****

**Project Stage 3: Final Evaluation**

**Project Title**:

Simulate a Mobile Robot in a Warehouse Using Gazebo

**Presented by:**

L Phanindra Naidu -AV.EN.U4CSE22023

S Akhil – AV.EN.U4CSE22140

V Hemanth Kumar -AV.EN.U4CSE22147

T Uday Kiran– AV.EN.U4CSE22245

**Date of Submission: [25-03-2025]**

**Abstract:**

**Background:**

*The demand for autonomous mobile robots (AMRs) in warehouses is increasing day by day. These AMRs play an important role in managing and transporting inventory in warehouses. Navigating in these environments required optimal path planning and obstacle avoidance techniques.*

**Research Gap:**

*A\* and Dijkstra are used in static environments for pathfinding. However, when facing an unexpected obstacle, a planned path may be blocked.*

*While SLAM is used in real-time mapping but struggles to integrate with the correct path*

**Objective:**

*The project develops a hybrid ROS system. by integrating both SLAM and path planning algorithms (A\* and Dijkstra) to create an optimal path and avoid obstacles along the way by recalculating the path to ensure safe and efficient warehouse navigation.*

**Method:**

*The occupancy grid map is used for implementing the A\* algorithm for primary path planning. If by any chance it failed, Dijkstra serves as a backup. SLAM (Simultaneous Localization and Mapping) is used to map unknown environments by robot position within the environment. Dynamic Window Approach (DWA) ensures obstacle avoidance. The simulation is carried out using MATLAB R2024b, ROS Noetic, and Gazebo 11 on Ubuntu 24.04 running in VirtualBox.*

**Key Findings:**

*Simulation shows that the A\* algorithm performs well in most scenarios. And if it ever fails, Dijkstra is there for fallback. SLAM is for improving environmental understanding for the robot. DWS is for obstacle avoidance, although it takes time to correct the path by repositioning the robot to ensure safety in the presence of obstacles.*

**Simulation Tool:**

*MATLAB R2024b with ROS Toolbox, Navigation Toolbox, and Gazebo 11 in Ubuntu 24.04.*

**Introduction:**

**Background:**

*The project is to develop an autonomous navigating system for a warehouse environment by ROS, MATLAB, and Gazebo. The way it works is occupancy grid mapping and path planning algorithms so that the robot can navigate in the environment simultaneously using sensors for real-time understanding of the environment while avoiding obstacles.*

**Problem Statement:**

*The problem is to develop a virtual environment where the mobile robot can be tested in navigation and behavior. The main aim is to create a controlled environment where mobile robots can interact with obstacles, detect surroundings, and achieve navigation tasks efficiently without wasting energy.*

**Objectives:**

1. *Gazebo and MATLAB are used to create a mobile robot in a 3D virtual environment.*
2. *The mobile robot moves and avoids obstacles in a simulated work space.*
3. *Implemented the A\* algorithm for primary pathfinding; if it fails, it uses Dijkstra as a backup and SLAM for real-time observation.*
4. *MATLAB and Gazebo are integrated to control and analyze the robot simulation result.*
5. *Dynamic window approach for obstacle avoidance.*

**Research Questions:**

1. *Can A\* and Dijkstra's algorithm effectively navigate the environment?*
2. *Can the dynamic repathing detect obstacles?*

**Literature Review:**

**Overview of past studies:**

*Algorithms like A\* and Dijkstra are best used for navigating an autonomous system, but it didn't perform well in a dynamic environment. SLAM techniques are used for environmental understanding, but they need map updates regularly. Dynamic Windows approach and vector field histogram or obstacle used avoidance techniques, but they are slow in pathfinding****.***

**Research Gap:**

1. *Limited research on fallback algorithms such as in this project, the Dijkstra algorithm with real-time LIDAR sensors to correct robot actions*
2. *Combining path planning with the SLAM dynamic window approach is an under-explored area.*

**Motivation:**

*To develop a hybrid system with path planning and real-time observation SLAM and Dynamic Window Approach for obstacle avoidance to improve the navigation.*

**Contribution:**

*This project introduces a comprehensive ROS-based navigation system that seamlessly switches between A\* and Dijkstra algorithms while dynamically correcting paths based on SLAM data and obstacle detection.*

**Methodology:**

**Description of the robotic system :**

1. ***Platform****: TurtleBot3 Burger model running on Ubuntu 24.04 (VirtualBox).*
2. ***ROS Version****: ROS Noetic for communication and control.*
3. ***Simulators****: MATLAB R2024b and Gazebo 11.*
4. ***Sensors****: Simulated LiDAR for obstacle detection and environment perception.*

**A\* Algorithm:**

*1. Initialize open list with start node [row, col, g=0, f=h], closed list empty.*

*2. Select node with minimum f = g + h.*

*Explore neighbors (up, down, left, right) if free and unvisited.*

*4. Update g, h, f, and parent if a shorter path is found.*

*5. Repeat until goal reached or open list exhausted.*

**Pseudocode:**

**PROGRAM BidirectionalWarehouseNavigation**

*// Constants*

*CONST:*

*GOAL\_TOLERANCE = 0.05*

*INFLATION\_RADIUS = 0.25*

*SAFE\_CLEARANCE = 0.15*

*MAX\_REPLANS = 5*

*CONTROL\_RATE = 15Hz*

*EMERGENCY\_DIST = 0.025*

*SLOW\_DOWN\_DIST = 0.75*

*FUNCTION main():*

*// 1. Initialization Phase*

*ROS\_INIT('http://10.21.195.237:11311')*

*map = GET\_OCCUPANCY\_MAP('/map')*

*INFLATE\_MAP(map, INFLATION\_RADIUS)*

*// 2. Setup Start and Goal Positions*

*fixed\_start = FIND\_VALID\_START(map, SAFE\_CLEARANCE)*

*original\_goal = FIND\_VALID\_GOAL(map, fixed\_start, SAFE\_CLEARANCE)*

*// 3. Path Planning*

*path\_to\_goal = PLAN\_PATH\_ASTAR(map, fixed\_start, original\_goal)*

*IF path\_to\_goal EMPTY:*

*path\_to\_goal = PLAN\_PATH\_DIJKSTRA(map, fixed\_start, original\_goal)*

*path\_to\_start = PLAN\_PATH\_ASTAR(map, original\_goal, fixed\_start)*

*IF path\_to\_start EMPTY:*

*path\_to\_start = REVERSE\_PATH(path\_to\_goal)*

*// 4. Robot Initialization*

*initial\_yaw = CALC\_INITIAL\_YAW(path\_to\_goal)*

*TELEPORT\_ROBOT(fixed\_start, initial\_yaw)*

*// 5. Main Navigation Loop*

*current\_phase = 'toGoal'*

*recovery\_state = 'normal'*

*replan\_count = 0*

*waypoint\_index = 1*

*WHILE (NOT goal\_reached OR NOT start\_reached):*

*// Perception Update*

*robot\_pose = GET\_ROBOT\_POSE()*

*scan\_data = GET\_LIDAR\_SCAN()*

*// Phase Management*

*IF current\_phase == 'toGoal':*

*target = original\_goal*

*current\_path = path\_to\_goal*

*ELSE:*

*target = fixed\_start*

*current\_path = path\_to\_start*

*// Waypoint Tracking*

*[waypoint\_index, path\_error] = UPDATE\_WAYPOINT(robot\_pose, current\_path, waypoint\_index)*

*// Goal Checking*

*IF DISTANCE(robot\_pose, target) <= GOAL\_TOLERANCE:*

*HANDLE\_GOAL\_REACHED()*

*CONTINUE*

*// Obstacle Detection*

*[emergency\_stop, obstacle\_detected, obs\_dist, obs\_angle] =*

*ANALYZE\_SCAN(scan\_data, EMERGENCY\_DIST, SLOW\_DOWN\_DIST)*

*// State Machine*

*SWITCH recovery\_state:*

*CASE 'normal':*

*[v, omega] = CALC\_CONTROL(robot\_pose, current\_path, waypoint\_index, current\_phase)*

*IF emergency\_stop:*

*recovery\_state = 'avoiding'*

*avoidance\_start = NOW()*

*STOP\_ROBOT()*

*ELIF obstacle\_detected AND replan\_count > 0:*

*recovery\_state = 'avoiding'*

*avoidance\_start = NOW()*

*STOP\_ROBOT()*

*ELSE:*

*[v, omega] = ADJUST\_FOR\_OBSTACLE(v, omega, obs\_dist, obs\_angle)*

*SEND\_VELOCITY(v, omega)*

*CASE 'avoiding':*

*[v, omega] = AVOID\_OBSTACLE(obs\_angle, obs\_dist)*

*SEND\_VELOCITY(v, omega)*

*IF NOT emergency\_stop AND NOT obstacle\_detected:*

*recovery\_state = 'replanning'*

*replan\_start = NOW()*

*STOP\_ROBOT()*

*CASE 'replanning':*

*IF (NOW() - replan\_start) > 1s:*

*new\_path = REPLAN\_PATH(robot\_pose, target, map)*

*IF new\_path VALID:*

*IF current\_phase == 'toGoal':*

*path\_to\_goal = new\_path*

*ELSE:*

*path\_to\_start = new\_path*

*waypoint\_index = 1*

*replan\_count += 1*

*recovery\_state = 'normal'*

*ROTATE\_TO\_PATH(new\_path)*

*// Visualization*

*UPDATE\_NAV\_DISPLAY(map, robot\_pose, current\_path, waypoint\_index, recovery\_state)*

*// Safety Check*

*IF replan\_count >= MAX\_REPLANS:*

*ABORT("Maximum replan attempts reached")*

*SLEEP(1/CONTROL\_RATE)*

*// Cleanup*

*STOP\_ROBOT()*

*SHOW\_NAVIGATION\_STATS()*

*FUNCTION HANDLE\_GOAL\_REACHED():*

*IF current\_phase == 'toGoal':*

*goal\_reached = TRUE*

*ROTATE\_TO\_PATH(path\_to\_start)*

*current\_phase = 'toStart'*

*waypoint\_index = 1*

*ELSE:*

*start\_reached = TRUE*

*FUNCTION CALC\_CONTROL(pose, path, waypoint\_idx, phase):*

*// PID-based path following*

*target\_point = path[waypoint\_idx]*

*dx = target\_point.x - pose.x*

*dy = target\_point.y - pose.y*

*target\_angle = atan2(dy, dx)*

*heading\_error = NORMALIZE\_ANGLE(target\_angle - pose.yaw)*

*// Cross-track error calculation*

*IF waypoint\_idx > 1:*

*path\_vec = path[waypoint\_idx] - path[waypoint\_idx-1]*

*robot\_vec = [pose.x, pose.y] - path[waypoint\_idx-1]*

*cross\_track = CROSS\_PRODUCT(robot\_vec, path\_vec) / path\_vec.length()*

*ELSE:*

*cross\_track = 0*

*// PID Controller*

*p\_gain = 0.8*

*i\_gain = 0.01*

*d\_gain = 0.05*

*integral\_error += heading\_error*

*derivative\_error = heading\_error - last\_error*

*omega = p\_gain\*heading\_error + i\_gain\*integral\_error + d\_gain\*derivative\_error*

*// Speed control*

*IF phase == 'toGoal':*

*base\_speed = 0.3*

*ELSE:*

*base\_speed = 0.25*

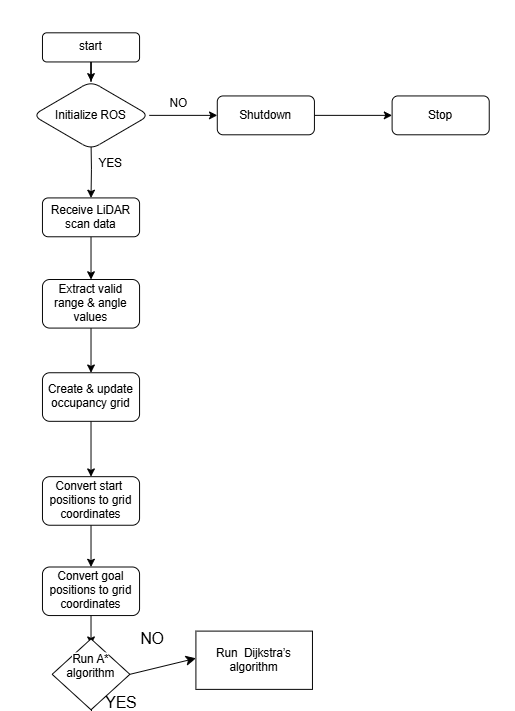
*speed\_factor = cos(heading\_error)^2*

*distance\_factor = min(1, DISTANCE(pose, target\_point)/0.5)*

*v = base\_speed \* speed\_factor \* distance\_factor*

*RETURN [CLAMP(v, 0.05, 0.4), CLAMP(omega, -1.5, 1.5)]*

**Flowchart:**

****

****

**Explanation of Flowchart:**

1*. Start*

* *Initialize the system.*

*2. Check if the simulator is active & initialize ROS*

* *Ensure the simulation environment (Gazebo) is running.*
* *Start ROS and necessary components.*

*3. Acquire LIDAR Data*

* *Collect sensor data for environment mapping.*

*4. Process LIDAR Data into Occupancy Grid*

* *Convert raw sensor readings into a grid map.*
* *Identify obstacles and free spaces.*

*5. Check if a Goal is Set*

* *Yes → Proceed to path planning.*
* *No → Wait for user input.*

*6. Path Planning (A or Dijkstra)\**

* *Compute the optimal path from the robot’s position to the goal.*

*7. Send Movement Commands*

* *Publish velocity commands for the robot to follow the planned path.*

*8. Navigate to the Goal*

* *Move the robot while continuously monitoring the path.*

*9. Obstacle Detection & Dynamic Replanning*

* *Check for unexpected obstacles in real-time.*
* *If an obstacle is detected, recalculate the path.*

*10. Check if the Goal is Reached*

* *Yes → Stop the robot.*
* *No → Continue moving and monitoring.*

*11. End*

* *The robot successfully reaches the destination and stops.*

**Research design :**

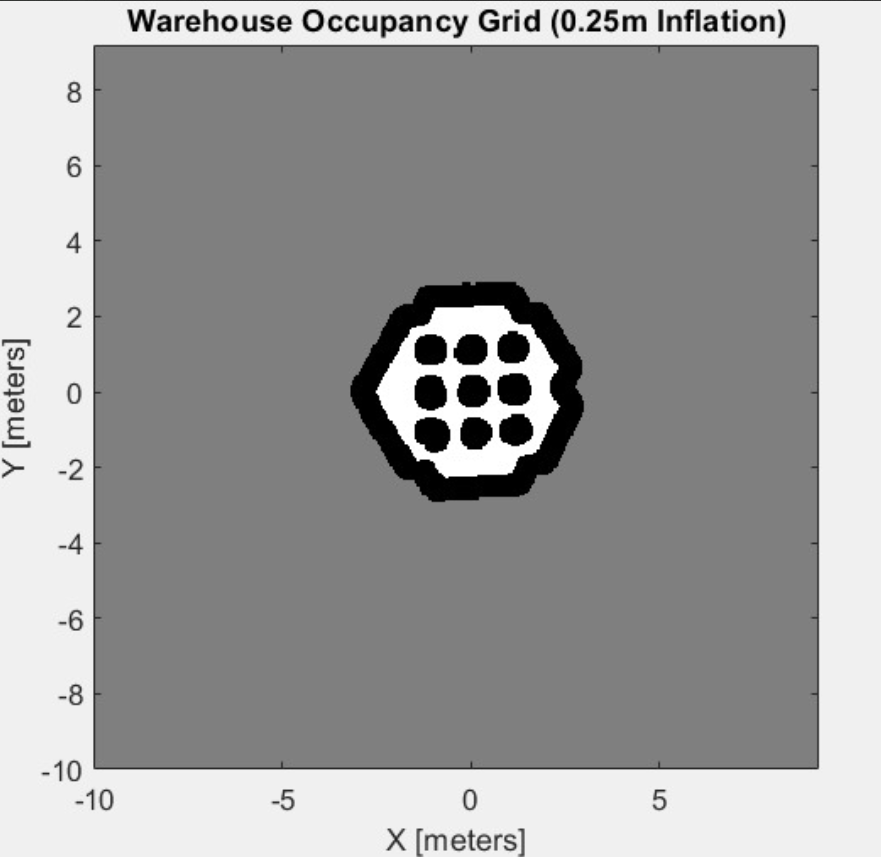
*Quantitative Analysis: Evaluate algorithm efficiency, SLAM accuracy, and obstacle handling effectiveness.*

**Result and Discussion:**

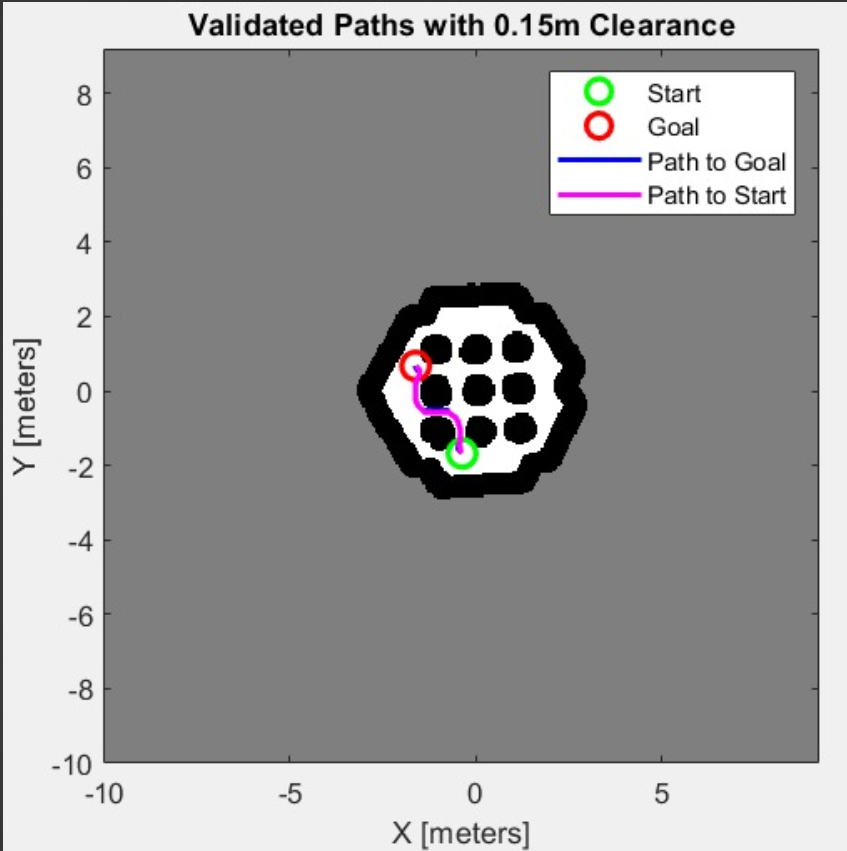
**Tools:**

*MATLAB R2024b, ROS Noetic, and Gazebo 11 on Ubuntu 24.04 running in VirtualBox.*

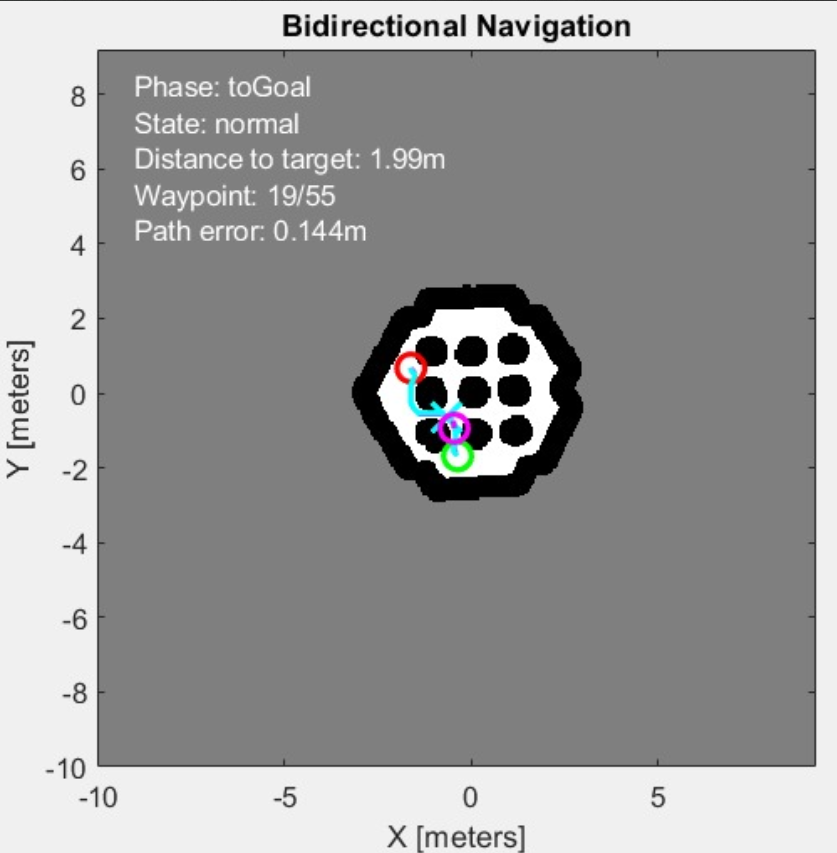
**Simulation Results:**

****

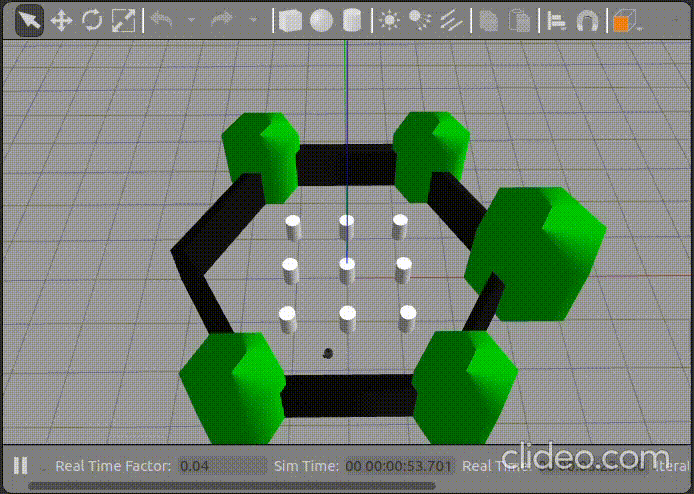
*This figure is the grid of the virtual environment. SLAM is used to scan the environment to get its grid; it is the first file to load into the code. It helps to find the path and navigate through obstacles effectively, ensuring that the system can operate efficiently in real-time scenarios.*

**

*In this image, the path that the robot follows to navigate through the environment is visible.*

**

*In this image it shows how the robot is following the path planned in the grid; it helps us to know where the robot is moving, its start and end points, distance, waypoints, and path error.*

**

*It is the final simulation video taken from Gazebo. gives us a clear 3D environment . The robot follows the same path from the grid above.*

**Conclusion and Recommendations:**

*The proposed system effectively integrates A\* and Dijkstra algorithms for efficient path planning, with Dijkstra serving as a fallback when A\* fails to find a feasible route. This adaptability ensures accurate localization in dynamic environments. The Dynamic Window Approach (DWA) enables robots to efficiently detect and avoid obstacles, which is crucial in unpredictable storage settings. Additionally, by integrating path planning, SLAM, and obstacle avoidance, the system enhances overall navigation efficiency.*

*To further optimize performance, AI-based models can be incorporated to predict obstacle movement, improving navigation accuracy. Expanding the system to support multi-robot coordination will enhance efficiency in large-scale operations by enabling collaboration and collision avoidance. Implementing the system on real-world platforms like TurtleBot3 will help validate simulation results and assess real-time performance. Furthermore, cloud-based control can be leveraged for real-time map updates and robot coordination, allowing seamless data sharing and improved operational efficiency. These enhancements will make the autonomous navigation system more adaptable to real-world applications*

**Contribution:**

*Developed a dynamic ROS-based navigation system with real-time path replanning, map updates, and robust obstacle handling*.

**Limitations and suggestions for future research:**

1. *The LIDAR sensor sector limits obstacle recognition in distant environments.*
2. *SLAM requires periodic recalibration to maintain accuracy in large environments.*

**References:**

MATLAB:- <https://in.mathworks.com/help/robotics/ug/simulate-a-mobile-robot-using-gazebo.html>

Gazebo:- <https://classic.gazebosim.org/tutorials>

Set Up Gazebo Simulation Environment:-

<https://in.mathworks.com/help/rtw/vexv5/ug/setup-gazebo.html>