EE 615
Experiment No. 2

Path Following and Obstacle Avoidance using Pure Pursuit and VFH

Lab report submitted by

Group 12
Drishtant Jain: 24M1085
Gagan G: 24M1090

April 08, 2025

Under the guidance of

Prof. Debasattam Pal
Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Contents

1 Introduction

This experiment aims to explore the implementation of autonomous navigation techniques in mobile robots, specifically focusing on path following with obstacle avoidance. The task is to guide a robot from a given start location $(2, 4)$ to a final target location $(10, 4)$, passing through an intermediate point at $(2, 10)$. During its motion, the robot must detect and avoid obstacles placed in its environment while ensuring that it follows the specified path as closely as possible.

To achieve this, two fundamental algorithms are implemented and tested: the Pure Pursuit path following algorithm and the Vector Field Histogram (VFH) obstacle avoidance method. The Pure Pursuit algorithm is responsible for steering the robot along a series of predefined waypoints, using a geometric approach based on lookahead distance. The VFH algorithm, on the other hand, reacts to nearby obstacles by dynamically computing safe directions to avoid collisions.

The experiment was carried out in a simulated environment using MATLAB and Simulink. A template Simulink model named `pathFollowingWithObstacleAvoidanceExample.slx` was provided, along with a MATLAB script `Experiment2.m`. The model includes subsystems for path following and obstacle avoidance, which were incomplete and required implementation by the student. The subsystems were filled in with custom logic for the Pure Pursuit and VFH algorithms as per the instructions in the code comments.

Through this experiment, we gained practical experience in implementing real-time control algorithms in a simulation-based robotics framework. We observed how the robot's trajectory changes in response to different tuning parameters like the lookahead distance in Pure Pursuit and the threshold in VFH. The experiment also highlighted the importance of integrating multiple control strategies to ensure robust navigation in uncertain environments.

The following sections of this report detail the control strategy used, the mathematical derivations involved, the implemented code, simulation results, and the observations made during the course of the experiment.

2 Aim

The aim of this experiment is to implement a control strategy that enables a mobile robot to follow a given path while avoiding obstacles in its environment. Specifically, the robot is required to move from the start position at $(x, y) = (2, 4)$ to the target position at $(x, y) = (10, 4)$ via an intermediate waypoint at $(x, y) = (2, 10)$.

3 Objectives

- Understand and implement the **Pure Pursuit** algorithm for following a sequence of waypoints in a 2D environment.
- Implement the **Vector Field Histogram (VFH)** algorithm to enable dynamic and local obstacle avoidance based on sensor data.
- Integrate both control strategies within a **Simulink** model and simulate the robot's trajectory using the provided ROS-compatible environment.

- Tune the control parameters to ensure smooth path tracking and effective obstacle avoidance, while maintaining:
  - Accurate traversal through intermediate waypoint $(2, 10)$ and final goal at $(10, 4)$.
  - Safe navigation without collision, even in the presence of multiple obstacles.

4 Control Algorithm

The **control algorithm flowchart** for path following with obstacle avoidance is shown below. It outlines the high-level logic used to simulate the robot's movement through waypoints while avoiding obstacles using Pure Pursuit and VFH methods.

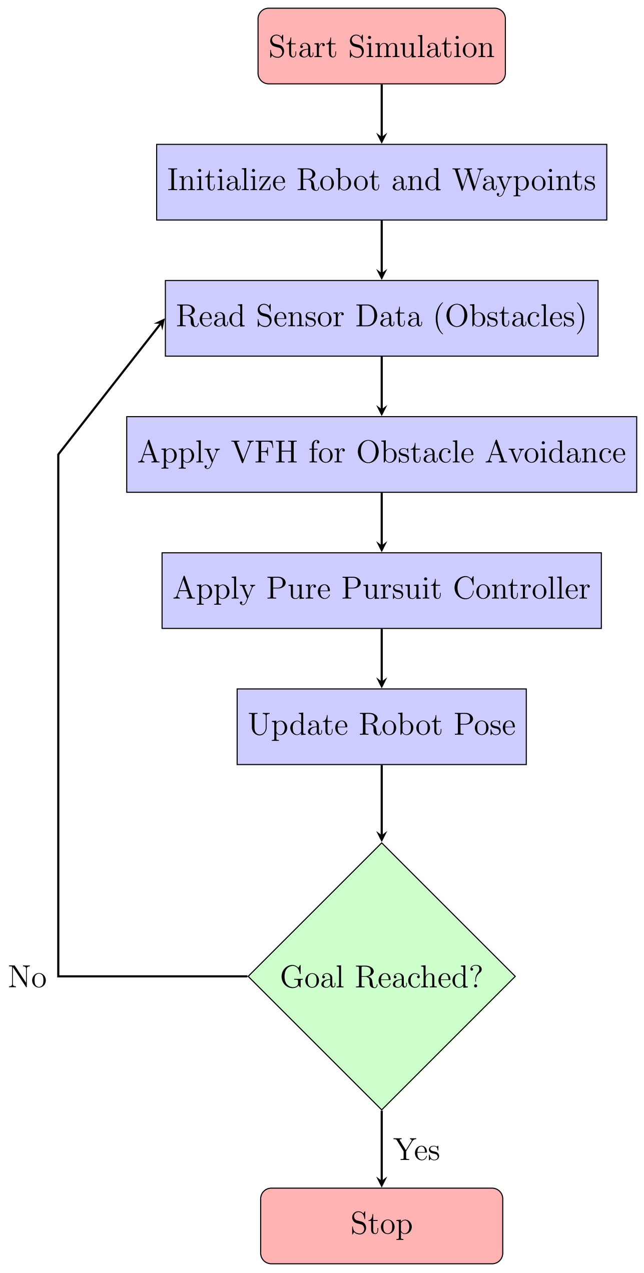
Start Simulation
Initialize Robot and Waypoints
Read Sensor Data (Obstacles)
Apply VFH for Obstacle Avoidance
Apply Pure Pursuit Controller
Update Robot Pose
Goal Reached?
No
Yes
Stop

Figure 1: Control Algorithm Flowchart for Path Following with Obstacle Avoidance

5 Algorithms Involved

This experiment involves two key algorithms — Pure Pursuit for path following and Vector Field Histogram (VFH) for obstacle avoidance. Each of these algorithms is based on distinct mathematical principles, as described below.

1. Pure Pursuit Path Tracking

The Pure Pursuit algorithm determines the curvature of a circular arc that will steer the robot from its current location to a goal point located along the reference path. This goal point is placed at a fixed distance ahead of the robot, known as the **lookahead distance**, denoted by L_d.

Let the coordinates of the goal point in the robot's local frame be (x, y). The radius R of the arc that connects the robot's current position to the goal point is given by:

$$R = \frac{L_d^2}{2y} \tag{1}$$

The curvature κ of the arc is the reciprocal of the radius:

$$\kappa = \frac{2y}{L_d^2} \tag{2}$$

If the vehicle's wheelbase is L, the required steering angle δ can be computed using:

$$\delta = \tan^{-1}(L \cdot \kappa) \tag{3}$$

This approach simplifies path tracking into a geometric problem and provides smooth, human-like trajectories. As shown in Coulter's work [?], the effectiveness of the algorithm depends heavily on the selection of the lookahead distance L_d — smaller values yield tighter turns but may cause instability, while larger values result in smoother but less responsive behavior.

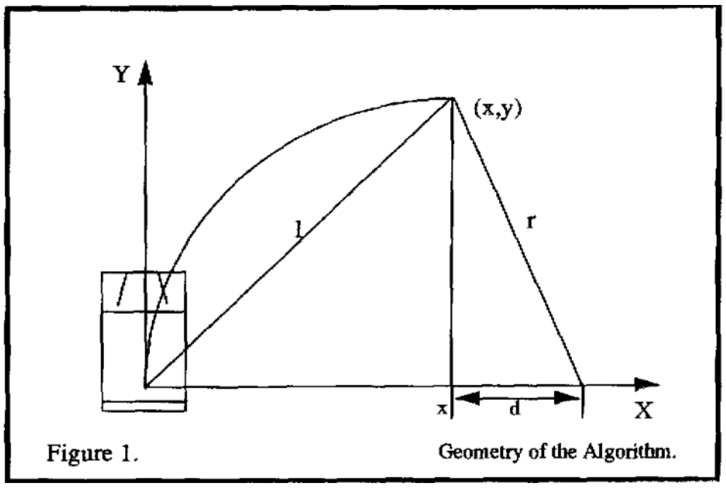

Figure 2: Pure Pursuit Geometry: Robot turning towards a goal point at distance L_d

2. Vector Field Histogram (VFH)

The VFH algorithm is a local obstacle avoidance strategy that transforms sensor data into a polar histogram to represent the obstacle density in various directions around the robot. It then selects a steering direction that is both obstacle-free and aligned closely with the desired path.

Histogram Construction: Sensor readings are accumulated in a 2D Cartesian histogram grid C, centered around the robot. Each cell (i, j) in the grid stores a **certainty value** $c_{i,j}$ indicating the likelihood of an obstacle's presence.

To create the polar histogram H, the active region C^* of size $w \times w$ centered on the robot is projected onto a set of n angular sectors. Each sector k in the polar histogram contains the **polar obstacle density** h_k calculated as:

$$h_k = \sum_{i,j \in C^*} (c^*_{i,j})^2 (a - b d_{i,j}) \tag{4}$$

Where:

- $c^*_{i,j}$ is the certainty value of cell (i, j)
- $d_{i,j}$ is the distance from cell (i, j) to the robot center
- a, b are constants to weigh distance influence

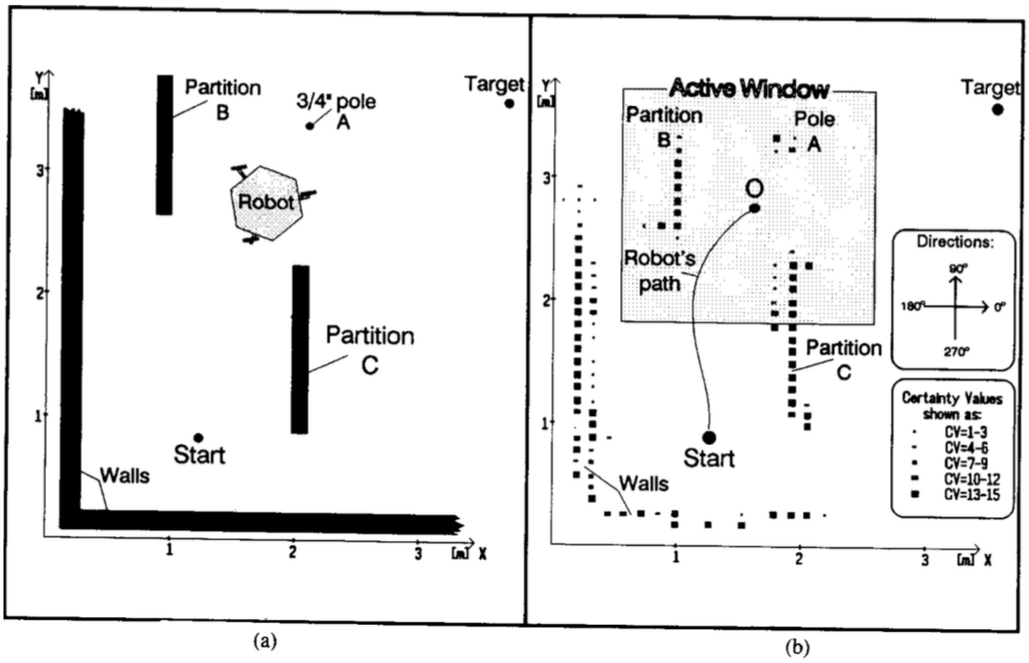

Fig. 5. (a) Example of an obstacle course. (b) The corresponding histogram grid representation.

Figure 3: (a) Example of an obstacle course layout with partitions and poles. (b) Corresponding histogram grid showing certainty values and active window. Source: Borenstein and Koren, 1991 [?].

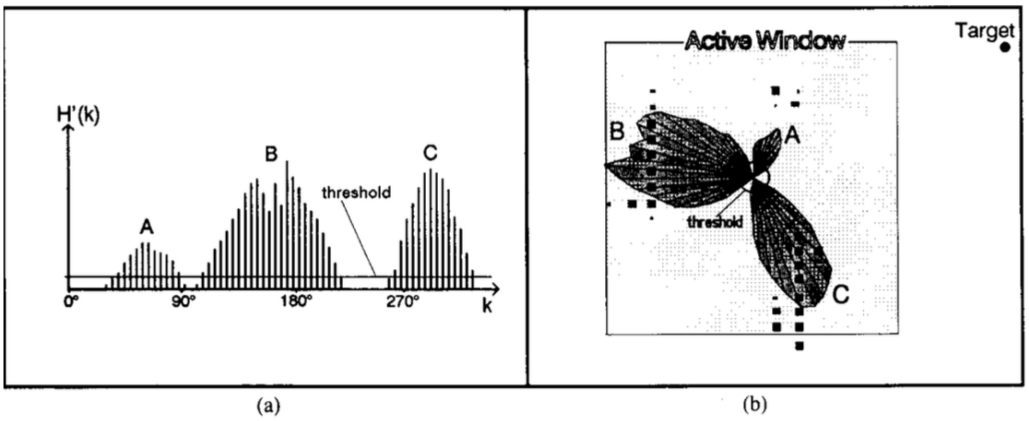

Fig. 6. (a) Polar obstacle density represented in the smoothed polar histogram $H'(k)$—relative to the robot's position at O [in Fig. 5(b)]. (b) The same polar histogram as in (a), shown in polar form and overlaying part of the histogram grid of Fig. 5(b).

Figure 4: (a) Smoothed polar obstacle density histogram $H'(k)$ showing sectors with obstacles A, B, and C. (b) Same histogram shown in polar form overlaid on the histogram grid. Source: Borenstein and Koren, 1991 [?].

Candidate Valley Selection: A threshold is applied to identify "valleys" (directions with low h_k values). Among these, the direction closest to the target heading is selected as the desired steering angle. The method is reactive and allows the robot to adapt quickly to unknown obstacles without prior mapping.

3. Combined Control Architecture

The overall control algorithm combines both strategies in a hybrid framework:

- The VFH module detects obstacles and outputs a safe direction for motion.
- The Pure Pursuit controller computes curvature to follow the reference path.
- The robot's actual heading is adjusted based on the blend of both strategies.

This hybrid structure ensures that while the robot remains goal-oriented via Pure Pursuit, it also retains the flexibility to deviate momentarily in the presence of obstacles using VFH.

6 Code Implementation and Explanation

This section includes the actual MATLAB functions developed and used in this experiment for the Pure Pursuit and Vector Field Histogram (VFH) control strategies.

1. Pure Pursuit Algorithm

The Pure Pursuit function tracks a list of waypoints and computes linear and angular velocity commands to move the robot toward the next point using a geometric lookahead approach.

Listing 1: Pure Pursuit Function

```
function [LinVel, AngVel, TargetDir] = PursuitFunction(CurrentPose, WayPoints)
L = 1;
robotX = CurrentPose(1);
robotY = CurrentPose(2);
robotHeading = CurrentPose(3);
sizeWayPoints = size(WayPoints);
LinVel = 0.5;
persistent visitStatus
if(isempty(visitStatus))
    visitStatus = zeros(1,sizeWayPoints(1));
end
for i = 1:sizeWayPoints(1)
    if(visitStatus(i) == 0)
        if(i < sizeWayPoints(1))
            if((robotX - WayPoints(i,1))^2 + (robotY - WayPoints(i,2))^2 < L^2)
                visitStatus(i) = 1;
            end
        else
            if((robotX - WayPoints(i,1))^2 + (robotY - WayPoints(i,2))^2 < 0.1)
                visitStatus(i) = 1;
            end
        end
    end
end
currentWaypoint = zeros(1,2);
for i = 1:sizeWayPoints(1)
    if(visitStatus(i) == 0)
        currentWaypoint(1) = WayPoints(i,1);
        currentWaypoint(2) = WayPoints(i,2);
        break
    end
end
goalPoint = zeros(1,2);
distance2WP = sqrt((currentWaypoint(1) - robotX)^2 + (currentWaypoint(2) - robotY)^2);
p1X = robotX + L*(currentWaypoint(1) - robotX)/distance2WP;
p1Y = robotY + L*(currentWaypoint(2) - robotY)/distance2WP;
p2X = robotX - L*(currentWaypoint(1) - robotX)/distance2WP;
p2Y = robotY - L*(currentWaypoint(2) - robotY)/distance2WP;
if((p1X - currentWaypoint(1))^2 + (p1Y - currentWaypoint(2))^2 < (p2X - currentWaypoint(1))^2 + (p2Y - currentWaypoint(2))^2)
    goalPoint(1) = p1X;
    goalPoint(2) = p1Y;
else
    goalPoint(1) = p2X;
    goalPoint(2) = p2Y;
```

```
end
R = (L^2)/(2*(goalPoint(1) - robotX));
% R = (L^2)/(2*(distance2WP*sin(atan2(currentWaypoint(2) -
    ↪ robotY, currentWaypoint(1) - robotX) - robotHeading)));
AngVel = LinVel/R;
TargetDir = -robotHeading + atan2(goalPoint(2) - robotY,
    ↪ goalPoint(1) - robotX);
```

Explanation: This function keeps track of visited waypoints, computes a new goal point at a distance L, and generates a steering command using curvature-based turning.

2. Vector Field Histogram (VFH) Algorithm

This function processes LiDAR sensor data to identify obstacle-free sectors in a polar histogram and selects a direction closest to the target.

Listing 2: VFH Algorithm for Obstacle Avoidance

```
function steering_angle = VFHfunction(ranges, angles,
    ↪ target_direc)

a = 6;
d_max = 6;
b = a/d_max;
alpha = pi/180;
N_sectors = 181;
threshold = 2;
% s_max = 4; first good run
% s_max = 7;
s_max = 10;
L = 10;
h_out = zeros(N_sectors,1);
h_avg_out = zeros(N_sectors,1);
h_bin_out = zeros(N_sectors,1);
angle_index = int32(angles./alpha);
min_angle_index = min(angle_index(1:21));
angle_index = (angle_index - min_angle_index) + 1;

%% calculating h_out

for i = 1:N_sectors
    for j = 1:21
        if(angle_index(j) == i)
            if(~isnan(ranges(j)))
                c = 1;
                m = (c^2)*(a - b*ranges(j));
                h_out(i) = h_out(i) + m;
            end
        end
    end
end
```

```matlab

%% calculating h_avg_out

for i = 1:N_sectors
    for j = -L:L
        if(i+j <= N_sectors) && (i+j >=1)
            h_avg_out(i) = h_avg_out(i) +
               ↪ (L-abs(j)+1)*h_out(i+j);
        end
    end
    h_avg_out(i) = h_avg_out(i)/(2*L + 1);
end

%% calculating h_bin_out

for i = 1:N_sectors
    if(h_avg_out(i) > threshold)
        h_bin_out(i) = 1;
    end
end

%% valley identification

valley_started = 0;
valley_count = 0;
valley_arr = 200*ones(1, 30, 'int32');
for i = 1:N_sectors
    if(valley_started == 0)
        if(h_bin_out(i) == 0)
            valley_count = valley_count + 1;
            valley_arr(2*valley_count - 1) = i;
            valley_started = 1;
        end
    else
        if(h_bin_out(i) == 1)
            valley_arr(2*valley_count) = i;
            valley_started = 0;
        end
    end
end
if(valley_count ~= 0)
    if(valley_arr(2*valley_count) == 200)
        valley_arr(2*valley_count) = 181;
    end
end

%% valley selection
if(valley_count > 0)
    k_target = int32((target_direc + pi/2)/alpha);
    candidate_valley_index = 1;
    delta_k = int32(200);
    delta_sign = 1;
    for i = 1:valley_count
        for j = valley_arr(2*i - 1):valley_arr(2*i)
            if(delta_k > abs(j - k_target))
                delta_k = abs(j - k_target);
                candidate_valley_index = i;
                if(delta_k ~= 0)
                    delta_sign = double((j - k_target)/abs(j - k_target));
                end
            end
        end
    end
    candidate_valley = [valley_arr(2*candidate_valley_index - 1), valley_arr(2*candidate_valley_index)];

%% steering angle calculation
    kn = k_target + delta_sign*delta_k;
    sector_count = abs(candidate_valley(1) - candidate_valley(2));
    if(sector_count > s_max)
        kf = kn + s_max;
        if((kf > min(candidate_valley)) && (kf < max(candidate_valley)))
            kf = kn + s_max;
        else
            kf = kn - s_max;
        end
        kc = (kn + kf)/2;
    else
        kf = kn + sector_count;
        if((kf > min(candidate_valley)) && (kf < max(candidate_valley)))
            kf = kn + sector_count;
        else
            kf = kn - sector_count;
        end
        kc = (kn + kf)/2;
    end
    k_arr = [kn, kc, kf];
    steering_angle = double(kc)*alpha - pi/2;
else
    k_arr = int32([0, 0, 0]);
    steering_angle = NaN;
end
```

Explanation: The VFH algorithm identifies valleys of low obstacle density and chooses the sector that best aligns with the target direction. The histogram is smoothed and thresholded to ensure robustness.

7 Results and Observations

This section documents the performance of the hybrid control system combining Pure Pursuit and VFH, using a simulated environment in Simulink. The robot was tasked with navigating from the start location $(2, 4)$ to the goal $(10, 4)$ via the intermediate point $(2, 10)$, while avoiding obstacles dynamically.

1. Simulation Parameters

The following parameters were used in the final simulation:

- Number of histogram sectors: 180
- Histogram constant: $a = 6$
- Distance scaling constant: $d_{\max} = 6$
- Threshold for obstacle density: 2.0
- Lookahead distance (L): 1.0 m
- Valley selection span ($s_{\max}$): 4 sectors
- Linear velocity: 0.5 m/s
- Constants $b = 1$, $c = 1$

2. Path Variation with Different Waypoints

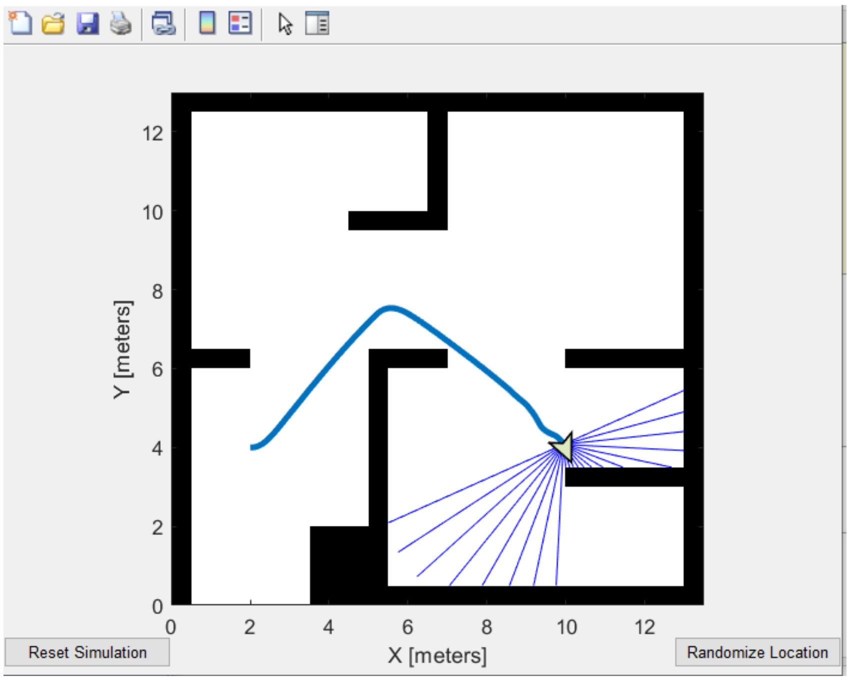

Figure 5: Robot trajectory for Waypoints Set 1 [2 4;6 8;10 4] — smoother curve through central corridor to reach goal

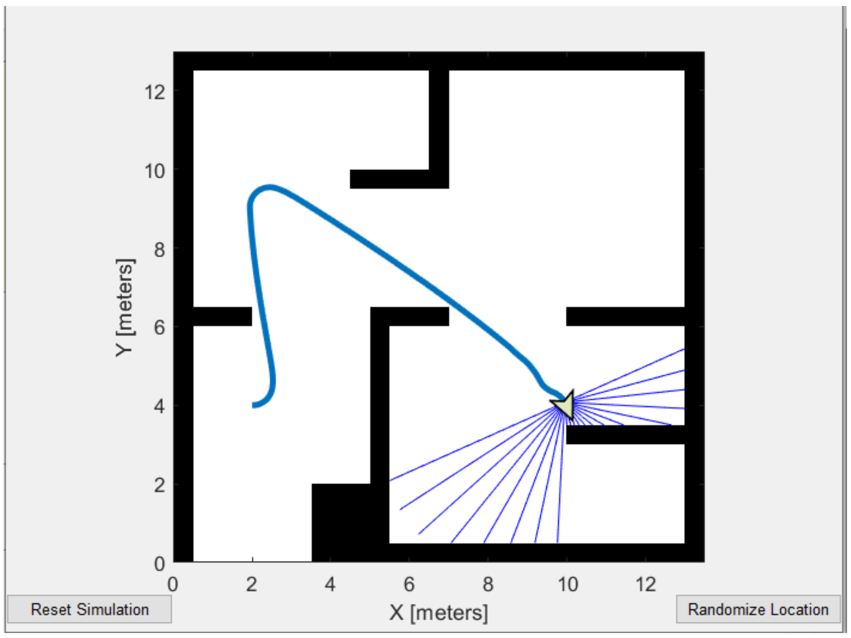

Figure 6: Robot trajectory for Waypoints Set 2 [2 4;2 10;10 4] — tighter turns in upper-left corner before realignment

- The robot adapts its motion based on the relative layout of the waypoints and nearby obstacles.
- In Figure 5, the robot takes a central route with minimal steering effort.
- In Figure 6, the route introduces tight maneuvering before aligning with the corridor, demonstrating the responsiveness of the VFH controller.

8 Challenges Faced and Solutions

During the course of implementing and simulating the hybrid navigation strategy, several challenges were encountered. The following table summarizes the issues and the corresponding solutions applied:

- **Overshooting Near Waypoints:**
  The robot occasionally overshot the intermediate waypoint, especially during sharp turns or when transitioning between open and narrow spaces.

  Solution: The lookahead distance L in the Pure Pursuit algorithm was reduced from 1.5 to 1.0 to allow for tighter curvature and more accurate tracking near critical junctions.

- **Narrow Corridor Navigation Failure:**
  In tight passages, VFH initially failed to select a valley due to dense obstacle data across sectors.

  Solution: The valley width parameter $s_{\max}$ was decreased to 4 to enable selection of narrow but passable corridors.

9 Conclusion

This experiment successfully demonstrated the integration of a path-following controller (Pure Pursuit) with a real-time obstacle avoidance mechanism (Vector Field Histogram) in a simulated mobile robot navigation task. The objective was to guide the robot from a start location to a target via an intermediate waypoint while dynamically avoiding obstacles in an unknown environment.

The Pure Pursuit controller effectively maintained the robot's trajectory by computing curvature toward a lookahead goal point along the path. The VFH algorithm operated in parallel to detect nearby obstacles using Lidar data and re-route the robot through safe steering valleys. The combined use of these two algorithms provided a robust navigation strategy capable of adapting to both global path planning and local reactive control requirements.