Autonomous Robot Navigation in MATLAB

Almansur Kakimov

February 2025

1 Introduction

This document provides a comprehensive explanation of a MATLAB-based robot navigation system. It covers environment setup, localization, path planning, motion control, and obstacle avoidance.

2 Environment Setup

The robot operates on a 10×10 grid with random obstacles and goal positions.

2.1 MATLAB Code for Environment Setup

```
mapSize = [10, 10];
numObstacles = round(0.15 * prod(mapSize));
obstacles = randi([1, 10], numObstacles, 2);
goal = randi([1, 10], 1, 2);

disp("Environment setup complete.");
```

3 Localization

The robot tracks its position using odometry, which updates based on motion commands.

3.1 MATLAB Code for Odometry

```
function odometryPos = updateOdometry(odometryPos, robotPos,
    robotAngle, robotSpeed)
    odometryPos = odometryPos + robotSpeed * [cos(robotAngle), sin(
        robotAngle)];
end
```

4 Path Planning: A* Algorithm

The A* search algorithm finds the shortest path to the goal while avoiding obstacles.

4.1 MATLAB Code for A* Pathfinding

```
path = aStarPathfinding(robotPos, goal, obstacles, mapSize);
```

5 Path Smoothing: Bézier Curves

To ensure smooth navigation, a Bézier curve interpolates the A* path.

5.1 MATLAB Code for Bézier Curve Generation

```
function smoothPath = bezierSmoothPath(path)
    t = linspace(0, 1, 100);
    n = size(path, 1) - 1;
    smoothPath = zeros(length(t), 2);
    for i = 0:n
        B = nchoosek(n, i) .* (t .^ i) .* ((1 - t) .^ (n - i));
        smoothPath = smoothPath + B' * path(i + 1, :);
    end
end
```

6 Motion Execution

The robot gradually turns toward the next waypoint before moving forward.

6.1 MATLAB Code for Motion Control

```
direction = nextPos - robotPos;
targetAngle = atan2(direction(2), direction(1));
angleError = targetAngle - robotAngle;
angleError = mod(angleError + pi, 2 * pi) - pi;
```

7 Obstacle Avoidance

If an obstacle is detected nearby, the robot dynamically adjusts its trajectory.

7.1 MATLAB Code for Obstacle Avoidance

```
for i = 1:length(xObst)
    dist = sqrt((xObst(i) - robotPos(1))^2 + (yObst(i) - robotPos
        (2))^2);
    if dist < proximityRadius
        angleToObstacle = atan2(yObst(i) - robotPos(2), xObst(i) -
            robotPos(1));
        avoidanceAngle = angleToObstacle + pi / 2;
        robotPos = robotPos + [cos(avoidanceAngle) * 0.5, sin(
            avoidanceAngle) * 0.5];
        break;
    end
end
```

8 Summary of Key Algorithms

Functionality	Algorithm Used
Localization	Odometry-based tracking
Path Planning	A* Algorithm
Path Smoothing	Bézier Curve Interpolation
Motion Control	Angle-based rotation + speed control
Obstacle Avoidance	Proximity-based reactive control

Table 1: Overview of Navigation Algorithms

9 Conclusion

The integration of A* pathfinding, Bézier smoothing, and reactive obstacle avoidance ensures robust performance in a simulated environment.