Homework:02 For CSCI6650 - P001 Intelligent Agents

Sourav Raxit

February 2, 2024

Problem Statement

Implement a robot patrol algorithm in CoppeliaSim that navigates randomly within a 10-meter by 10-meter rectangular area, avoiding a polygon-shaped obstacle.

The bottom-left corner of the rectangular area is located at coordinates (-5, -5). The top-right corner of the rectangular area is located at coordinates (5, 5).

Algorithm Pseudocode

```
FUNCTION getObjectState(target)
    # Retrieve the current position of the target object in the simulation
    p = sim.getObjectPosition(target, -1)
    RETURN [p[0], p[1]]

FUNCTION isPointInsideTriangle(A, B, C, P)
    # Determine if a point is inside a triangle using determinants
    M = [[B[0] - A[0], C[0] - A[0]], [B[1] - A[1], C[1] - A[1]]]
    M_AP = [[B[0] - A[0], P[0] - A[0]], [B[1] - A[1], P[1] - A[1]]]
    det_M = M[0][0] * M[1][1] - M[0][1] * M[1][0]
    det_M_AP = M_AP[0][0] * M_AP[1][1] - M_AP[0][1] * M_AP[1][0]
    RETURN 0 < det_M * det_M_AP AND det_M * det_M_AP <= det_M * det_M

FUNCTION sysCallInit()
    sim = require('sim')
    # Initialize necessary variables and set up the simulation environment

FUNCTION sysCallThread()
    obstacle = np.array([[-2.5, 2.5], [0, 0.83333333], [2.5, -2.5], [-2.5, 0.83333333]])
    ws = [[5, 5], [5, -5], [-5, -5], [-5, 5]]
    U = [[0, 0.1], [0, -0.1], [0.1, 0], [-0.1, 0]]
    target = sim.getObject("/target")

    WHILE NOT sim.getSimulationStopping()
        x = getObjectState(target)
        u = RANDOM.choice(U)
        nx = [x[0] + u[0], x[1] + u[1]]

        IF isPointInsideTriangle(ws[0], ws[1], ws[2], nx) AND isPointInsideTriangle(ws[0], ws[3], ws[2], nx)
            IF NOT isPointInsideTriangle(obstacle[0], obstacle[1], obstacle[2], nx) AND NOT isPointInsideTriangle(obstacle[0], obstacle[3], obstacle[2], nx)
                sim.setObjectPosition(target, -1, [nx[0], nx[1], 0.5])
                sim.step()  # Resume in the next simulation step
```
Listing 1: Algorithm Pseudocode

Discussion

Workspace Division:

- **Description:** The rectangular area in the simulation is logically split into two triangles. This division likely serves as a method for simplifying geometric calculations or spatial checks.
- **Significance:** The division into triangles could be motivated by the ease of implementing algorithms that involve triangle-based spatial queries, such as determining whether a point lies inside a triangle.

Obstacle Avoidance:

- **Description:** The obstacle, represented as a polygon, is also divided into two triangles. The algorithm assesses whether the new target position falls within these triangles. If it does, the movement step is not executed to circumvent the obstacle.
- **Significance:** Dividing the obstacle into triangles facilitates the use of geometric methods for obstacle avoidance. Checking containment

within triangles allows for a straightforward determination of whether a point is inside the obstacle, simplifying collision avoidance logic.

Random Action Selection:

- **Description:** The algorithm introduces a random element into the robot's movement by selecting a control input randomly from a predefined set (`U`). This set contains different directional movements such as North, South, East and West.
- **Significance:** Randomizing the robot's actions adds an element of unpredictability to its behavior. This randomness is useful for exploring different parts of the workspace, contributing to more dynamic and varied patrol patterns.

Considerations:

- **Description:** This section outlines critical considerations for the algorithm's effective use.
- **Leveraging Geometric Methods:** The algorithm relies on geometric methods, specifically determinants, to efficiently determine spatial relationships. This choice suggests a focus on computational efficiency and accuracy in spatial calculations.
- **Accurate Definition of Elements:** Emphasizes the importance of accurately defining the workspace, obstacle shape, and logic for obstacle avoidance. Precision in these definitions is crucial for the reliable functioning of the algorithm within specified constraints.