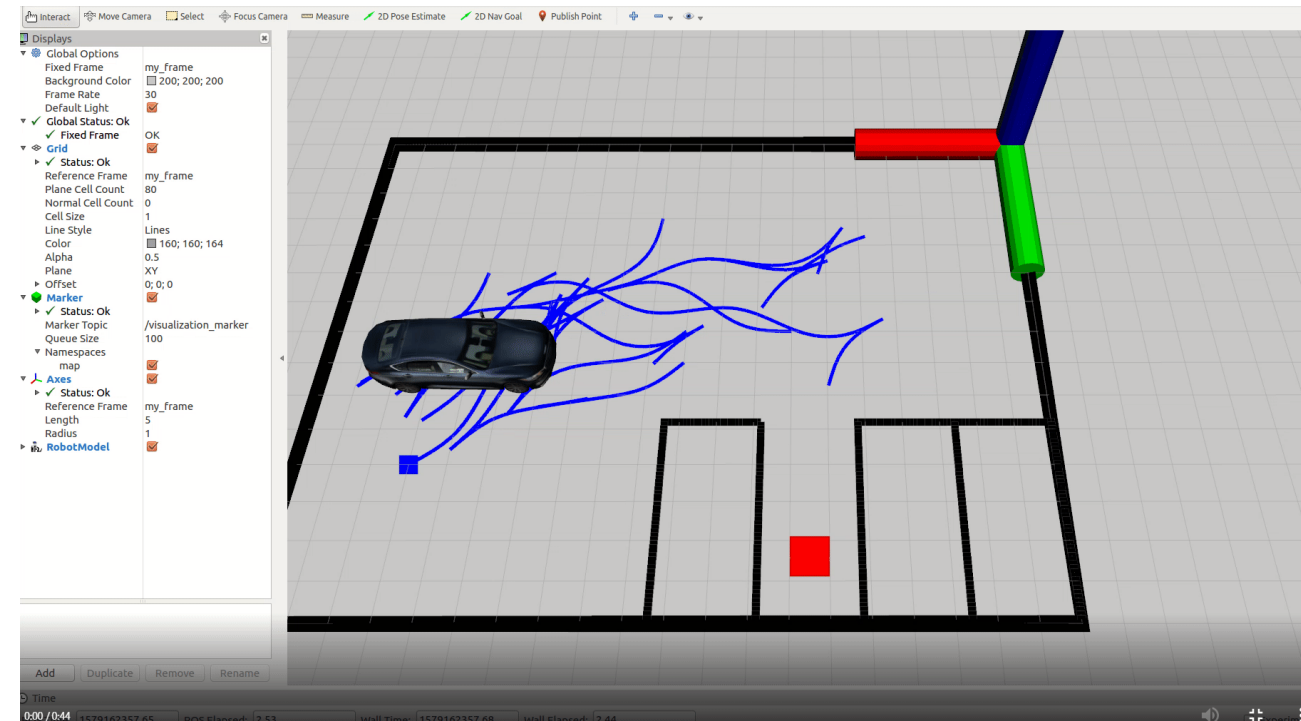


Figure 2. CC path 예시



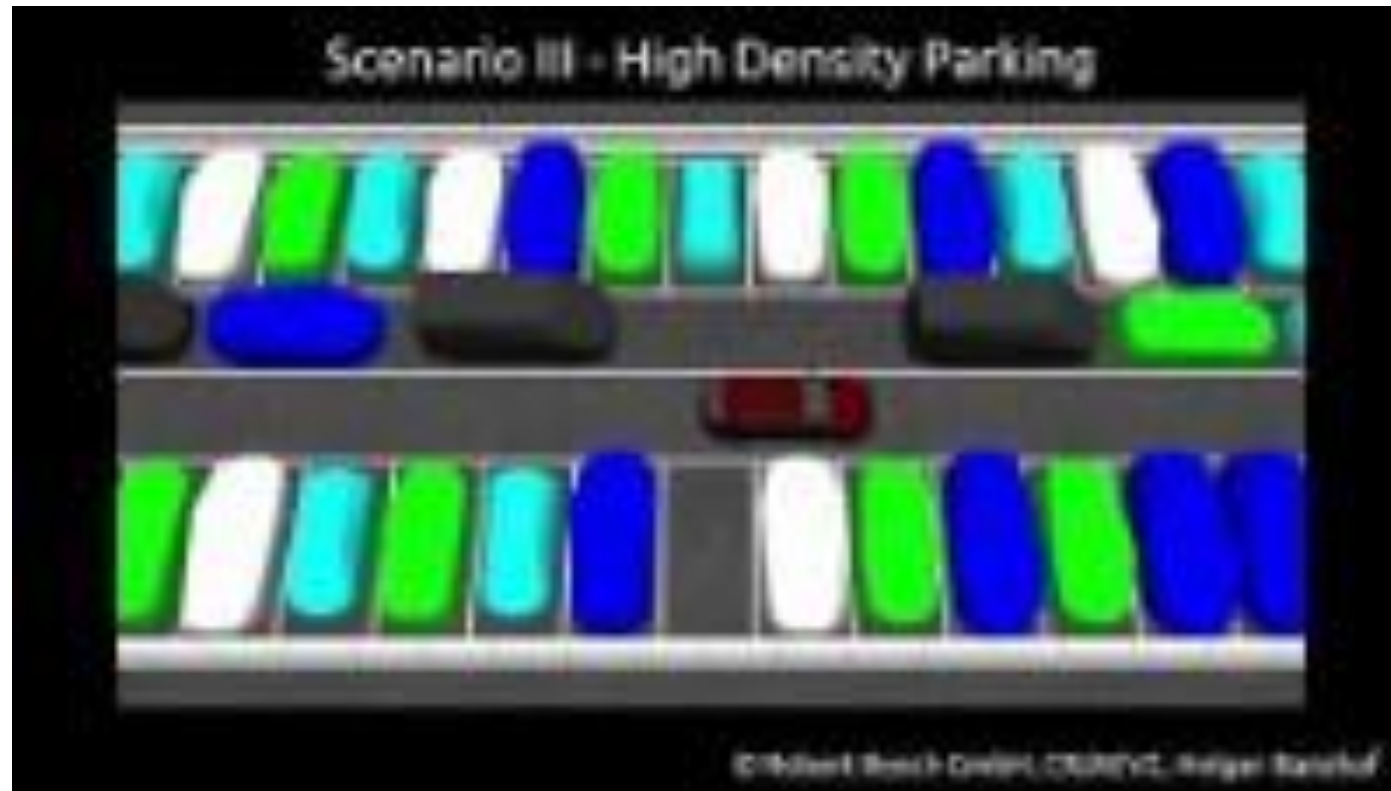
# Searching based path planning

- RRT\* with CCRS
  - Example
    - 후진 주차 시나리오
    - RRT\* 알고리즘에서 expansion 시 단순한 직선이 아니라 CCRS path 이용
    - 따라서 차량이 전진/후진주행을 통해 tracking 가능한 경로가 생성됨



# Searching based path planning

- RRT\* with CCRS
  - Example
    - Video: <https://youtu.be/DLjeuGgDcTM>



# Searching based path planning

- Informed RRT\* Algorithm

- 기존 RRT\*의 단점

- Sampling의 효율성이 떨어짐
    - Large planning problem에서 좋은 quality의 solution 얻기 힘들어짐

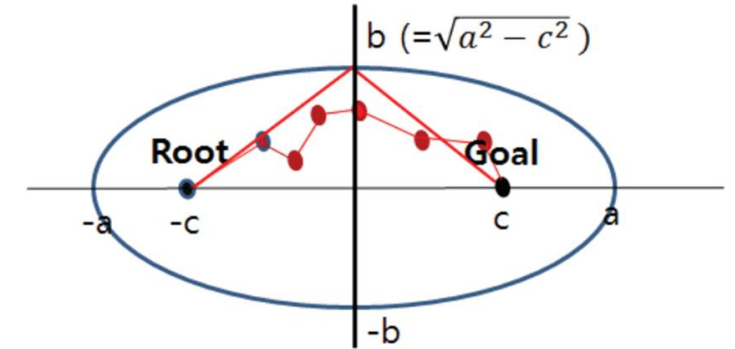
- Informed RRT\*로 단점 개선

- Searching area를 제한해서 planning 효율성을 높이는 방식
    - 타원 위의 한 점에서 두 초점을 잇는 직선 길이의 합이 장축의 길이와 동일하다는 타원의 성질 이용

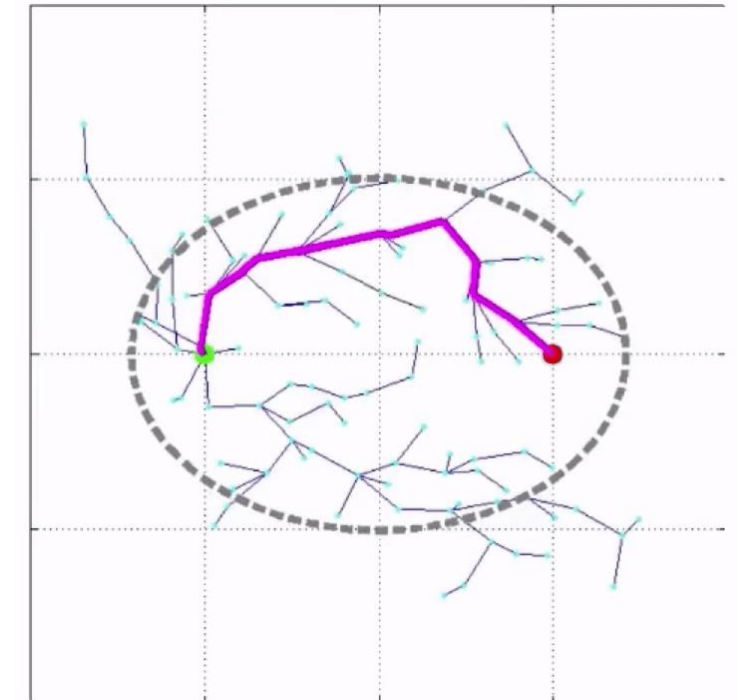
- ① 우선 start에서 goal로 도달하는 initial solution을 구함

- ② 얻은 solution의 경로 길이를 장축길이를 하며 start과 goal을 초점으로 하는 타원 생성  
→ 타원의 특성으로 인해 타원 밖 영역에는 worse candidate만 존재

- ③ 타원 내부에서 sampling을 진행하며 solution을 발전시켜나감



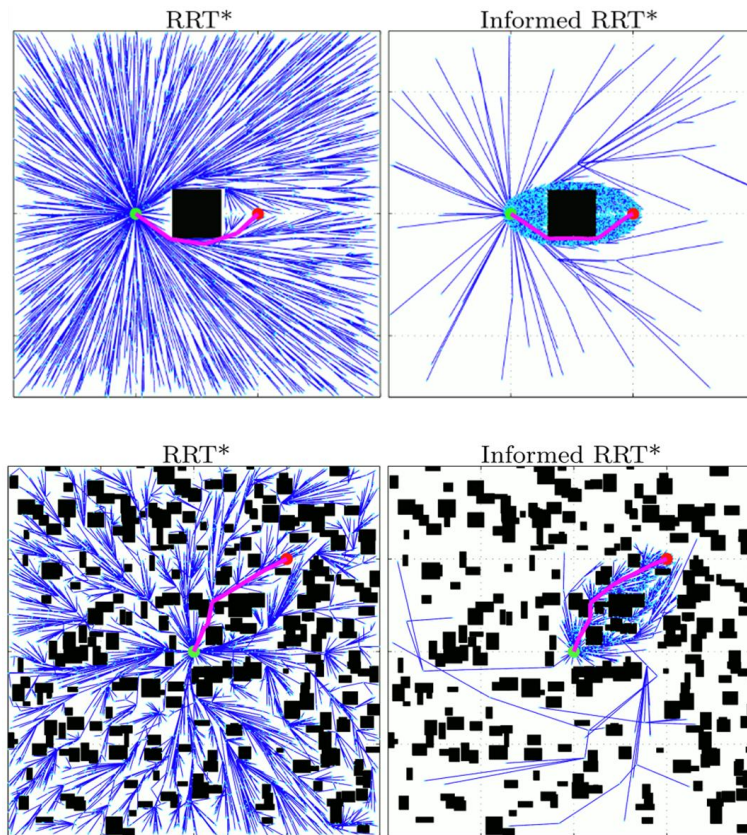
(장축길이 = 경로 길이 =  $2a$ )



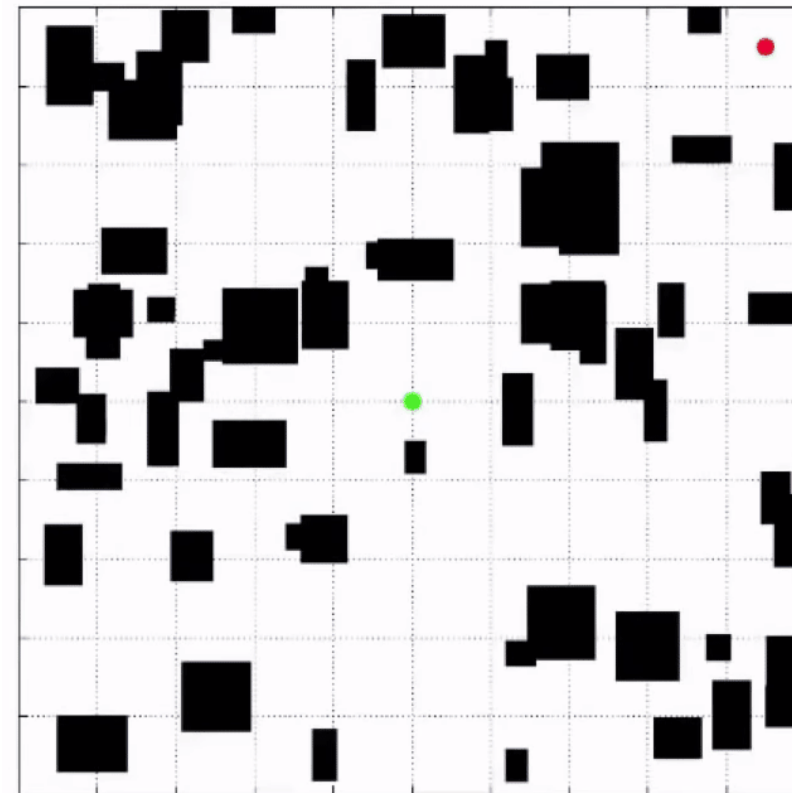
Informed RRT\*

# Searching based path planning

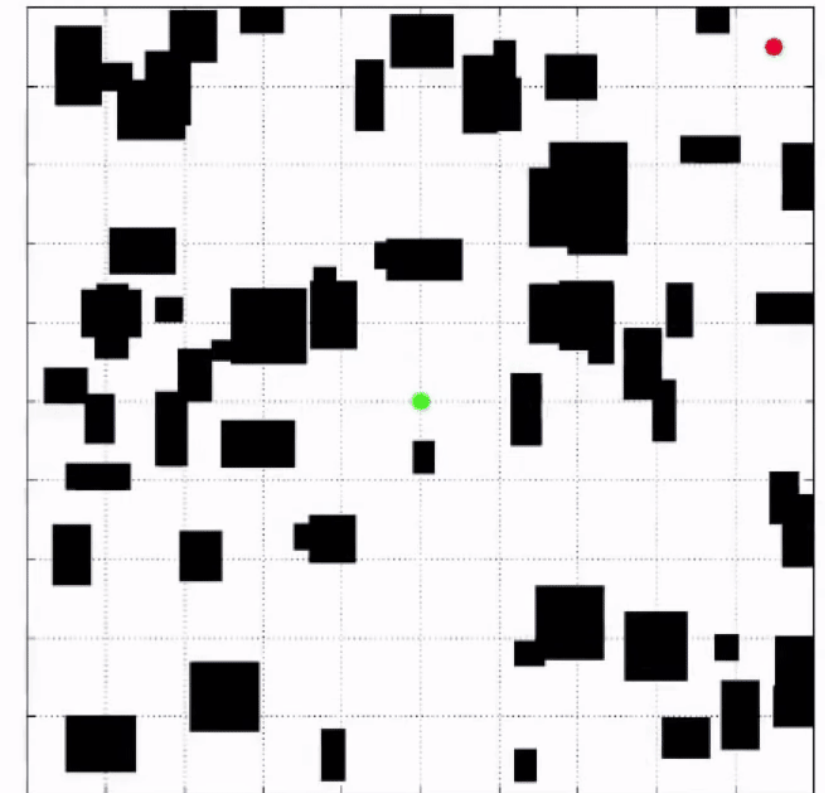
- Informed RRT\* Algorithm
  - Example



Informed RRT\*



RRT\*



Informed RRT\*

# Searching based path planning

- NetworkX package

Powerful tool for some applications using graph structure

- It provides classes for graphs which allow multiple edges between any pair of nodes

Helpful guide for NetworkX

- <https://networkx.org/documentation/stable/tutorial.html>

Create graph

```
>>> import networkx as nx
>>> G = nx.Graph()
```

>>>

Add nodes

```
>>> G.add_nodes_from([
...     (4, {"color": "red"}),
...     (5, {"color": "green"}),
... ])
```

>>>

Add edge

```
>>> G.add_edge(1, 2)
>>> e = (2, 3)
>>> G.add_edge(*e) # unpack edge tuple*
```

>>>

## Contact

[Mailing list](#)  
[Issue tracker](#)  
[Source](#)

## Releases

### Stable (notes)

2.5.1 — April 2021  
[download](#) | [doc](#) | [pdf](#)

### Latest (notes)

2.6 development  
[github](#) | [doc](#) | [pdf](#)

## Archive

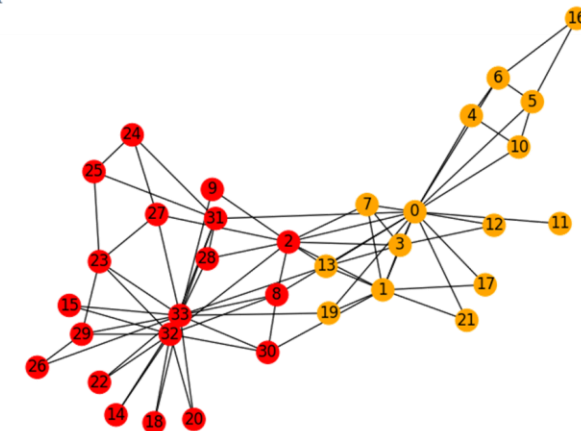


NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.



## Software for complex networks

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Open source [3-clause BSD license](#)
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy to teach, and multi-platform





Thank You

