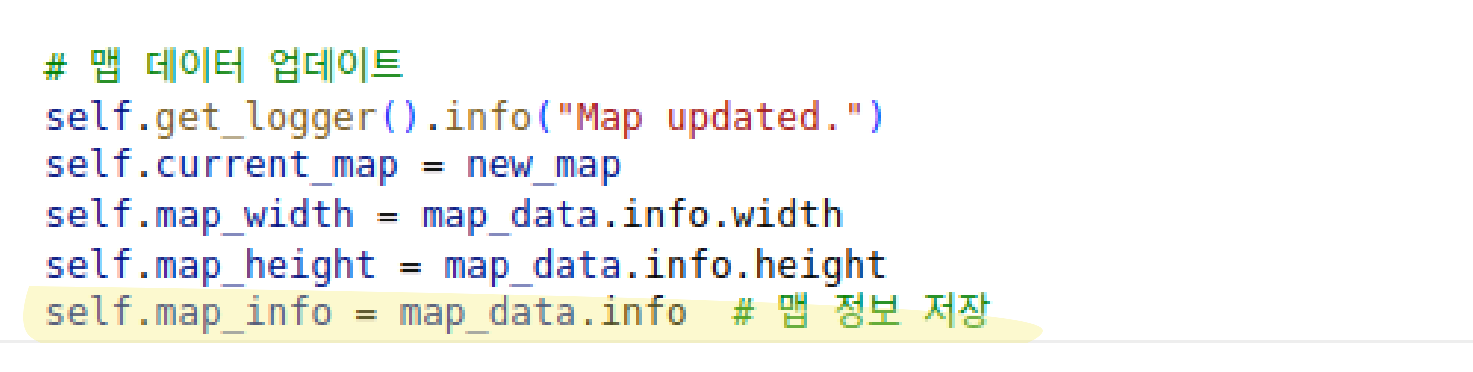
SLAM 모델 기반 다중이용시설 로봇 주행 환경 장애물 인식 모델 개발

1조

MAPPING LOGIC

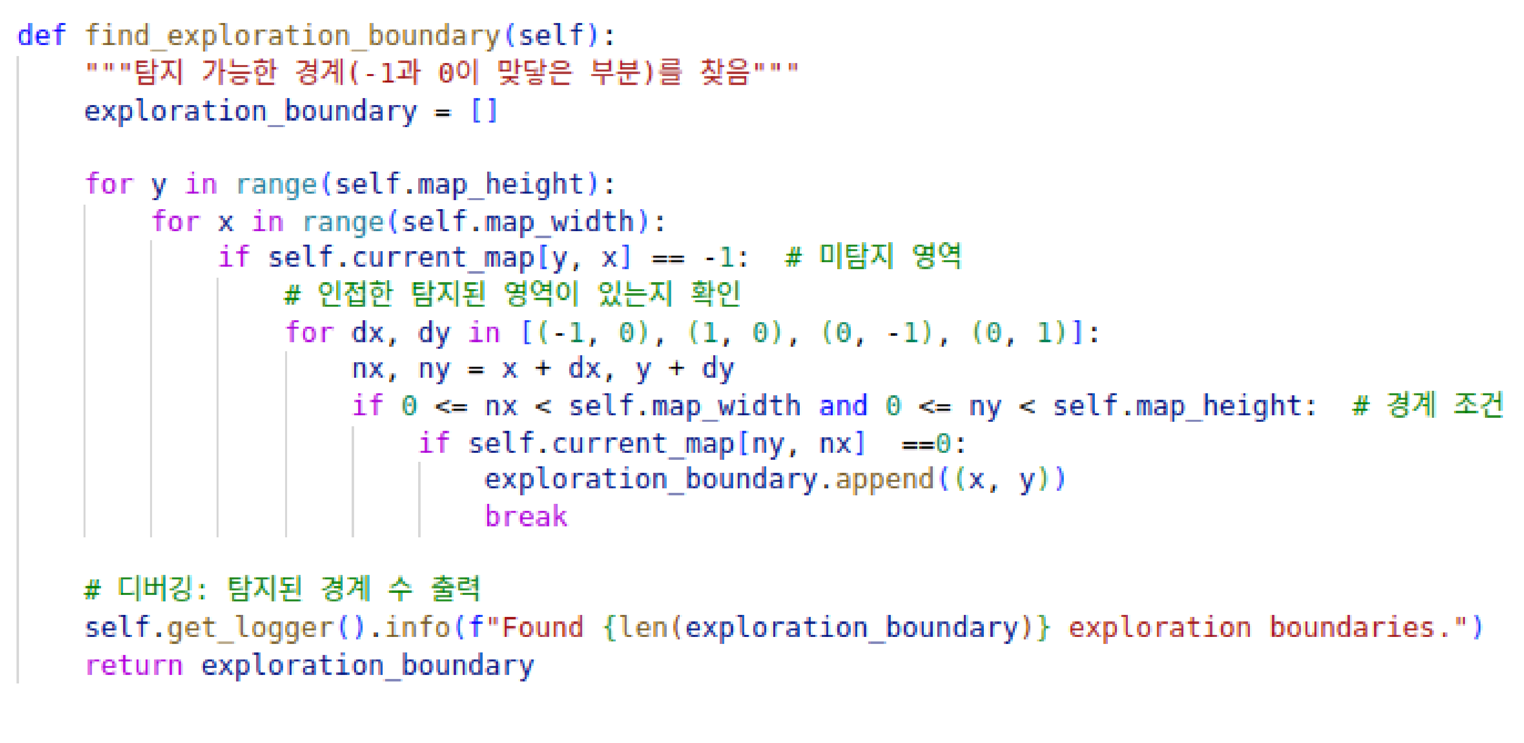


```
# 맵 데이터 업데이트
self.get_logger().info("Map updated.")
self.current_map = new_map
self.map_width = map_data.info.width
self.map_height = map_data.info.height
self.map_info = map_data.info  # 맵 정보 저장
```

-1: 미탐지 구역

0: 탐색완료, 주행가능 공간

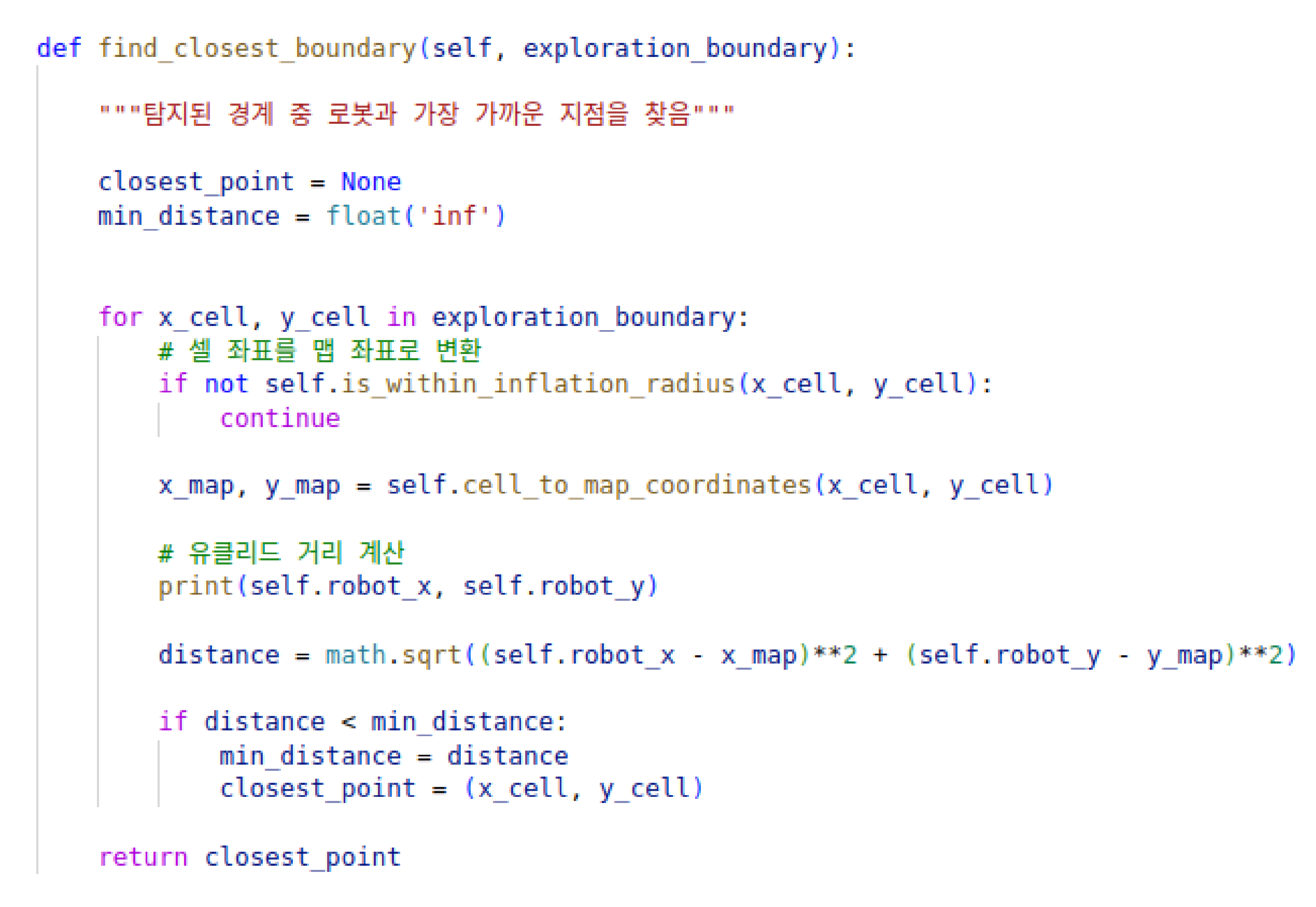
100: 장애물로 인한 주행 불가



```
def find_exploration_boundary(self):
    """탐지 가능한 경계(-1과 0이 맞닿은 부분)를 찾음"""
    exploration_boundary = []

    for y in range(self.map_height):
        for x in range(self.map_width):
            if self.current_map[y, x] == -1:  # 미탐지 영역
                # 인접한 탐지된 영역이 있는지 확인
                for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < self.map_width and 0 <= ny < self.map_height:  # 경계 조건
                        if self.current_map[ny, nx]  ==0:
                            exploration_boundary.append((x, y))
                            break

    # 디버깅: 탐지된 경계 수 출력
    self.get_logger().info(f"Found {len(exploration_boundary)} exploration boundaries.")
    return exploration_boundary
```

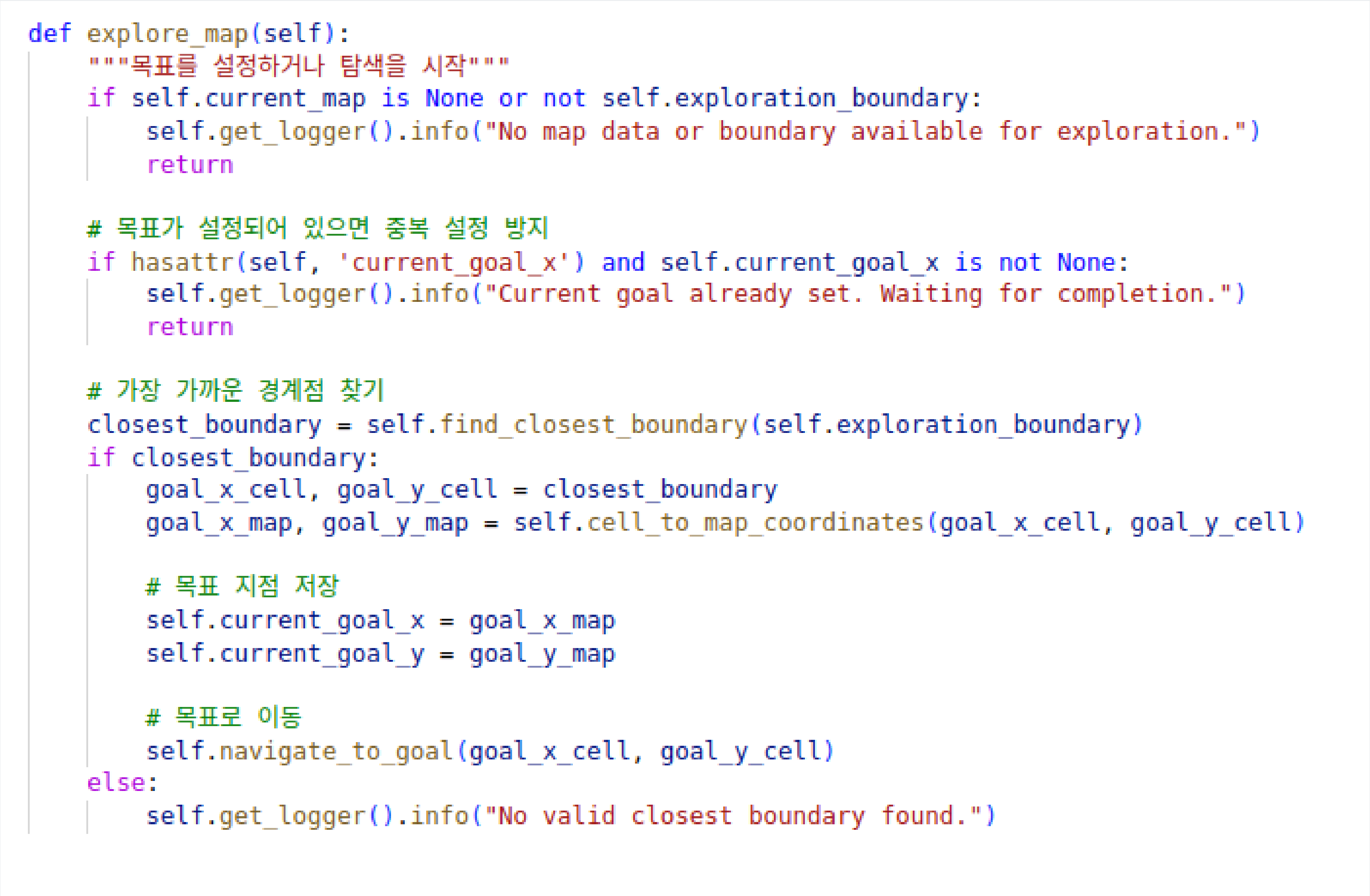
탐지구역과 미탐지 경계 확인해

가장 가까운 미탐지 구역 탐색



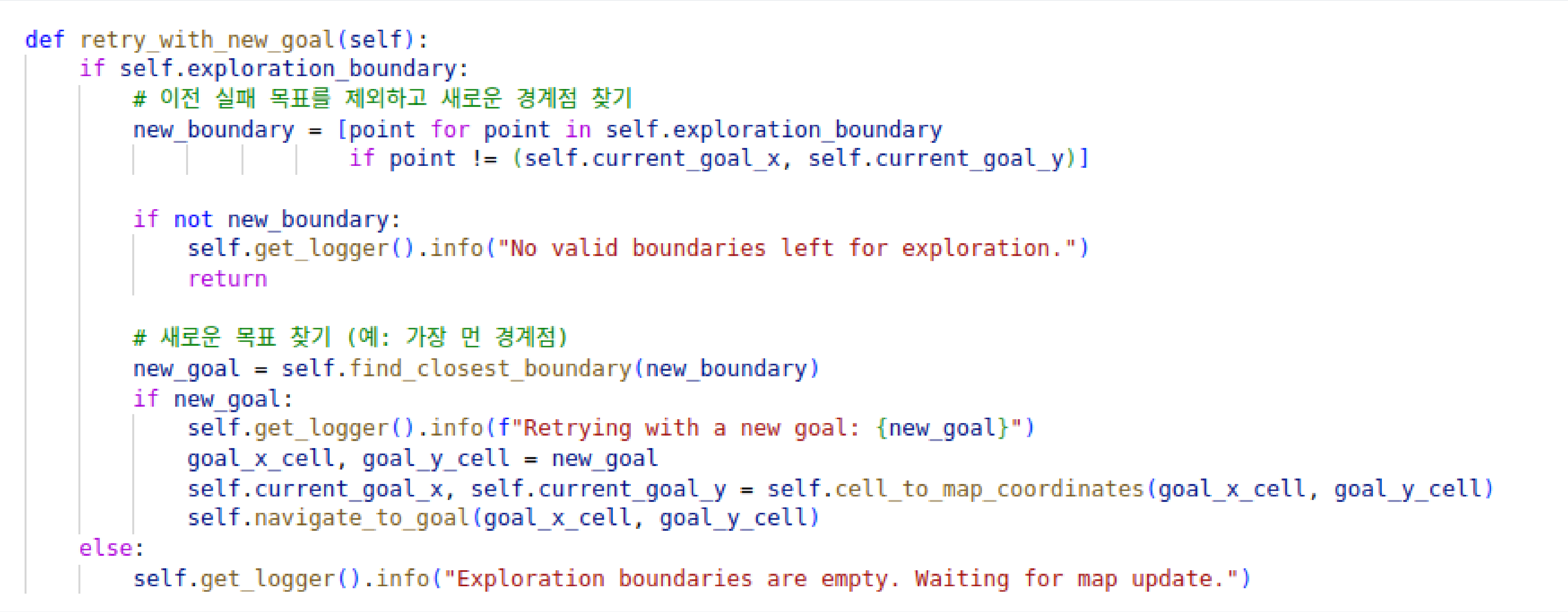
```
def find_closest_boundary(self, exploration_boundary):

    """탐지된 경계 중 로봇과 가장 가까운 지점을 찾음"""

    closest_point = None
    min_distance = float('inf')


    for x_cell, y_cell in exploration_boundary:
        # 셀 좌표를 맵 좌표로 변환
        if not self.is_within_inflation_radius(x_cell, y_cell):
            continue

        x_map, y_map = self.cell_to_map_coordinates(x_cell, y_cell)

        # 유클리드 거리 계산
        print(self.robot_x, self.robot_y)

        distance = math.sqrt((self.robot_x - x_map)**2 + (self.robot_y - y_map)**2)

        if distance < min_distance:
            min_distance = distance
            closest_point = (x_cell, y_cell)

    return closest_point
```

MAPPING LOGIC



```python
def explore_map(self):
    """목표를 설정하거나 탐색을 시작"""
    if self.current_map is None or not self.exploration_boundary:
        self.get_logger().info("No map data or boundary available for exploration.")
        return

    # 목표가 설정되어 있으면 중복 설정 방지
    if hasattr(self, 'current_goal_x') and self.current_goal_x is not None:
        self.get_logger().info("Current goal already set. Waiting for completion.")
        return

    # 가장 가까운 경계점 찾기
    closest_boundary = self.find_closest_boundary(self.exploration_boundary)
    if closest_boundary:
        goal_x_cell, goal_y_cell = closest_boundary
        goal_x_map, goal_y_map = self.cell_to_map_coordinates(goal_x_cell, goal_y_cell)

        # 목표 지점 저장
        self.current_goal_x = goal_x_map
        self.current_goal_y = goal_y_map

        # 목표로 이동
        self.navigate_to_goal(goal_x_cell, goal_y_cell)
    else:
        self.get_logger().info("No valid closest boundary found.")
```

가장 가까운 미탐지 구역을
GOAL로 지정하며 MAP 생성

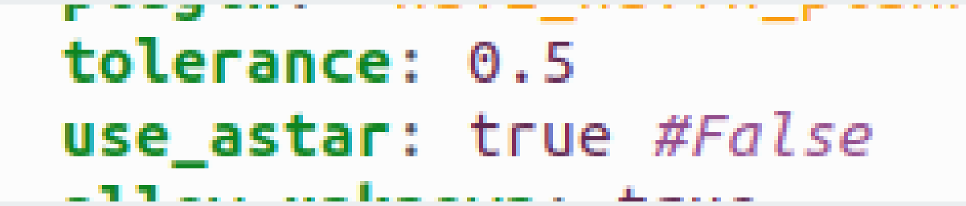


```python
def retry_with_new_goal(self):
    if self.exploration_boundary:
        # 이전 실패 목표를 제외하고 새로운 경계점 찾기
        new_boundary = [point for point in self.exploration_boundary
                        if point != (self.current_goal_x, self.current_goal_y)]

        if not new_boundary:
            self.get_logger().info("No valid boundaries left for exploration.")
            return

        # 새로운 목표 찾기 (예: 가장 먼 경계점)
        new_goal = self.find_closest_boundary(new_boundary)
        if new_goal:
            self.get_logger().info(f"Retrying with a new goal: {new_goal}")
            goal_x_cell, goal_y_cell = new_goal
            self.current_goal_x, self.current_goal_y = self.cell_to_map_coordinates(goal_x_cell, goal_y_cell)
            self.navigate_to_goal(goal_x_cell, goal_y_cell)
    else:
        self.get_logger().info("Exploration boundaries are empty. Waiting for map update.")
```

GOAL에 도달하지 못하면
재탐색

PARAMETER TUNNING

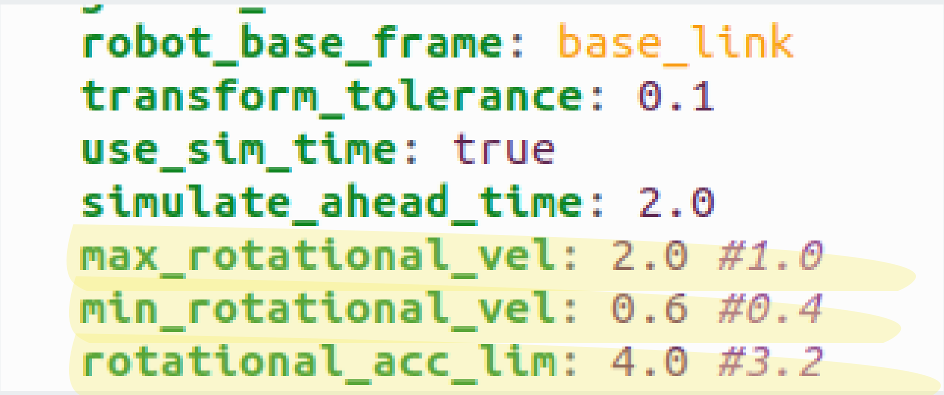
planner_server



```
tolerance: 0.5
use_astar: true #False
```

A*는 경로 계획 알고리즘

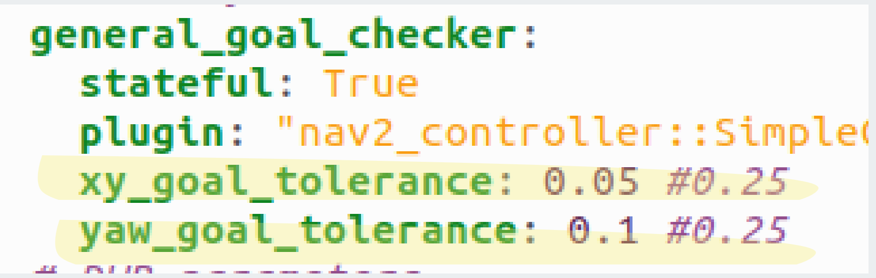
behavior_server



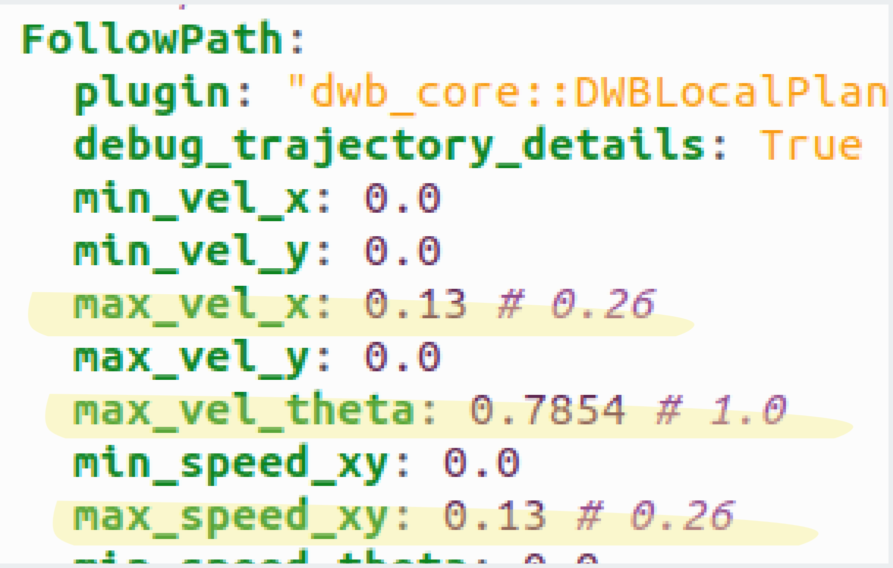
```
robot_base_frame: base_link
transform_tolerance: 0.1
use_sim_time: true
simulate_ahead_time: 2.0
max_rotational_vel: 2.0 #1.0
min_rotational_vel: 0.6 #0.4
rotational_acc_lim: 4.0 #3.2
```

회전 속도, 가속도 제어

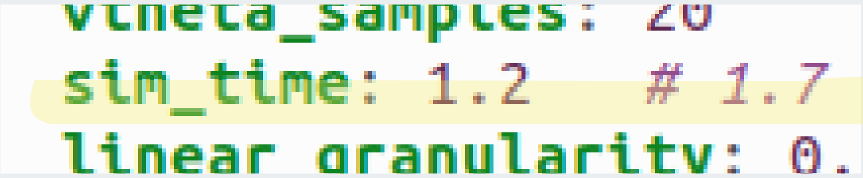
controller_server



```
general_goal_checker:
  stateful: True
  plugin: "nav2_controller::Simple
  xy_goal_tolerance: 0.05 #0.25
  yaw_goal_tolerance: 0.1 #0.25
```



```
FollowPath:
  plugin: "dwb_core::DWBLocalPlan
  debug_trajectory_details: True
  min_vel_x: 0.0
  min_vel_y: 0.0
  max_vel_x: 0.13 # 0.26
  max_vel_y: 0.0
  max_vel_theta: 0.7854 # 1.0
  min_speed_xy: 0.0
  max_speed_xy: 0.13 # 0.26
```



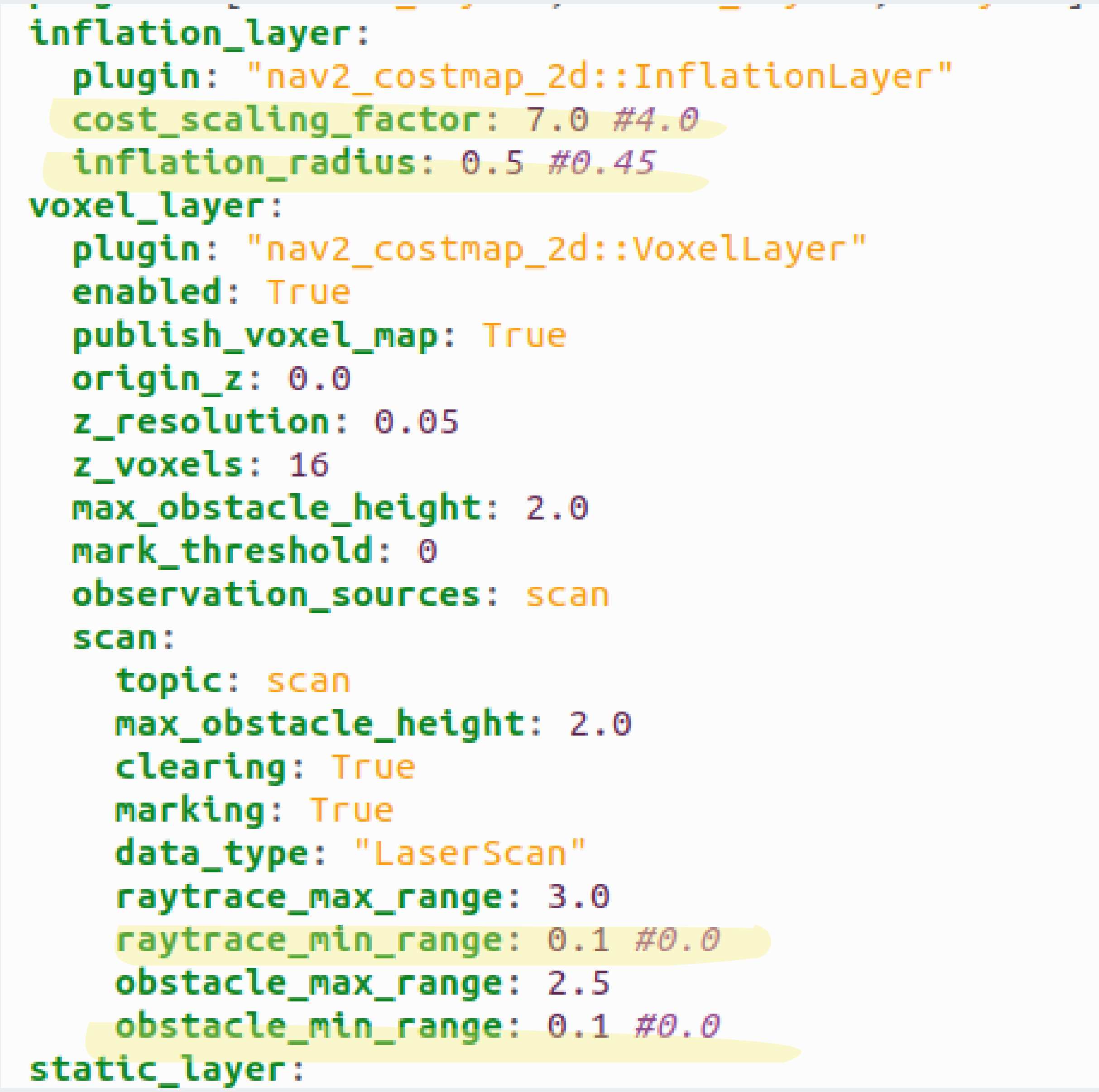
```
  vtheta_samples: 20
  sim_time: 1.2    # 1.7
  linear_granularity: 0.
```

도달 거리, 각도 판단 오차
도달 거리, 각도 계산

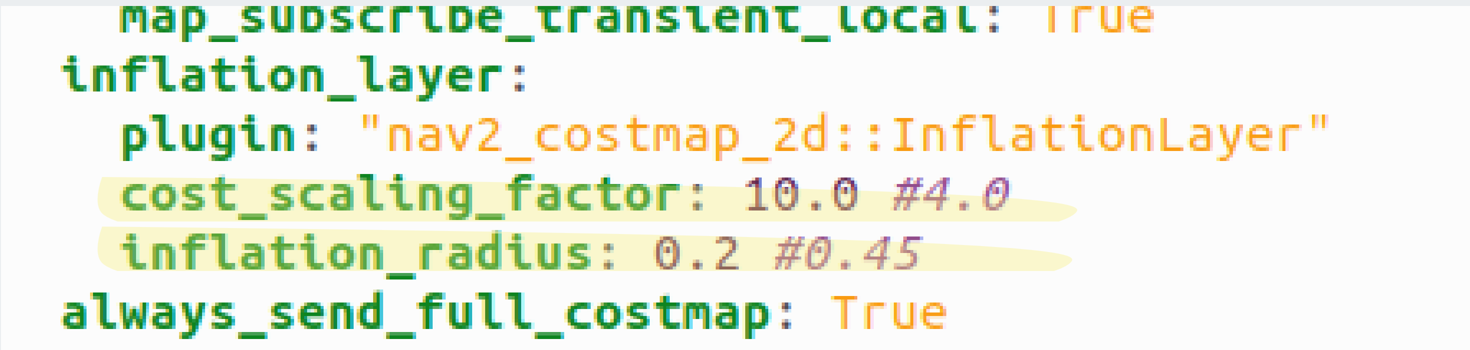
PARAMETER TUNNING

local_costmap

global_costmap



```
inflation_layer:
  plugin: "nav2_costmap_2d::InflationLayer"
  cost_scaling_factor: 7.0 #4.0
  inflation_radius: 0.5 #0.45
voxel_layer:
  plugin: "nav2_costmap_2d::VoxelLayer"
  enabled: True
  publish_voxel_map: True
  origin_z: 0.0
  z_resolution: 0.05
  z_voxels: 16
  max_obstacle_height: 2.0
  mark_threshold: 0
  observation_sources: scan
  scan:
    topic: scan
    max_obstacle_height: 2.0
    clearing: True
    marking: True
    data_type: "LaserScan"
    raytrace_max_range: 3.0
    raytrace_min_range: 0.1 #0.0
    obstacle_max_range: 2.5
    obstacle_min_range: 0.1 #0.0
static_layer:
```



```
  map_subscribe_transient_local: True
inflation_layer:
  plugin: "nav2_costmap_2d::InflationLayer"
  cost_scaling_factor: 10.0 #4.0
  inflation_radius: 0.2 #0.45
always_send_full_costmap: True
```

장애물 주변의 비용 계산

레이저 스캔 데이터에서 무시할 최소 거리 범위

THANK YOU