AIM

To design and simulate a sampling-based motion planning algorithm (RRT) for an autonomous agricultural robot that navigates through a field with obstacles (crop rows) to efficiently perform tasks such as soil sampling and weed detection, while avoiding collisions and visualizing the planned trajectory.

ALGORITHM (RRT – Rapidly Exploring Random Tree)

RRT is a randomized path-planning algorithm well-suited for high-dimensional or cluttered environments.

Key Steps

1. Initialize the tree with the robot's start position.
2. Randomly sample a point in the field.
3. Find the nearest node in the tree.
4. Extend the tree toward the sampled point by a fixed step size.
5. Check for collisions with obstacles (crop rows).
6. Add the new node if collision-free.
7. Repeat until the goal is reached.
8. Extract the path from start to goal.

PSEUDOCODE

```
RRT(start, goal, map):
    Tree ← start
    while goal not reached:
        X_rand ← random sample
        X_near ← nearest(Tree, X_rand)
        X_new ← steer(X_near, X_rand)
        if collision_free(X_near, X_new):
            add X_new to Tree
            set parent(X_new) = X_near
            if distance(X_new, goal) < threshold:
                connect goal
                return path
    return failure
```

INPUT

- Field size (2D grid)
- Start position of robot
- Goal position (sampling/weed detection point)
- Obstacles (crop rows)
- Step size
- Maximum iterations

OUTPUT

- Collision-free trajectory from start to goal
- Visualization of:
    - Obstacles (crop rows)
    - RRT tree
    - Final path

PYTHON IMPLEMENTATION (Simulation + Visualization)

```
import random import math import matplotlib.pyplot as plt # Environment size
WIDTH, HEIGHT = 100, 100 # Start and Goal START = (10, 10) GOAL = (90, 90) #
Obstacles (crop rows as rectangles) OBSTACLES = [ (20, 0, 10, 70), (40, 30, 10,
70), (60, 0, 10, 70), ] STEP_SIZE = 5 MAX_ITER = 5000 class Node: def
__init__(self, x, y, parent=None): self.x = x self.y = y self.parent = parent
def distance(n1, n2): return math.hypot(n1.x - n2.x, n1.y - n2.y) def
collision(x, y): for ox, oy, w, h in OBSTACLES: if ox <= x <= ox + w and oy <=
y <= oy + h: return True return False def steer(from_node, to_point): angle =
math.atan2(to_point[1] - from_node.y, to_point[0] - from_node.x) new_x =
from_node.x + STEP_SIZE * math.cos(angle) new_y = from_node.y + STEP_SIZE *
math.sin(angle) return Node(new_x, new_y, from_node) def rrt(): tree =
[Node(*START)] for _ in range(MAX_ITER): rand_point = (random.uniform(0,
WIDTH), random.uniform(0, HEIGHT)) nearest = min(tree, key=lambda node:
math.hypot(node.x - rand_point[0], node.y - rand_point[1])) new_node =
steer(nearest, rand_point) if not collision(new_node.x, new_node.y):
```

```
tree.append(new_node) if math.hypot(new_node.x - GOAL[0], new_node.y - GOAL[1])
< STEP_SIZE: goal_node = Node(*GOAL, new_node) tree.append(goal_node) return
tree, goal_node return tree, None def extract_path(goal_node): path = [] node =
goal_node while node: path.append((node.x, node.y)) node = node.parent return
path[::-1] # Run RRT tree, goal_node = rrt() path = extract_path(goal_node) #
Visualization plt.figure(figsize=(8, 8)) # Obstacles for ox, oy, w, h in
OBSTACLES: plt.gca().add_patch( plt.Rectangle((ox, oy), w, h, color='green',
alpha=0.6)) # Tree for node in tree: if node.parent: plt.plot([node.x,
node.parent.x], [node.y, node.parent.y], 'gray', linewidth=0.5) # Path if path:
px, py = zip(*path) plt.plot(px, py, 'r', linewidth=3, label="Planned Path") #
Start and Goal plt.scatter(*START, color='blue', s=100, label="Start")
plt.scatter(*GOAL, color='red', s=100, label="Goal") plt.title("RRT Path
Planning for Agricultural Robot") plt.xlim(0, WIDTH) plt.ylim(0, HEIGHT)
plt.legend() plt.grid() plt.show()
```

RESULT

- The robot successfully plans a collision-free trajectory through crop rows.
- The RRT tree explores the field, adapting to obstacle constraints.
- The final red path represents an efficient route for soil sampling or weed detection.
- Visualization confirms feasibility in agricultural environments.