  
UNIVERSITY OF HERTFORDSHIRE  
School of Physics, Engineering and Computer Science

MSc Artificial Intelligence and Robotics  
7COM1039-0509-2024 - Advanced Computer Science Masters Project  
July 14, 2025

LLM-Guided Multi-Agent-Robot Navigation Using A\* and Sensor-Based Control for Obstacle Avoidance in CoppeliaSim

Name: Mohamad Dirani  
Student ID: 23074344  
Supervisor: Grigorios Skaltsas

# Introduction and Overview

## Background and Motivation

Multi-robot autonomous systems are essential in modern industrial and service applications where tasks need to be distributed and executed efficiently. Key to the success of these systems is reliable navigation—finding and following safe paths to goals while avoiding collisions. Classical path planning algorithms such as A\* are widely used for global navigation (Candra et al., 2020), while real-time reactive methods handle local obstacle avoidance (Li et al., 2017).

Recent advances in Large Language Models (LLMs), such as OpenAI's GPT-4, offer new opportunities for human-robot interaction. These models can understand natural language instructions and convert them into structured commands, providing a flexible interface for dynamic goal assignment (OpenAI, 2023). This project leverages LLMs to introduce a human-in-the-loop control mechanism, integrating speech-based input with real-time autonomous planning and avoidance systems.

## Research Question

* Main Research Question:

How do dynamic obstacles and environmental complexity affect the robots' ability to navigate and complete tasks using the implemented avoidance logic? (Gourley and Trivedi, 1994)

* Sub-Research Questions:

1. How does coordination between multiple agents influence the overall task efficiency and response time in the system? (Längle and Wörn, 2001; Parisotto et al., 2017)
2. Does human intervention using LLM-based commands for one robot impact the task performance or progress of other robots operating simultaneously in a shared environment? (Längle and Wörn, 2001)
3. Does the robot perform differently when encountering static obstacles, dynamic obstacles, or other robots as obstructions? (Gourley and Trivedi, 1994)

## Literature Review

This literature review examines eight scholarly works central to LLM-driven, multi-agent robot navigation and control. Each article is contextualized, critiqued, and compared against the goals and implementation of the current project. The review addresses how each contributes to the four main research questions.

* Gourley and Trivedi (1994) – Sensor-Based Obstacle Avoidance and Mapping:

This early work uses ultrasonic sensors and reactive force fields for fast navigation. It supports sub-question 3 by showing the critical role of local perception in high-speed environments. My system draws from this principle by using directional vision sensors and recursive sidestepping to compare performance across static and dynamic obstacles, including robots.

* Candra et al. (2020) – Path Planning with A\*:

This comparative study of A\* and Dijkstra validates A\*’s advantage in planning speed and efficiency. Their results support my choice of A\* for goal assignment. I compare completion times and rerouting across agents, contributing directly to sub-question 1 on coordination and efficiency.

* Parisotto et al. (2017) – Neural SLAM:

Parisotto et al. propose a memory-augmented agent framework that excels at collaborative exploration. Though my system does not include learned SLAM, the scalability tests I conduct on multiple robots simulate their conditions, which helps evaluate sub-question 1 concerning scalability and inter-agent coordination.

* Längle and Wörn (2001) – Human–Robot Cooperation in Multi-Agent Systems:

This article introduces structured role negotiation between humans and robots. In my system, I issue commands mid-task to specific robots using LLMs. Observing how other robots react (e.g., pause, wait, reroute) directly informs sub-question 2 regarding interference from human intervention.

* Mu et al. (2023) – ChatGPT with Knowledge Graphs:

Mu et al. demonstrate improved task parsing with LLM-KG integration. While my project omits the KG, I evaluate GPT-4’s capacity to produce actionable JSON commands from speech. These findings directly inform coordination strategies explored in sub-question 1 and informs the system’s scalability.

* OpenAI (2023) – GPT-4 for Robotic Task Planning:

OpenAI’s work highlights GPT-4’s strength in transforming natural language into robotics plans. My project builds on this by evaluating GPT-4’s real-time responsiveness and task accuracy in simulation. This informs sub-question 2.

* Farley et al. (2022) – Comparing CoppeliaSim, Gazebo, MORSE, Webots:

This paper provides a detailed comparison of robot simulators. Their findings favor CoppeliaSim for accuracy and versatility, justifying my selection. It indirectly supports sub-question 1, as the simulator's precision and responsiveness affect coordination outcomes.

* KUKA Platform and Mecanum Design – Real-World Mecanum Wheel Dynamics (2020):

This paper provides a mathematical and simulation-based analysis of Mecanum wheels using the KUKA youBot as an example (Rus. J. Nonlin. Dyn., 2020). It accounts for real-world friction, wheel slip, and kinematic constraints in omnidirectional platforms. My simulation reflects these principles by translating theoretical wheel behavior into Python-based velocity mapping for precise robot control. This reinforces sub-question 3 by demonstrating how correct kinematics affect navigation reliability around static and dynamic obstacles.

This review supports the use of A\*, vision-based avoidance, and LLM-driven control in coordinated multi-robot systems. It also validates my choice of CoppeliaSim as a simulation platform. The reviewed works guide how I interpret obstacle types, robot interference, and communication timing across sub-tasks in my system.

## Methodology

This project implements a simulated multi-robot navigation system that leverages classical planning, real-time obstacle avoidance, and human-in-the-loop goal assignment via Large Language Models (LLMs). The simulation is developed in CoppeliaSim, chosen for its high-fidelity physics, support for omnidirectional robots, and native ZeroMQ communication support. Compared to other simulators like Gazebo and Webots, Farley et al. (2022) found CoppeliaSim to be the most versatile and accurate, especially for testing kinematics-based behavior in cluttered environments.

The environment is constructed as a 2D occupancy grid populated with static and dynamic obstacles. Robots are equipped with SICK S300 directional sensors on all sides (front, back, left, right) to enable sensor-based obstacle avoidance, a technique inspired by Gourley and Trivedi (1994), who demonstrated the effectiveness of local sensor feedback in reactive path correction.

**Robot Platform: KUKA youBot with Mecanum Wheels**

Each robot is modeled after the KUKA youBot, which features Mecanum wheels for full omnidirectional motion. This configuration is selected for its flexibility in navigating confined environments, allowing movement in any planar direction without rotating the chassis. KUKA (2020) provides a thorough simulation and mathematical treatment of real-world mecanum dynamics, confirming the practical advantages in systems requiring side-stepping and tight coordination.

**Navigation: A\* Algorithm**

For global path planning, the system employs the A\* algorithm due to its efficiency in grid-based environments. According to Candra et al. (2020), A\* significantly outperforms Dijkstra in computational speed while maintaining optimality. The path planner constructs a dynamic occupancy grid that updates based on obstacle presence, ensuring real-time path recalculation when needed.

**Goal Assignment: LLM Integration**

Robot goal positions are assigned dynamically through GPT-4, integrated via the OpenAI API. Users provide natural language commands—either spoken or typed—which the LLM parses into structured JSON goals. This method reflects the approach in OpenAI (2023), demonstrating GPT-4's strength in converting natural instructions into robotic plans. Mu et al. (2023) extend this further by integrating knowledge graphs, though this project focuses purely on LLM parsing.

**Multi-Robot Coordination**

The simulation includes at least two omnidirectional robots, expanding to more in future iterations. When a single goal is issued, the nearest robot is selected based on Euclidean distance. When multiple goals are presented, robots divide the tasks. Coordination is further enhanced by path-sharing logic: if a robot blocks another's path, it temporarily yields and clears the way. This dynamic cooperation strategy is inspired by Längleand Wörn (2001), who explored agent-based negotiation in multi-robot systems.

**System Behavior Scenarios**

The methodology supports several interaction patterns, which are tested and analyzed:

1. Static navigation with no human intervention
2. Dynamic goal reassignment mid-task (e.g., LLM command to change the goal)
3. Emergency stops issued by humans (via LLM) during path execution
4. Return-to-base behavior on demand
5. Blocked-path resolution via inter-robot coordination

These scenarios directly address the research aims, especially assessing how robots handle:

* Obstacle-rich environments
* Sudden human intervention
* Coordination across shared paths

## Project Artefact

The artefact is a modular software system developed in Python and integrated with CoppeliaSim via ZeroMQ. It simulates a team of omnidirectional robots equipped with front, back, left, and right sensors. Each robot:

* Receives human commands parsed by GPT-4 via OpenAI’s API.
* Uses an A\* algorithm to plan a path to the assigned goal.
* Applies sensor-based logic to avoid obstacles.

Modules developed include:

* main.py: simulation loop and orchestrator
* robot\_controller.py: control logic
* obstacle\_awareness.py: local avoidance
* astar\_env.py and map\_builder.py: path planning environment
* LLM.py: speech recognition and LLM integration

## Tools and Techniques

* Simulation: CoppeliaSim
* Programming: Python
* Control Architecture: A\* planner + LiDAR-based reactive logic
* LLM Interaction: OpenAI GPT-4 + speech\_recognition module (OpenAI, 2023)
* Communication: ZeroMQ

## Deliverables

* Fully functional multi-robot simulation
* Modular codebase
* LLM command interpreter
* Evaluation results (navigation success rate, command parsing accuracy, etc.)
* Final report and demonstration

## Ethical, Legal, Professional, and Social Issues

In the development and execution of this project, several ethical, legal, professional, and social considerations have been taken into account to ensure responsible research practice and adherence to academic and technological standards.

**Ethics Approval**  
Ethical approval for this project was deemed unnecessary, as no personal or identifiable user data is collected, stored, or processed. All human-robot interactions are simulated, and user commands are either developer-generated or provided through synthetic voice or text input for testing purposes. The project does not involve human participants or sensitive data, which excludes it from institutional ethics review under current university guidelines.

**Data Usage and Privacy**  
All data used within the system—such as goal coordinates, speech prompts, and sensor readings—are generated within the simulation environment. There is no connection to external databases, and no real-time data from external users is stored. The LLM integration (e.g., with GPT-4 via OpenAI’s API) is restricted to developer-generated inputs used for testing command parsing. These inputs are not linked to real individuals, ensuring compliance with data protection and privacy principles such as those outlined in GDPR.

**Professional Conduct**  
The integration of OpenAI’s GPT-4 model is conducted under full compliance with OpenAI’s terms of service. The system design ensures that no misuse of the API occurs (e.g., no spam requests, no prohibited queries). Furthermore, all third-party libraries, frameworks, and APIs used in the project (e.g., ZeroMQ, speech\_recognition, CoppeliaSim) are open source or properly licensed for academic use, and attribution is provided where appropriate. Version control and documentation practices are followed throughout the development to maintain transparency, reproducibility, and accountability.

**Social Implications**  
This project contributes positively to the field of human-robot interaction by demonstrating how natural language interfaces—powered by Large Language Models—can make robotic systems more accessible to non-expert users. By enabling intuitive command input through speech or text, this design significantly enhances accessibility for users without formal robotics training. This supports broader goals in inclusive technology and democratized automation, where robotic systems are made usable in educational, industrial, and assistive settings with minimal learning curves.

Additionally, the project's emphasis on collaborative robot behavior and dynamic human intervention aligns with ethical AI development principles, such as transparency, responsiveness, and user-centered design. These considerations reflect a responsible approach to robotics research, ensuring that technical innovation is matched by a commitment to ethical and professional standards.

# Progress to Date

## Work Done

**Designed and Configured a CoppeliaSim Environment for Multi-Robot Navigation**  
Developed a simulation world with grid-based layout, incorporating static walls and movable obstacles to emulate real-world clutter. This setup facilitates evaluation under varying environmental complexity as posed in the main research question.

Implemented A Path Planning with Real-Time Occupancy Grid Updates  
Developed a Python-based A\* algorithm that recalculates paths based on live occupancy grids. These grids are dynamically updated using feedback from the robot's local sensors, ensuring that navigation remains responsive to new or moving obstacles.

**Integrated Directional SICK S300 Sensors for Obstacle Detection and Alignment**  
Placed and calibrated virtual SICK S300 sensors in four directions (front, back, left, right) per robot. Implemented logic to interpret signals and determine whether to sidestep, realign, or reroute. This follows principles from reactive avoidance literature, enabling precise local navigation.

**Developed Recursive Decision-Making for Navigating Blocked Paths**  
To handle blockage scenarios more effectively, a state-driven control loop was implemented, testing lateral movement, and rechecking paths—allowing recovery from deadlocks. This decision-making loop is essential for performance in dense or dynamic environments.

**Integrated GPT-4 via OpenAI API for Natural Language Goal Assignment**  
Connected OpenAI’s GPT-4 model with the simulation using speech recognition and JSON translation. This allows users to set or change goals during execution using natural language, offering a human-in-the-loop control mechanism.

**Calibrated Robot Motion Dynamics: Speed and Directional Control**  
Tuned wheel velocities for accurate mecanum-based motion, ensuring forward, lateral, and diagonal translations correspond to intended commands. This calibration was essential for reliable execution of both planned paths and avoidance maneuvers.

**Implemented Nearest-Robot Task Allocation Strategy**  
Added logic to compute Euclidean distances between all robots and the user-defined goal, assigning the task to the nearest free agent. This supports sub-question 1 on task efficiency and cooperative performance.

**Enabled Cooperative Path Management Between Robots**  
Implemented coordination logic where a robot blocking another’s path detects the interference and moves aside. This feature promotes collaborative task execution in shared spaces and prevents deadlocks caused by static inter-agent positioning.

## Problems Encountered

**Incorrect Motion Due to Mecanum Wheel Logic**  
**Problem**: Initially, the robot would rotate or jitter instead of translating in the intended direction (e.g., forward motion caused diagonal drift). This was due to an incorrect velocity mapping for the Mecanum wheels, which require specific combinations of clockwise and counterclockwise rotations across the four wheels.  
**Solution**: The issue was resolved by revisiting the Mecanum kinematics equations and correctly implementing the velocity vector to wheel speed mapping. Resources from the CoppeliaSim Forum and the KUKA youBot control logic were instrumental in debugging this behavior.

**Repetitive Entry into Failed Movement Loops**  
**Problem**: The robot would get stuck in loops when a goal was unreachable due to persistent obstacles, repeatedly trying the same path without escape logic.  
**Solution**: Introduced a fallback mechanism that detects repeated failures (e.g., lack of movement after several ticks) and triggers alternate behaviors like sidestepping or realignment. Added termination conditions to avoid infinite loops. This solution was influenced by concepts in finite state machines and planning recovery from CoppeliaSim use cases and forums.

**Speech Recognition Errors**  
**Problem**: Speech-to-text conversion failed under unclear or noisy input, leading to unusable commands or missed triggers.  
**Solution**: To mitigate this, a simple retry mechanism was added. If the parsed command from speech was empty or nonsensical, the system would re-prompt the user or fall back to a default command. The speech\_recognition Python library’s documentation helped in customizing error handling: [SpeechRecognition Library](https://pypi.org/project/SpeechRecognition/).

**Initial Difficulties Connecting Python to CoppeliaSim via ZeroMQ**  
**Problem**: Connection between the Python scripts and CoppeliaSim using ZeroMQ failed intermittently, especially during early development.  
**Solution**: Solved by ensuring the correct version of the zmqRemoteApi and sim.py files were used. Also had to confirm Python was running in an environment compatible with CoppeliaSim’s version (V4.4+). The timeout settings were adjusted, and the socket server was properly initialized. The official CoppeliaSim Remote API documentation was essential.

**Adjusting Sensor Angles for Accurate Forward, Side, and Rear Detection**  
**Problem**: Sensors initially failed to detect nearby objects in their respective directions because they were either misaligned or had narrow detection cones.  
**Solution**: Adjusted the sensor orientation manually in the CoppeliaSim scene—setting correct rotations in the object properties. This fine-tuning ensured that front-facing sensors detected forward obstacles and side sensors covered lateral fields. The process was aided by visualization in the CoppeliaSim GUI and documentation on proximity sensor behavior.

**Inconsistent Sensor Detection Results**  
**Problem**: Some sensors would inconsistently detect obstacles even when objects were clearly within range, leading to unreliable avoidance behavior.  
**Solution**: Calibrated the sensor detection thresholds and max detection distances. Additionally, implemented signal smoothing and noise filtering logic in Python to average multiple frames of sensor data. These adjustments were inspired by user discussions on the CoppeliaSim Q&A forum.

**Complexity in Implementing Avoidance Logic**  
**Problem**: Real-time avoidance decisions conflicted with planned A\* paths, especially when needing to switch axes or sidestep persistently blocked routes.  
**Solution**: Developed a recursive decision-making system that evaluates the robot's current direction and attempts sidesteps or backtracking if the path is blocked. The coordination between real-time sensor input and A\*-based planning was handled using flags and prioritized decision layers. Design inspiration was taken from layered control architectures in reactive robotics.

# Planned Work

## Remaining Tasks

In the next phase of development, the system will incorporate **inter-robot clearance behavior** to enhance coordination. Specifically, when multiple goals are assigned, there are scenarios in which one robot may unintentionally obstruct the path of another. A mechanism will be implemented to detect such situations in real time. The obstructing robot will temporarily move aside, allowing the other to pass before resuming its own task. This behavior will be governed by shared path awareness and priority-based decision logic, further improving the system’s cooperative navigation capabilities.

In parallel, the simulation will be **scaled up to support more than two robots**. This expansion is intended to assess how system performance evolves with increasing agent numbers. Key performance indicators such as computational load, responsiveness, goal completion time, and inter-agent interference will be monitored and analyzed. The findings will inform the scalability limits of the current architecture and suggest potential optimizations for larger-scale deployments.

Another planned enhancement is the **introduction of more advanced LLM command capabilities**. These will include the ability to stop a specific robot mid-task or instruct it to return to its original starting point. Such functionality increases the flexibility and safety of the system, particularly in scenarios involving dynamic human intervention or emergency stops. These features will be designed using natural language command parsing, expanding on the current LLM integration.

As part of the system validation, extensive **performance data will be collected** under a variety of experimental conditions. These will include different obstacle layouts, goal configurations, and human-issued commands. The collected data will support quantitative evaluations and enable comparison across different test scenarios.

Following data collection, a comprehensive **evaluation phase** will be conducted to measure the system’s responsiveness to real-time inputs, success rate in reaching assigned goals, and accuracy of the LLM in interpreting and executing user commands. Logs and simulation recordings will be used to support this analysis.

Finally, preparations will begin for the **final project presentation and demonstration video**. This will include showcasing the key features of the system in action, highlighting successful multi-robot coordination, dynamic goal assignment through LLMs, and real-time obstacle avoidance. The presentation will summarize all development stages, research contributions, and findings, serving as a complete demonstration of the project outcomes.

## Evaluation Methods

To assess the effectiveness and robustness of the developed multi-robot navigation system, a multi-layered evaluation strategy will be employed. This evaluation focuses on task performance, human-robot interaction responsiveness, LLM accuracy, and system behavior under varying environmental and control conditions.

One of the primary evaluation methods involves comparing the total task completion time and collision avoidance success rates across different scenarios. These scenarios will vary in obstacle density (none, sparse, dense), number of robots (two or more), and the number of simultaneous goals. This will help assess how environmental complexity and multi-agent coordination influence navigation efficiency and safety.

The system’s integration with GPT-4 will be evaluated by measuring the LLM parsing precision, which involves comparing the expected goal commands (based on user input) with the actual coordinates interpreted and executed by the robots. Misinterpretations, errors in target location, or failure to trigger an action will be recorded and analyzed to understand the limitations of natural language command interpretation.

To gain a deeper understanding of how the system operates during execution, all runs will be documented using logs and annotated simulation recordings. These logs will include timestamps, robot positions, sensor readings, path changes, and decision-making events. Annotated screenshots and videos from CoppeliaSim will serve as visual evidence of robot behavior during critical moments, such as obstacle encounters or coordination episodes.

An additional method will involve comparing goal-reaching accuracy in obstacle-free and obstacle-rich environments. This will measure the precision of the robot’s final location relative to the assigned goal coordinates under both conditions. Deviations will be quantified using Euclidean distance to assess how sensor-based avoidance affects path fidelity.

The system will also be tested for its responsiveness to real-time LLM instructions during task execution. For example, while a robot is en route to a goal, a human-issued command may request it to halt, return to base, or reroute to a new goal. The latency between the command issuance and robot reaction will be recorded to determine how quickly and reliably the system adapts to dynamic human input. This reflects the core human-in-the-loop functionality being tested in the project.

Finally, all performance data will be statistically summarized and visualized through tables and graphs to support comparative analysis and facilitate discussion in the final report. These evaluations are crucial in addressing the project's research questions regarding robot cooperation, obstacle complexity, and human intervention via LLMs.

## Timeline

| **Task** | **Target Date** |
| --- | --- |
| Project Planning and Final Design Adjustments | July 14 – July 17 |
| Implement Inter-Robot Clearance Behavior | July 18 – July 22 |
| Expand to Multi-Robot (>2) System | July 23 – July 27 |
| Enhance LLM Command Capabilities (stop, return, etc.) | July 28 – Aug 1 |
| Finalize LLM Parser and JSON Integration | Aug 2 – Aug 5 |
| Conduct Obstacle vs. No-Obstacle Accuracy Testing | Aug 6 – Aug 9 |
| Evaluate Real-Time Responsiveness to LLM Commands | Aug 10 – Aug 13 |
| Log Metrics: Task Time, Goal Accuracy, Collision Rate | Aug 14 – Aug 16 |
| Compile Annotated Logs and Simulation Visuals | Aug 17 – Aug 18 |
| Analyze Results and Generate Graphs/Tables | Aug 19 – Aug 21 |
| Final Report Writing | Aug 22 – Sep 4 |
| Prepare Presentation and Demo Video | Sep 5 – Sep 12 |
| Final Submission | Sep 15 |

# Bibliography

* Gourley, C. and Trivedi, M.M., 1994. *Sensor-based obstacle avoidance and mapping for fast mobile robots*. Proceedings of IEEE Conference on Intelligent Robots and Systems. Available at: <https://ieeexplore.ieee.org/document/407951>
* Längle, T. and Wörn, H., 2001. *Human–robot cooperation using multi-agent-systems*. *Journal of Intelligent and Robotic Systems*, 32, pp.143–159.
* KUKA, 2020. *Platform taking into account real design of Mecanum wheels (as exemplified by KUKA youBot)*. *Russian Journal of Nonlinear Dynamics*, 16(2), pp.291–307. Available at: <https://www.mathnet.ru/eng/nd711>
* Candra, A., Budiman, M.A. and Hartanto, K., 2020. *Dijkstra’s and A-Star in Finding the Shortest Path: A Tutorial*. In: 2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA). IEEE, pp.1–6. Available at: <https://ieeexplore.ieee.org/document/9441911>
* Parisotto, E., Song, S., Zhang, C., et al., 2021. *Neural SLAM: Learning to Explore with External Memory*. In: International Conference on Learning Representations (ICLR). Available at: <https://arxiv.org/abs/1706.09520>
* Farley, A., Wang, J. and Marshall, J.A., 2022. *How to Pick a Mobile Robot Simulator: A Quantitative Comparison of CoppeliaSim, Gazebo, MORSE and Webots*. *Simulation Modelling Practice and Theory*, 120, 102629. Available at: <https://doi.org/10.1016/j.simpat.2022.102629>
* Mu, Q., Zhao, Y., Gao, Q. and Wang, X., 2023. *ChatGPT and Knowledge Graph for Human-Robot Interaction*. In: Xie, L. et al. (eds) *Intelligent Robotics and Applications*. Lecture Notes in Artificial Intelligence, vol 14398. Springer, Singapore. pp.340–351. Available at: <https://doi.org/10.1007/978-981-99-6495-6_30>
* OpenAI, 2023. *ChatGPT for Robotics: A New Approach to Human-Robot Interaction and Task Planning*. OpenAI Technical Report. Available at: <https://openai.com/research>

# Appendices

* Code samples
  + Path Planning with A\* (From astar.py)

A screen shot of a computer code

AI-generated content may be incorrect.

**Purpose**: Computes the shortest path between start and goal grid points using the A\* algorithm and converts it to metric coordinates.

* + Robot Motion with Obstacle-Aware Execution (From path\_executor.py)
* Appendix B – Robot Navigation Paths (Planned vs. Executed)
  + Figure B1 – Planned Path (Grid-Based A\*)

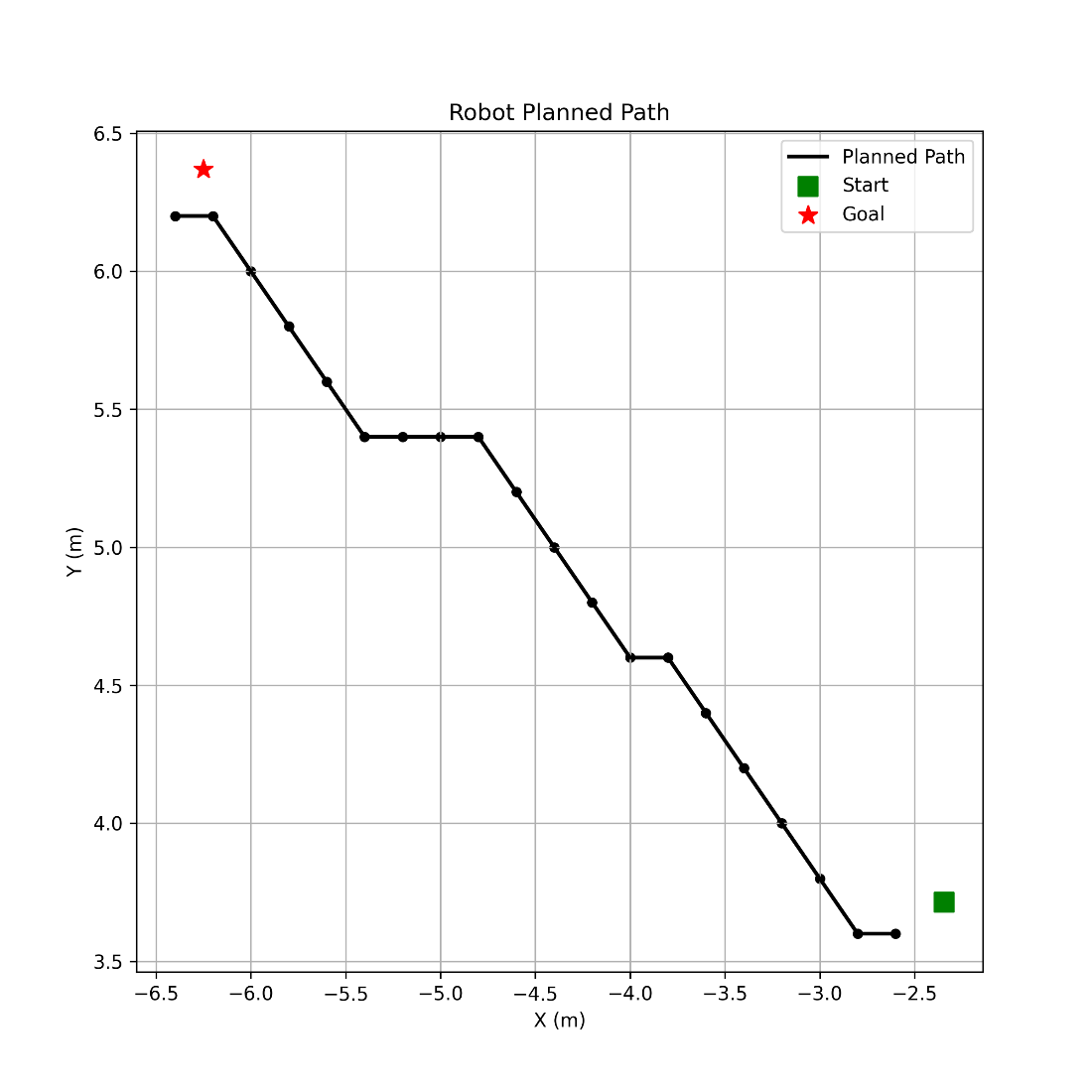


Figure 1 Planned Path (Grid-Based A\*)

This image represents the path generated by the A\* algorithm using a discretized occupancy grid. The robot calculates the optimal path by moving through the centers of grid cells, which explains the offset between the robot's **actual start position** and the **start of the planned path**, as well as a similar offset near the goal. The start and goal nodes in the planned path are snapped to the nearest grid centers for consistency in path computation.

* Figure B2 – Executed Path Without Obstacles

A graph of a robot execution

AI-generated content may be incorrect.

Figure 2 Executed Path Without Obstacles

This image shows the actual robot trajectory when executing the planned path in a static environment without obstacles. The robot follows the general direction of the planned path but deviates slightly due to kinematic constraints, velocity smoothing, and real-world execution delays. The offset between the final executed position and the goal location is within the system's acceptable **error threshold**, which defines how close the robot needs to be to the goal to consider the task successful.

* Figure B3 – Executed Path With Obstacle Avoidance

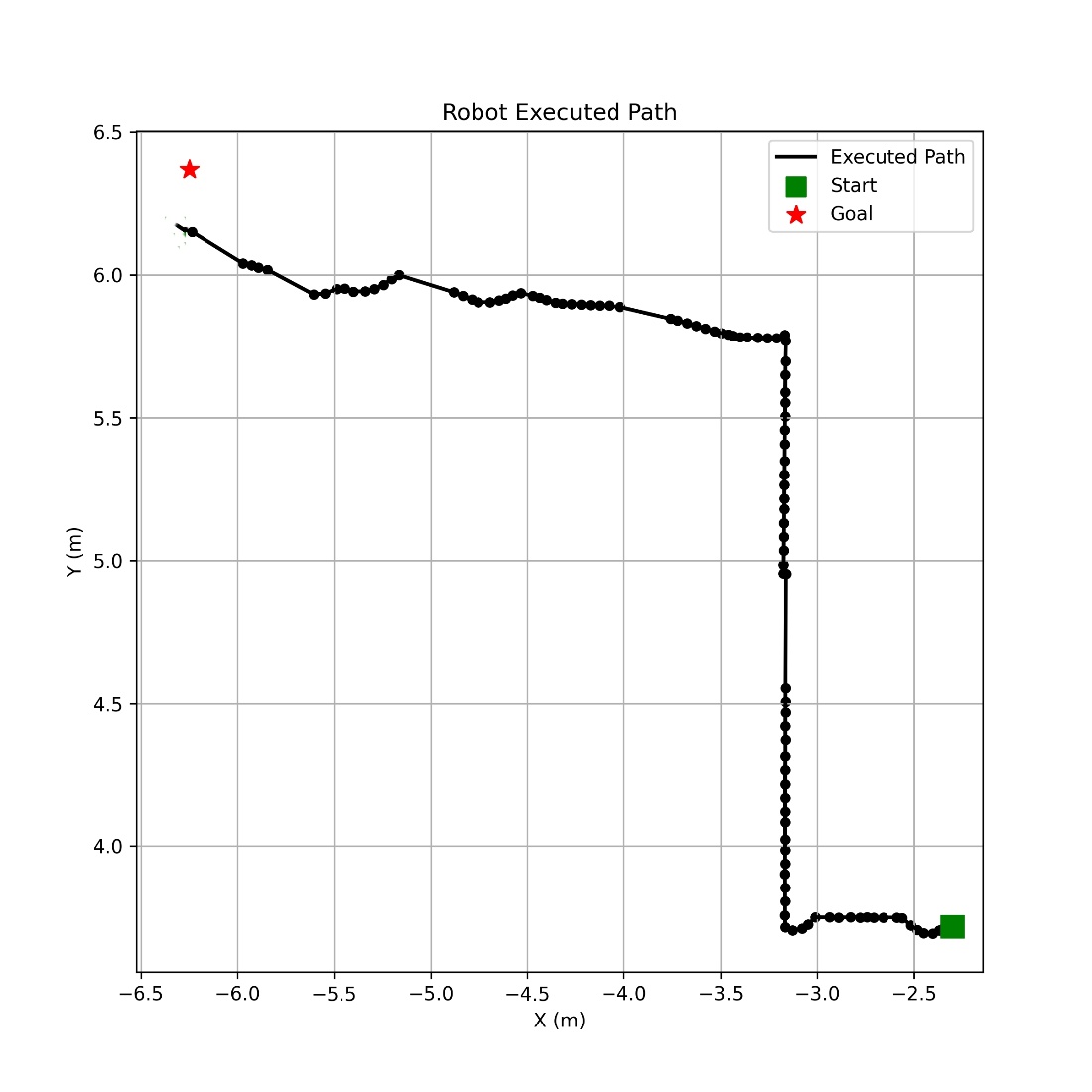


Figure 3 Executed Path with Obstacle Avoidance

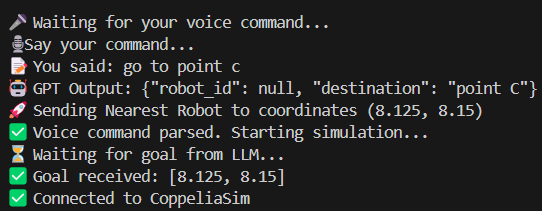
This trajectory illustrates the robot’s behavior when real-time obstacle avoidance is activated. The path includes deviations from the original A\* plan to sidestep detected obstacles using the SICK S300 directional sensors. The path is dynamically recalculated through recursive decision-making (e.g., axis alignment or sidestepping logic) when the robot detects blockage. Despite the detours, the robot converges toward the goal region, again with minor positional error due to physical constraints and dynamic corrections.

* Sample GPT prompts and outputs

A screenshot of a computer program

AI-generated content may be incorrect.

When the user says *“what are you doing now”*, the LLM returns a null destination. The system detects the invalid input and waits for a valid follow-up. Even with *“robot 1 what are you doing now”*, the result is the same. This highlights how the system handles unsupported queries without crashing.



The user command *“go to point C”* is correctly parsed by the LLM, which assigns the task to the nearest robot and converts it to coordinates [8.125, 8.15]. The system acknowledges the command and starts the simulation. This shows successful goal assignment without specifying a robot.

A screenshot of a computer program

AI-generated content may be incorrect.

In this case, the user says *“Road zero go to point A”*. The LLM correctly identifies Robot0 and destination point A, converting it to [1.5, 3.0]. The system starts the simulation with the correct robot and goal. This shows how manual robot selection works using natural language.

* Sensors Execution
* Gantt Chart

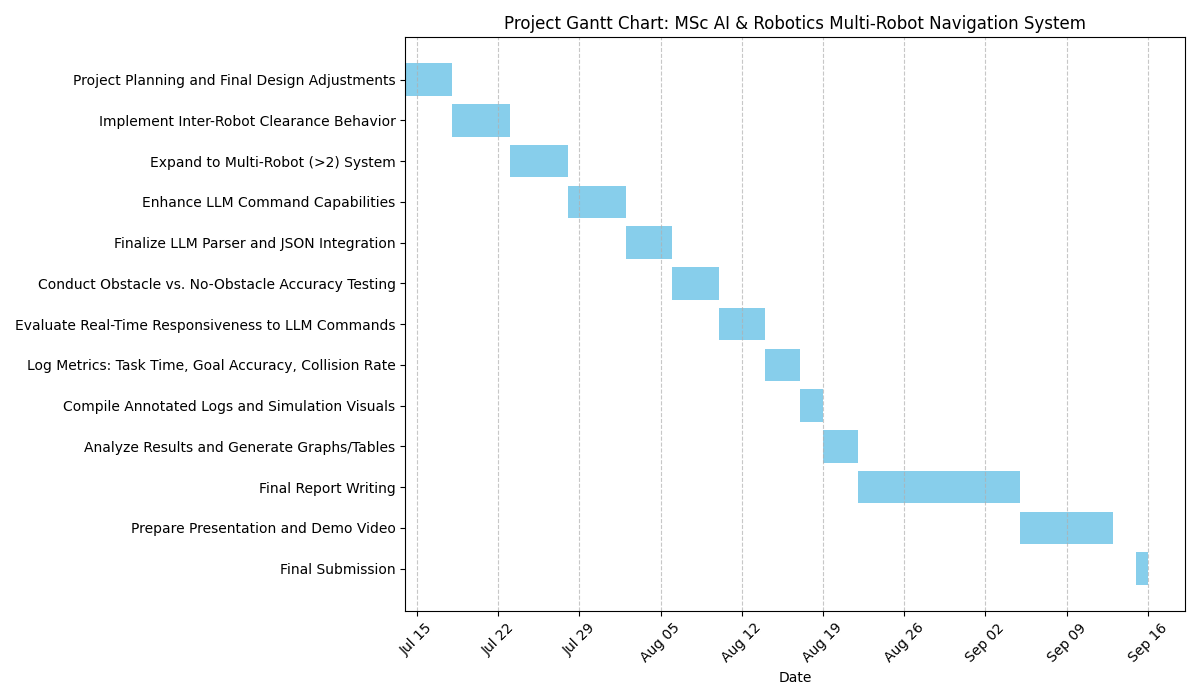


Figure 4 Gantt Chart for the planned work