A* 알고리즘

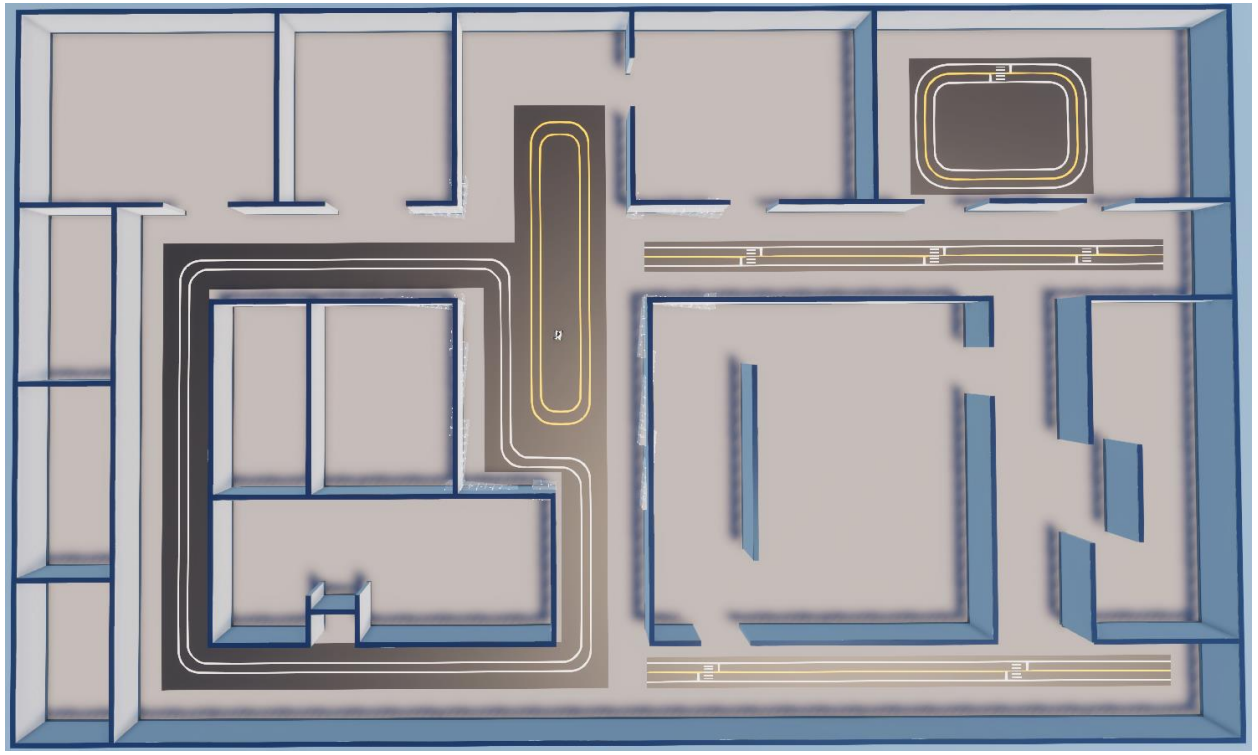
2021 / 04 / 27

Find Shortest Path

카카오 지도 – 광운대 중앙도서관

- 불특정 다수의 경로가 존재하는 상황에서 가장 빠른 길을 찾는 경우가 많음 (Ex. 네비게이션)
- 여러 조건을 고려하며 최단 경로를 찾는 문제는 가장 대표적인 비 결정론적 문제 (NP problem)

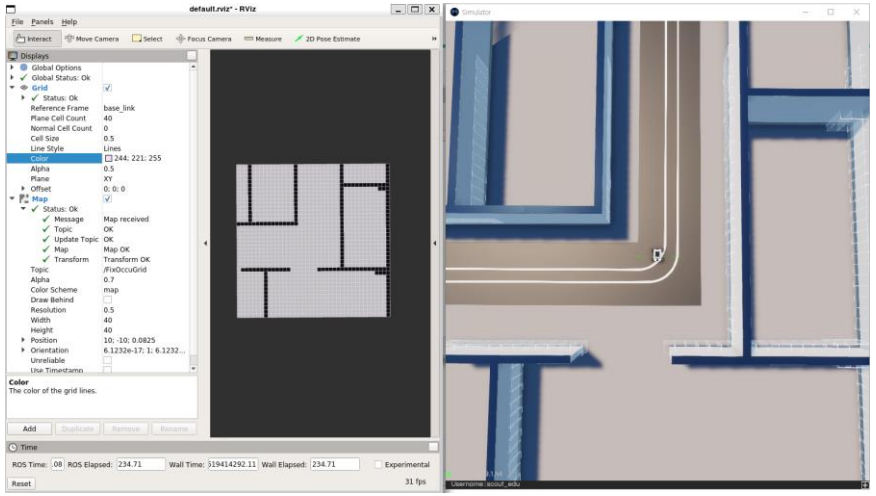
Find Shortest Path

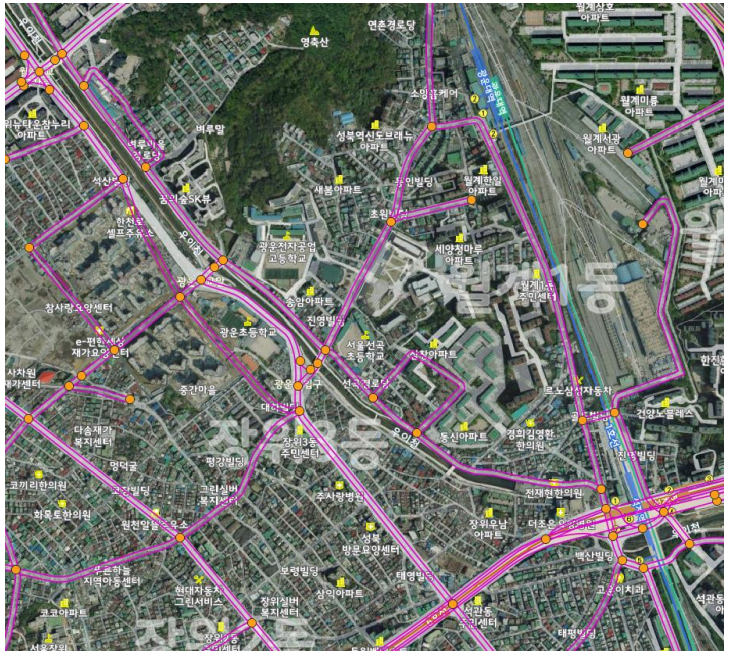


- 정밀 지도와 같은 지도가 주어지지 않은 상황
  - 특정 좌표로 이동하길 원함
    - 가장 빠른 길을 원함

Graph Map

- 주행 경로에 대한 제약 등이 고려된 지도를 생성
  - 단위 거리당 제약조건을 비용으로 지도 생성
    -> Occupancy Grid Map
    - Free Space등에서 사용되며 SLAM을 이용하여 지도를 생성할 수 있음
    - 각 칸에 비용을 넣어 장애물 뿐만 아니라 확률 지도 생성이 가능
      - 시뮬레이터의 비용 범위는 0 부터 100까지
      - 길은 0, 벽은 100, 장애물은 0보다 큰 수
    - 4방향에 관한 고려만 하면 되기 때문에 조건 계산이 간소해짐
    - 단위 거리가 작을 수록 지도가 정밀 해지며 로봇의 움직임이 정밀해짐
    - 소국적 지도로써 local planning 용도로 많이 사용됨
  - 특정 기준에 의한 경로를 이용한 지도 생성
    -> Graph Map
    - Occupancy Grid Map으로 그리기 어려운 지도를 Graph Map 형태로 사용
    - 각각의 특정 조건을 충족하는 지점을 Vertex로 설정한 뒤 Vertex들을 잇는 Edge들을 경로로써 사용
    - 특정 목적에 맞게 다른 형태의 지도를 Graph Map 형태로 변환하여 계산할 수 있음

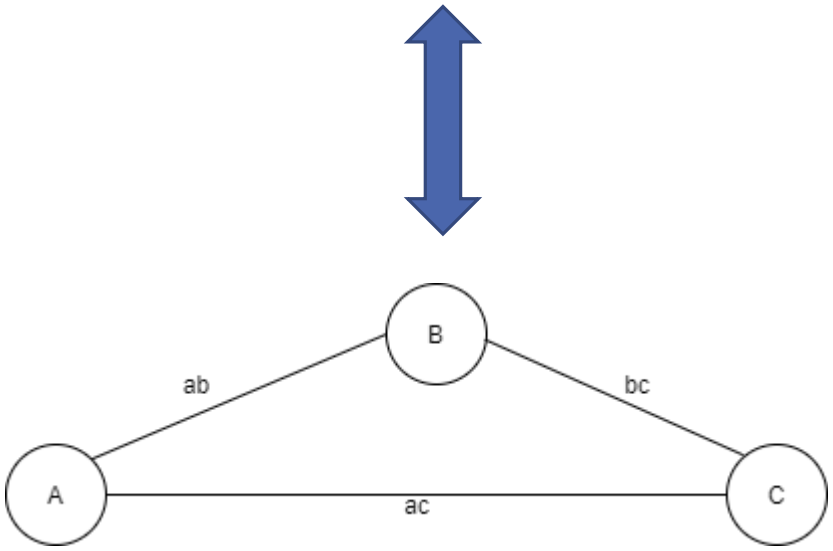




- **Graph Theory**
  - 꼭지점, 정점, 간선을 이용하여 객체간 관계를 모델링하는 이론
  - 지도 / 경로 등에서 많이 사용되는 데이터 구조 형태
  - 꼭지점(Vertex), 정점(Node)과 간선(Edge)을 이용하여 데이터를 표현할 수 있음
  - 각 점에 대한 비용(Cost)과 점과 점 사이의 간선 비용을 이용하여 비용이 가장 작은/큰 데이터 관계를 구할 수 있음

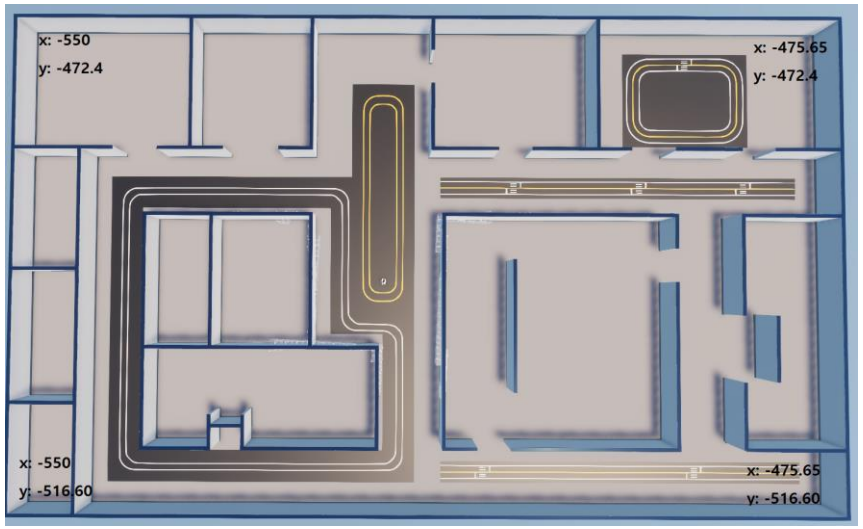
- $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ => V는 Vertex, E는 Edge
  $\mathbf{V} = \{A, B, C\}$, $\mathbf{E} = \{ab, bc, ac\}$
  - $A \rightarrow B : ab\ (cost\ 1)$
  - $B \rightarrow C : bc\ (cost\ 1)$
  - $A \rightarrow C : ac\ (cost\ 2)$

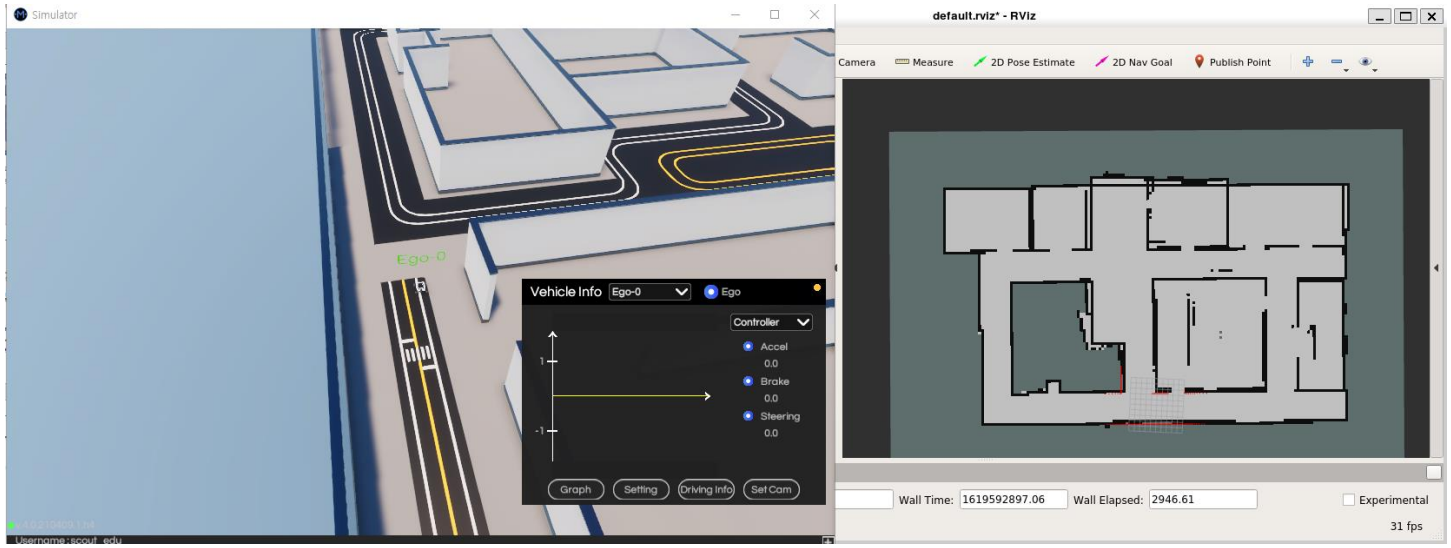
	A	B	C
A	0	1	2
B	1	0	1
C	2	1	0



Mapping

- ROS 패키지를 이용한 맵핑
  - 로봇의 Odometry, lidar를 이용하여 맵핑함
  - Gmapping
    - OpenSlam에서 공개한 오픈소스 맵핑 툴
    - 2D lidar의 센서 데이터와 lidar 와 로봇간 TF를 이용하여 데이터를 누적
    - sudo apt install ros-melodic-gmapping
    - Gmapping tool은 런타임 동안에 지속적으로 맵을 누적하여 송출함
    - 전체적인 맵핑이 완료되면 map-server를 이용하여 저장함
  - Map server
    - ROS 토픽으로 송출되어 나오는 지도 정보를 저장하거나 지도 파일을 로드하여 송출함
    - sudo apt install ros-melodic-map-server





맵핑 방법

- 우측의 런처를 생성하여 실행
- 맵핑이 다 완료되면 다음의 명령어를 실행
  - rosrun map_server map_saver –f 맵파일이름
- Gmapping을 종료한 뒤 저장한 맵 파일을 실행
  - rosrun map_server map_server 맵파일이름.yaml

```
<launch>
  <node pkg="gmapping" type="slam_gmapping" name="scout_gmapping" output="screen">
    <remap from="/scan" to="/lidar2D" />
    <param name="map_frame"                value="map" />
    <param name="base_frame"               value="base_link" />
    <param name="scan"                     value="lidar2D" />
    <param name="xmin"                     value="-560" />
    <param name="xmax"                     value="-466" />
    <param name="ymin"                     value="-526" />
    <param name="ymax"                     value="-462" />
    <param name="map_update_interval"      value="5" />
    <param name="delta"                    value="0.5" />
    <param name="lstep"                    value="0.1" />
    <param name="llsamplerange"            value="0.1" />
    <param name="llsamplestep"             value="0.1" />
    <param name="particles"                value="50" />
    <param name="resampleThreshold"        value="0.4" />

    <param name="srr"                      value="0.05" />
    <param name="srt"                      value="0.05" />
    <param name="str"                      value="0.05" />
    <param name="stt"                      value="0.2" />
  </node>
</launch>

```

Find Shortest Path

I. Dijkstra Algorithm

- 가장 유명한 최단 경로 계산 알고리즘
- 모든 경로에 대한 최소 비용 탐색
- 많은 파생 알고리즘이 존재함

II. A* (A-Star) Algorithm

- Dijkstra 알고리즘을 개량한 알고리즘
- 계산 속도와 규모, 편향성을 고려함
- 베이스 이론으로 다른 파생 알고리즘이 많이 생겨남
  (Ex. Lifelong Planning A*, D*, Etc.)

광운대 부근 표준 링크 노드 – 길 탐색

I. Dijkstra Algorithm

- 특정 지점에서 다른 지점으로 가기까지의 최단 거리를 전부 계산
- 도착점을 찾을 때 까지 갈 수 있는 모든 방향을 대상으로 함
- 장점
  - ✓ 정확한 최소 경로를 탐색할 수 있음

- 단점
  - ✓ 모든 경로를 전부 탐색하기 때문에 연산 횟수가 많아짐 -> 느려짐
  - ✓ 지도가 커질 수록 부하가 많이 걸림

1. 그래프 구조의 모든 경로 대상을 메모리에 올림
2. 현재 로봇이 위치한 vertex에서 갈 수 있는 vertex들에 대하여 비용 계산을 시작
3. 모든 진행 방향에 대하여 비용을 계산하여 그중 가장 작은 비용의 방향으로 진행
4. 모든 비용을 계산하면 탐색 종료

```
function Dijkstra(Graph, source):
    create vertex set Q
    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u:
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Dijkstra algorithm pseudo code - Wikipedia

A* Algorithm

I. A* (A-Star) Algorithm

- 특정 지점에서 다른 지점으로 가기까지의 최단 거리를 특정 기준을 고려하여 계산 (Heuristic)
- 도착점을 찾을 때 까지 갈 수 있는 방향에 대하여 가장 빨리 탐색된 경로를 정답으로 처리
- 장점
  - ✓ Heuristic한 기준에 따라서 연산 횟수가 줄어들 수 있음

- 단점
  - ✓ Heuristic한 기준이 현재 지도에 적합하지 않으면 Dijkstra와 차이가 없음
  - ✓ 정밀한 최소 비용 거리를 찾지 못할 수 있음

1. 추가적인 기준 함수 (Heuristic_f)를 설계
2. 그래프 구조의 모든 경로 대상을 메모리에 올림
3. 현재 로봇이 위치한 위치에서 갈 수 있는 정점들에 대하여 비용을 계산
4. 비용 계산시에 Heuristic_f에 의한 값을 추가
5. 가장 낮은 비용을 우선하며 탐색을 진행
6. 목적지에 도달하면 탐색 종료

```
function reconstruct_path(cameFrom, current)
    total_path = {current}
    while focus in cameFrom.Keys:
        focus_edge = cameFrom[focus]
        total_path.push_front(0, focus_edge)
    return total_path

function A_Star(start, goal, heuristic_f)
    openSet = {start}
    cameFrom = an empty map
    gScore = map with default value of Infinity
    gScore[start] = 0
    fScore = map with default value of Infinity
    fScore[start] = heuristic_f(start)
    while openSet is not empty
        current = the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)
        openSet.Remove(current)
        for each neighbor of current
            tentative_gScore = gScore[current] + length(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                cameFrom[neighbor] = current
                gScore[neighbor] = tentative_gScore
                fScore[neighbor] = gScore[neighbor] + heuristic_f(neighbo
r)
                if neighbor not in openSet
                    openSet.add(neighbor)

    return failure
```