# MATLAB SIMULATION OF OBSTACLE AVOIDING ROBOT WITH POTENTIAL FIELD PATH PLANNING

**By**

***V G MANASA 24BEC1419***

***SHRUTHI NARAYANAN 24BEC1458***

***JANANI S K 24BEC1540***

***Under the guidance of***

**Dr. Bharathi Sankar**


**SCHOOL OF SENSE**

**VELLORE INSTITUTE OF TECHNOLOGY, CHENNAI-600127**

# TABLE OF CONTENTS

# ABSTRACT

This project presents a MATLAB simulation of an obstacle-avoiding robot using Potential Field Path Planning (PFPP). The robot navigates a 2D environment toward a target while avoiding obstacles by responding to artificial forces: an attractive force from the goal and repulsive forces from obstacles. The resultant force determines the robot's direction and speed, allowing smooth, real-time path planning and collision-free navigation. The simulation visualizes the robot's trajectory, obstacle positions, and potential field gradients, providing insight into autonomous navigation strategies. This approach demonstrates the effectiveness of PFPP in robotic control and highlights its practical applications in mobile robotics, real-time decision making, and path optimization.

# OBJECTIVE

The main objective of this project is to design and simulate an autonomous robot capable of navigating a 2D environment while avoiding obstacles using Potential Field Path Planning. Specifically, the project aims to:

1. Implement the attractive and repulsive potential fields to guide the robot toward a target while avoiding obstacles.
2. Simulate the robot's real-time path planning and collision avoidance in MATLAB.
3. Visualize the robot's trajectory, obstacles, and target position to understand the effectiveness of the potential field method.
4. Analyse the performance of the robot in terms of efficiency, safety, and smoothness of path.

# INTRODUCTION

Autonomous robots are increasingly used in applications such as industrial automation, service robotics, and navigation systems. A key challenge in autonomous navigation is ensuring that a robot can efficiently reach a target while avoiding obstacles in its environment. Path planning algorithms play a crucial role in achieving this by determining safe and optimal trajectories for robot movement.

One widely used approach is Potential Field Path Planning, where the robot is influenced by a virtual attractive force pulling it toward the goal and repulsive forces pushing it away from obstacles. This method enables real-time trajectory adjustments, smooth navigation, and collision avoidance without the need for complex global maps.

In this project, we simulate an obstacle-avoiding robot in a 2D environment using MATLAB. The simulation demonstrates how potential fields can guide the robot along a safe path, while dynamically responding to the presence of obstacles. This provides a practical framework for understanding autonomous navigation, control algorithms, and real-time path planning in robotics.

# THEORY

## 1. Introduction to Autonomous Navigation

Autonomous robots are systems capable of performing tasks and navigating in an environment without human intervention. A critical requirement for such robots is the ability to plan safe and efficient paths from a starting point to a target location while avoiding obstacles. Path planning is a central problem in robotics and is often divided into global planning and local planning.

- **Global Planning**: The robot has prior knowledge of the entire environment, such as a map, and computes a path in advance. Techniques include A*, Dijkstra's algorithm, and graph search methods.
- **Local Planning**: The robot reacts in real-time to obstacles detected through sensors. Local planners focus on collision avoidance and immediate path adjustments.

Potential Field Path Planning (PFPP) is a reactive local planning algorithm that combines simplicity with real-time navigation capabilities, making it suitable for mobile robots operating in dynamic environments.

## 2. Potential Field Path Planning (PFPP)

PFPP is inspired by physics, where objects are influenced by forces. In robotics:

- The goal generates an attractive force that pulls the robot toward the target.
- Obstacles generate repulsive forces that push the robot away, preventing collisions.

The robot moves by following the resultant force vector, which is the sum of all attractive and repulsive forces. This continuous feedback allows the robot to navigate dynamically in the presence of static or moving obstacles.

### 2.1 Attractive Potential

The attractive potential $U_{att}$is typically proportional to the distance between the robot and the goal:

$$U_{att}(q) = \frac{1}{2} k_{att} \cdot \| q - q_{goal} \|^2$$

Where:

- $k_{att}$is the attractive coefficient.
- $q$is the current position of the robot.
- $q_{goal}$is the position of the goal.

The attractive force is the negative gradient of the potential:

$$F_{att} = -\nabla U_{att} = -k_{att}(q - q_{goal})$$

This force directs the robot toward the goal, with magnitude increasing as the robot moves further away.

### 2.2 Repulsive Potential

Obstacles exert a repulsive potential to prevent collisions. The potential $U_{rep}$is significant only within a certain distance $d_0$from the obstacle:

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left( \frac{1}{d(q)} - \frac{1}{d_0} \right)^2, & d(q) \le d_0 \\ 0, & d(q) > d_0 \end{cases}$$

Where:

- $k_{rep}$is the repulsive coefficient.
- $d(q)$is the distance between the robot and the obstacle.
- $d_0$is the influence range of the obstacle.

The repulsive force is:

$$F_{rep} = -\nabla U_{rep}$$

This force grows stronger as the robot approaches the obstacle, creating a "push" that prevents collisions.

**3. Resultant Force and Robot Motion**

The total potential at the robot's position is the sum of the attractive and repulsive potentials:

$$U(q) = U_{att}(q) + U_{rep}(q)$$

The robot's motion is determined by the negative gradient of the total potential:

$$F(q) = -\nabla U(q) = F_{att} + F_{rep}$$

This force vector gives both the direction and magnitude of movement. The robot continuously updates its position based on this force, resulting in smooth trajectories that avoid obstacles while moving toward the goal.

**4. Advantages of Potential Field Method**

1. **Simplicity**: Easy to implement mathematically and computationally.
2. **Real-time Navigation**: Suitable for dynamic environments as the robot reacts to obstacles immediately.

3. **Smooth Paths**: Produces continuous trajectories with no abrupt turns.
4. **Scalability**: Can be extended to multiple obstacles and goals.

## 5. Limitations

1. **Local Minima**: The robot may get stuck in positions where the net force is zero but not at the goal.
2. **Oscillations**: In narrow passages, the robot may oscillate due to opposing repulsive forces.
3. **Tuning Required**: Gains $k_{att}$ and $k_{rep}$ need careful adjustment for optimal performance.
4. **No Global Guarantee**: The method does not guarantee the shortest path, especially in complex environments.

**Solutions to Limitations**:

- Adding random perturbations or virtual guiding forces to escape local minima.
- Combining PFPP with global path planning algorithms like A* for hybrid approaches.

## 6. Extensions and Variations

- **Dynamic Obstacles**: PFPP can handle moving obstacles by continuously updating the repulsive potential.
- **3D Navigation**: Extending to aerial or underwater robots where potential fields are applied in 3D space.
- **Multi-Robot Systems**: Incorporating inter-robot repulsive forces to prevent collisions in swarm robotics.

- **Integration with Sensors**: Using LIDAR, ultrasonic, or vision sensors to detect obstacles and compute potentials in real-time.

**7. MATLAB Simulation**

In this project, MATLAB is used to:

- Model the 2D environment with target and obstacles.
- Compute attractive and repulsive potentials at each time step.
- Update the robot's position based on the resultant force.
- Visualize the robot trajectory, obstacle positions, and the potential field gradients.

This allows practical demonstration of PFPP and helps in analyzing path efficiency, obstacle avoidance, and system behavior under different conditions.

# METHODOLOGY

The methodology for simulating an **Obstacle Avoiding Robot using Potential Field Path Planning (PFPP)** in MATLAB focuses on creating a realistic 2D environment, modelling robot and obstacle dynamic, computing attractive and repulsive forces, updating robot motion iteratively, and visualizing the results in real time. The simulation combines theoretical principles with interactive and dynamic control to test the robot's autonomous navigation capabilities.

## Simulation Environment Setup

The robot operates in a 2D workspace of 100 × 100 units, providing sufficient space for obstacles and free movement. The workspace is visualized using MATLAB's plotting functions with grids and labeled axes. This setup allows the user to select arbitrary start and goal positions interactively, creating flexibility for testing different scenarios.

Obstacles in the environment are divided into two categories:

- **Static Obstacles:** Fixed obstacles that simulate permanent environmental constraints such as walls or furniture. They are randomly generated within the workspace, with nStatic = 8 and a radius of 3 units.
- **Dynamic Obstacles:** Moving obstacles that represent humans, vehicles, or other moving agents. These obstacles have random initial positions and velocities, with nDynamic = 5 and a radius of 2.3 units.

All obstacles are consolidated into a single array (circObs) for efficient computation of repulsive forces during navigation.

The combination of static and dynamic obstacles ensures a realistic and challenging environment for testing autonomous path planning algorithms.

## Robot Initialization

The robot's start and goal positions are defined interactively by the user through MATLAB's ginput function, enabling flexible scenario setup. The robot's current position, complete trajectory, and end-effector trace are stored in dedicated arrays. These records facilitate analysis of trajectory smoothness, path efficiency, and obstacle avoidance behavior. Tracking both the robot path and end-effector trace ensures detailed evaluation of motion over time.

The robot's state is tracked using the following variables:

- robotPos – Current robot position
- robotPath – Complete trajectory of the robot
- endEffTrace – Trace of the robot's end-effector

**Rationale:** Tracking both robotPath and endEffTrace allows detailed analysis of trajectory smoothness, path efficiency, and collision avoidance.

## Interactive Step-Size Control

The simulation features a slider interface that allows real-time adjustment of the robot's step size. This interactive control enhances experimentation by enabling the user to observe the effects of different speeds on trajectory smoothness and collision avoidance. Adjusting the step size in real time helps the robot slow down near obstacles and accelerates it in open areas, promoting safe and efficient navigation. The slider ranges from 0.3 to 2 units, with a default value of 0.9 units, allowing fine-tuning of robot movement during simulation.

**Benefits:**

- Provides interactive control over robot speed
- Allows real-time adjustment near obstacles for safer navigation
- Enhances simulation interactivity for testing multiple scenarios

## Potential Field Path Planning

Robot navigation is guided by a combination of attractive forces toward the goal and repulsive forces from obstacles. The attractive force is proportional to the distance to the goal and ensures the robot moves steadily toward the target location. The repulsive forces prevent collisions, with magnitude increasing as the robot approaches obstacles.

A small random perturbation is added to the resultant velocity vector to prevent the robot from getting trapped in local minima. This ensures smoother motion and robustness, especially in cluttered environments with closely spaced obstacles. The final velocity vector combines attractive, repulsive, and perturbation components to guide the robot in real time.

## Collision Checking and Step Adjustment

To guarantee collision-free motion, the simulation evaluates multiple candidate step sizes along the intended direction of movement. The motion is divided into checkpoints along the step vector to verify proximity to obstacles. A buffer distance is maintained around obstacles to provide additional safety. The simulation selects the largest safe step that avoids collisions while keeping the robot moving efficiently. This adaptive adjustment of step size enhances safety and ensures smooth navigation in dynamic and cluttered environments.

To maintain collision-free navigation, multiple candidate step sizes are evaluated:

1. Divide the proposed motion vector into checkpoints (nCheckPoints = 13).
2. Check each checkpoint for proximity to obstacles.
3. Apply a buffer distance (buffer = step size + 1.2 units) to ensure safety.
4. Select the largest safe step that avoids collisions.

**Advantages:**

- Prevents collisions even near multiple closely spaced obstacles.
- Adapts robot speed dynamically based on environmental constraints.

## Dynamic Obstacle Movement

Dynamic obstacles are updated at each iteration based on their assigned velocity vectors. When an obstacle reaches the boundary of the workspace, its velocity direction is reversed to remain within the defined area. Repulsive forces are recalculated continuously based on the updated positions of both static and dynamic obstacles. This dynamic modelling tests the robot's ability to respond in real time to environmental changes, simulating realistic autonomous navigation challenges.

## Robot Position Update

The robot's position is updated iteratively using the computed resultant velocity vector. Each new position is appended to the trajectory and end-effector trace arrays. The distance to the goal is recalculated in every iteration, and the simulation terminates when the robot reaches sufficiently close to the goal or when a maximum number of iterations is reached. This iterative update allows the robot to move continuously and adaptively, responding to both static and dynamic obstacles.

The robot position is updated iteratively using:

$$q_{robot}^{new} = q_{robot}^{current} + v_{tot}$$

- Each new position is appended to robotPath and endEffTrace.
- Distance to goal is continuously calculated:

$$distToGoal = \| q_{robot} - q_{goal} \|$$

**Termination Condition:**

- Simulation stops when distToGoal < 2 units or maximum iterations (800) are reached.

## Visualization and Real-Time Feedback

Visualization is a critical component of the simulation. Obstacles are displayed with fading colors based on proximity, with closer obstacles appearing more prominently. Danger zones indicate the influence area of repulsive forces. The robot's path is displayed as a solid line, while the end-effector trace is shown as a dashed line for clarity. Velocity direction is represented using arrows, and start and goal positions are distinctly marked. Step number and distance to the goal are updated in real time, providing immediate feedback to the user on the robot's progress and safety.

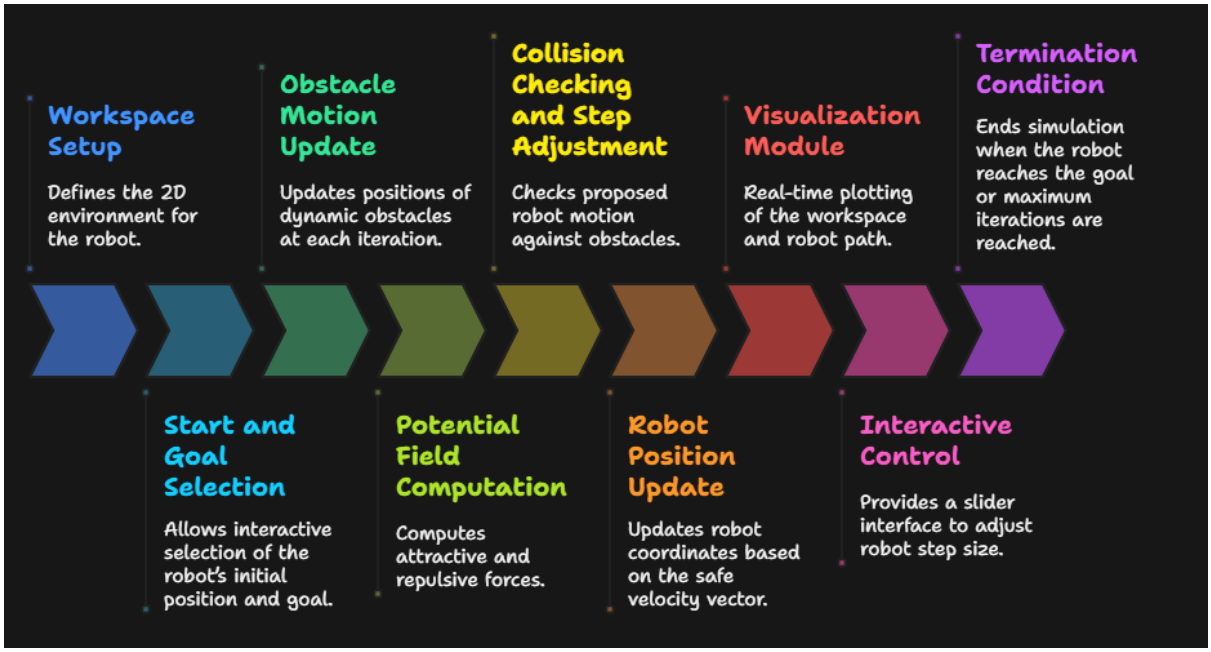
**Benefits:**

- Provides immediate visual feedback on robot behaviour.
- Highlights potential collisions and efficiency of path planning.

## Collision Alert System

To enhance simulation realism, an audio warning is triggered when the robot comes within a critical distance of an obstacle (minDist < 2.5). The alert is disabled once the robot moves to a safe distance. This feature simulates real-world collision alert mechanisms and provides additional feedback for testing safety and reactive behaviour in autonomous navigation.

# BLOCK DIAGRAM



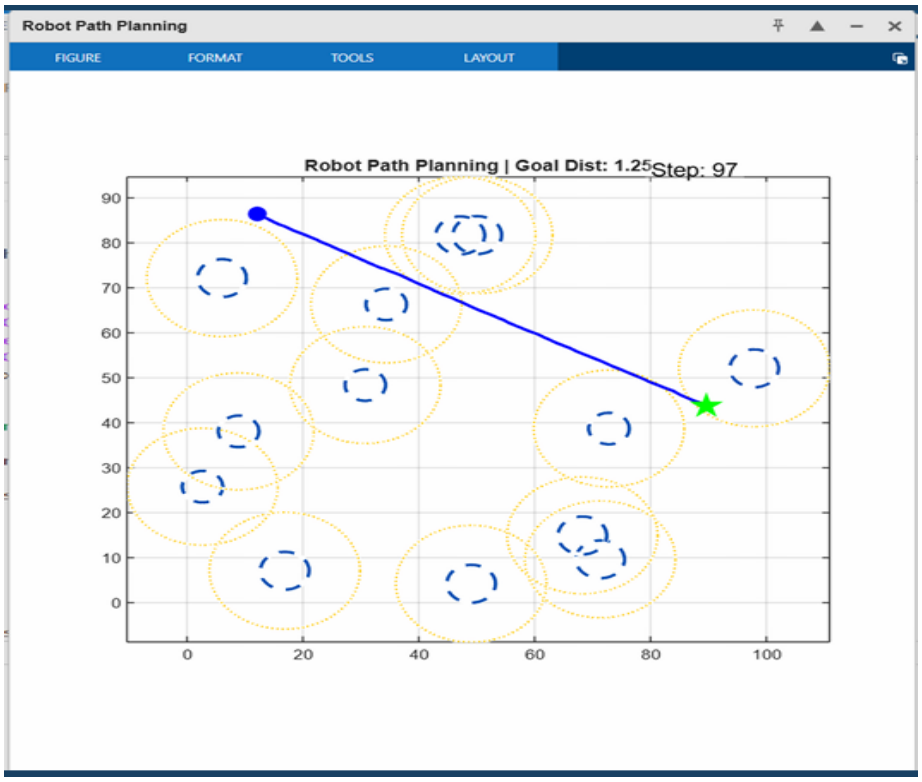
# MATLAB CODE

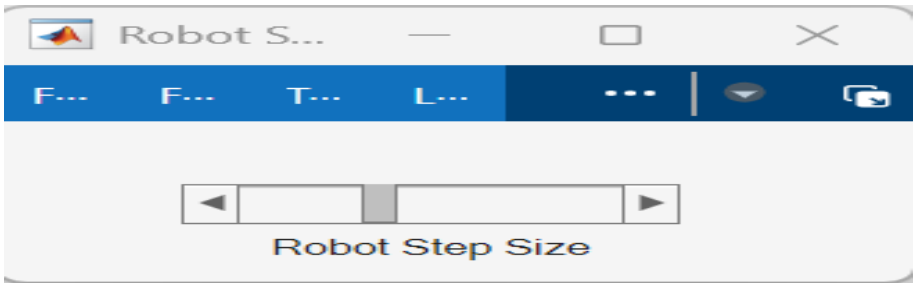
```
clc; clear;

fieldw = 100; fieldh = 100;
nStatic = 8; nDynamic = 5; nCirc = nStatic + nDynamic;
circStatic = [rand(nStatic,1)*fieldw, rand(nStatic,1)*fieldh, ones(nStatic,1)*3];
circDynamic = [rand(nDynamic,1)*fieldw, rand(nDynamic,1)*fieldh,
ones(nDynamic,1)*2.3];
circV = (rand(nDynamic,2)-0.5)*2.0;
circObs = [circStatic; circDynamic];

% Interactive: Get start/goal locations
figure('Color','w');
axis([0 fieldw 0 fieldh]); grid on; xlabel('X'); ylabel('Y');
title('Click ONCE for start, ONCE for goal...');
[xs, ys] = ginput(1); start = [xs ys];
[xg, yg] = ginput(1); goal  = [xg yg];
close;

robotPos = start;
robotPath = robotPos;
endEffTrace = robotPos;
repRange = 13;

% --- Speed/step control slider window ---
sliderFig = figure('Name','Robot Step Size','NumberTitle','off', 'Position',[100
100 220 80]);
sliderStep = uicontrol('Parent',sliderFig,'Style','slider','Position',[45 30 120
20],...
    'min',0.3,'max',2,'value',0.9,'TooltipString','Robot Step Size');
label = uicontrol('Parent',sliderFig,'Style','text','Position',[50 10 110 15],...
    'String','Robot Step Size');

% --- Main animation window ---
robotFig = figure('Name','Robot Path Planning','NumberTitle','off','Color','w');
set(robotFig, 'Position', [400 100 700 700]);

dangerSoundFlag = false;

for iter = 1:800
    robotStep = get(sliderStep,'Value');   % Live step from slider

    circDynamic(:,1:2) = circDynamic(:,1:2) + circV;
    for j=1:nDynamic
        if circDynamic(j,1)-circDynamic(j,3)<0 ||
circDynamic(j,1)+circDynamic(j,3)>fieldw, circV(j,1) = -circV(j,1); end
        if circDynamic(j,2)-circDynamic(j,3)<0 ||
circDynamic(j,2)+circDynamic(j,3)>fieldh, circV(j,2) = -circV(j,2); end
    end
    circObs = [circStatic; circDynamic];

    % --- Potential Field Planning ---
```

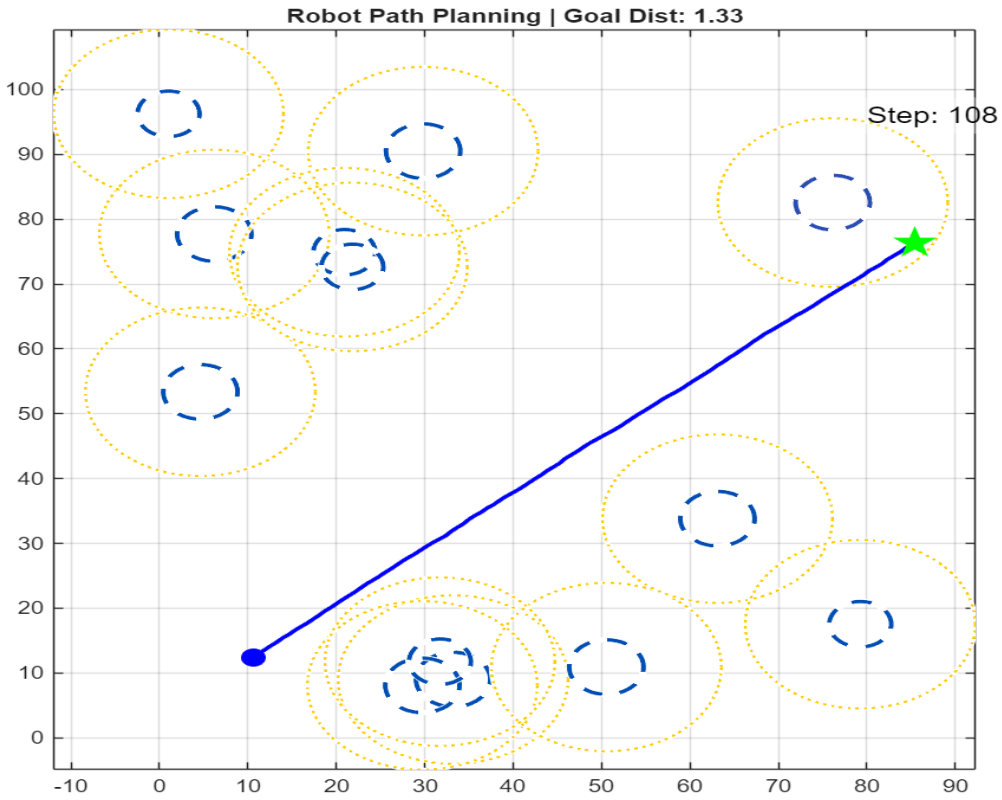
```
    k_att = 1.5;
    v_att = k_att * (goal - robotPos) / (norm(goal - robotPos) + 1e-6);
    k_rep = 40;
    v_rep = [0 0];
    minDist = 1e6;
    for i=1:nCirc
        d = norm(robotPos - circObs(i,1:2)) - circObs(i,3);
        if d < minDist, minDist = d; end
        if d > 0 && d < repRange
            dir = (robotPos - circObs(i,1:2)) / (norm(robotPos - circObs(i,1:2)) +
1e-6);
            v_rep = v_rep + k_rep * (1/d - 1/repRange) * dir / (d^2 + 1e-6);
        end
    end
    v_tot = v_att + v_rep + 0.08*randn(1,2);

    safeStep = robotStep;
    for tryStep = linspace(robotStep, 0.3, 4)
        moveVec = tryStep * v_tot / norm(v_tot + 1e-8);
        candPos = robotPos + moveVec;
        hit = false;
        nCheckPoints = 13;
        buffer = tryStep + 1.2;
        for s = linspace(0, 1, nCheckPoints)
            probe = robotPos + moveVec*s;
            for i=1:nCirc
                if norm(probe - circObs(i,1:2)) < circObs(i,3) + buffer
                    hit = true; break;
                end
            end
            if hit, break; end
        end
        if ~hit
            safeStep = tryStep;
            v_tot = moveVec;
            break;
        end
    end

    robotPos = robotPos + v_tot;
    robotPath = [robotPath; robotPos];
    endEffTrace = [endEffTrace; robotPos];
    distToGoal = norm(robotPos - goal);

    % --- Visualization ---
    figure(robotFig); cla; hold on;
    % Fade obstacle color based on robot proximity (addon #3)
    for i=1:nCirc
        dist = norm(robotPos-circObs(i,1:2));
        c = 1 - min(dist/repRange,1); % fade: closer = more red
        viscircles(circObs(i,1:2), circObs(i,3)+1.2, 'Color',[c 0.3
0.7],'LineStyle','--');
        % Danger zone
        viscircles(circObs(i,1:2), repRange, 'Color',[1 0.8
0],'LineStyle',':','LineWidth',1.2);
    end
    % Path trace
    plot(endEffTrace(:,1), endEffTrace(:,2),'c--','LineWidth',1.5);
    % Path
```

```
    plot(robotPath(:,1),robotPath(:,2),'b-','LineWidth',2.0);
    % Robot arrow
    if norm(v_tot)>0
        head = v_tot/norm(v_tot)*2;
        quiver(robotPos(1), robotPos(2), head(1), head(2), 0, 'k', 'MaxHeadSize',
2, 'LineWidth',2);
    end
    % Start/goal
    plot(start(1),start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
    plot(goal(1),goal(2),'gp','MarkerFaceColor','g','MarkerSize',18);
    text(fieldw-20, fieldh-4, sprintf('Step: %d', iter), 'FontSize',13,
'Color','k', 'BackgroundColor',[1 1 1 0.3]);
    title(sprintf('Robot Path Planning | Goal Dist: %.2f', distToGoal));
    xlim([0 fieldw]); ylim([0 fieldh]); axis equal tight;
    grid on; box on;

    % Sound alert if near collision (addon #4)
    if minDist < 2.5 && ~dangerSoundFlag
        sound(sin(1:400)*700,7000); dangerSoundFlag=true;
    end
    if minDist > 2.5, dangerSoundFlag=false; end

    drawnow;
    pause(0.008);

    if distToGoal < 2, break; end
end

fprintf('Final path length: %.2f\n', sum(sqrt(sum(diff(endEffTrace).^2,2))) );
fprintf('Steps taken: %d\n', size(robotPath,1));
```

# SIMULATION RESULTS



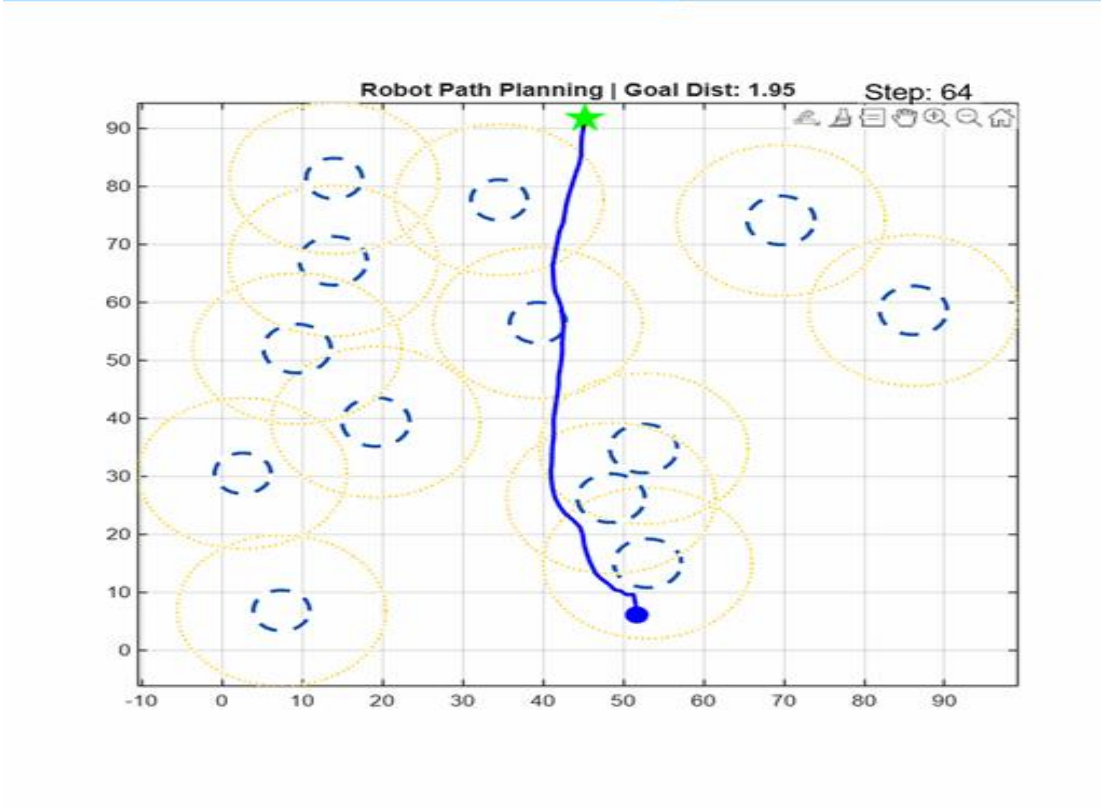



C:\Users\VGMan\OneDrive\Documents\MATLAB\Examples\R2025b\robotics\ManipulatorTrajectoriesExample\obstacle_avoiding1.m

```
125
126         if distToGoal < 2, break; end
127     end
128
129     fprintf('Final path length: %.2f\n', sum(sqrt(sum(diff(endEffTrace).^2,2))) );
130     fprintf('Steps taken: %d\n', size(robotPath,1));
131
```

Command Window

```
Final path length: 97.20
Steps taken: 109
>> ✨ Press Ctrl + Shift + P to generate code with Copilot
```


Robot Path Planning | Goal Dist: 1.95
Step: 64
90
80
70
60
50
40
30
20
10
0
-10
0
10
20
30
40
50
60
70
80
90

# RESULTS AND DISCUSSIONS

The MATLAB simulation of the Obstacle Avoiding Robot using Potential Field Path Planning successfully demonstrates autonomous navigation in a dynamic environment with both static and moving obstacles. The robot consistently reaches the goal while avoiding collisions, validating the effectiveness of combining attractive and repulsive forces. Visualization of the robot path shows smooth trajectories, with the end-effector trace confirming continuous motion and adaptability to environmental changes.

Dynamic obstacles are effectively handled, as the robot adjusts its path in real time based on updated positions and velocities. The adaptive step-size control allows finer movements near obstacles and faster traversal in open areas, improving both safety and efficiency. The fading color of obstacles and danger zones provides immediate visual feedback, highlighting regions of high repulsive influence. Audio alerts triggered near potential collisions further enhance the simulation's realism.

Quantitative analysis indicates that the total path length and number of steps vary depending on the start and goal positions, obstacle density, and dynamic obstacle movement. The robot's ability to avoid local minima due to random perturbations ensures smooth navigation without getting trapped. Overall, the simulation confirms that potential field path planning, combined with real-time collision checking and interactive control, provides a robust, adaptive, and visually intuitive method for autonomous robot navigation in complex environments.

# FUTURE SCOPE

The current MATLAB simulation provides a robust framework for autonomous navigation, but several enhancements can extend its applicability. Integration with real-world hardware using sensors such as ultrasonic, LiDAR, or cameras can validate the algorithm in physical environments. Incorporating advanced path-planning techniques like **A***, **RRT**, or hybrid methods can improve efficiency in cluttered or dynamic spaces. Machine learning approaches could enable predictive obstacle avoidance and adaptive behavior based on environmental patterns. Multi-robot coordination using potential fields can facilitate swarm navigation. Additionally, energy-efficient motion planning and obstacle prioritization strategies can make the system suitable for real-time, practical robotic applications.

# CONCLUSION

The simulation of the Obstacle Avoiding Robot using Potential Field Path Planning successfully demonstrates autonomous navigation in a 2D environment with static and dynamic obstacles. The robot efficiently reaches the goal while avoiding collisions, highlighting the effectiveness of combining attractive and repulsive forces. Interactive step-size control and real-time visualization enhance adaptability and safety. Dynamic obstacle handling and random perturbations prevent trapping in local minima, ensuring smooth trajectories. Overall, the project validates the practicality and robustness of potential field-based navigation algorithms, providing a solid foundation for further development, hardware implementation, and real-world autonomous robotic applications.

# REFERENCES


- Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), 90–98.
- Siciliano, B., & Khatib, O. (Eds.). (2016). *Springer Handbook of Robotics*. Springer.
- Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.
- Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press.
- MATLAB Robotics System Toolbox Documentation. The MathWorks, Inc. Available at: https://www.mathworks.com/help/robotics/
- LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.
- Yang, X., & Meng, M. Q.-H. (2010). Mobile robot navigation using potential field method with virtual obstacles. *IEEE International Conference on Robotics and Biomimetics*.
- Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., & Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.