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

1. Introduction and Overview

1.1. Background and Motivation

Multi-robot autonomous systems are increasingly critical in industrial, service, and research contexts, where distributed agents must navigate dynamic environments and coordinate to complete complex tasks. Their success depends not only on classical path planning (e.g., A*) and local obstacle avoidance (Candra et al., 2020; Li et al., 2017) but also on the ability to adapt to changing instructions, environments, and constraints.

Recent advances in Large Language Models (LLMs), notably OpenAI's GPT-4, have introduced new possibilities for human-robot interaction. These models can interpret flexible, ambiguous natural language input and convert it into structured task commands (OpenAI, 2023). As robotic systems grow in complexity, there is a growing need for interfaces that offer both intuitive control and interpretable reasoning.

This project explores how LLMs can serve as human-in-the-loop controllers in a multi-robot system, enabling speech- or text-based goal assignment alongside autonomous planning and coordination. By integrating LLMs with reactive navigation and path-planning logic, the system aims to assess real-time instruction parsing, inter-agent communication, and coordination behavior under various environmental and intervention conditions.

This study responds to current research gaps by investigating ambiguity handling, error propagation, and coordination stability in LLM-driven multi-agent systems, while contributing metrics and evaluation frameworks for future development and benchmarking.

2. Literature Review

2.1. Framing Large Language Models in Multi-Robot Systems

LLMs introduce new opportunities for flexible human-robot interaction but also raise significant safety and interpretability concerns. While traditional systems use deterministic rule-based or reactive logic (Gourley & Trivedi, 1994), LLMs like GPT-4 function as symbolic–sub symbolic bridges, translating natural language into structured robot commands. This project adopts an interface to explore task delegation in dynamic multi-robot environments.

Recent studies (Li et al., 2025; Agashe et al., 2023) highlight issues such as hallucination, latency, and lack of intention modeling, especially in multi-agent coordination. However, little work examines how instruction ambiguity, LLM parsing errors, or human trust affect real-time performance.

This project addresses that gap by evaluating:

- Ambiguity Sensitivity: How vague or context-dependent instructions impact goal selection and execution.
- Error Propagation: Whether parsing failures (e.g., incorrect goal JSON) lead to navigation conflicts or collisions.
- Trust Calibration: How users learn to predict, correct, or adapt to LLM behavior over time.

In contrast to prior work focused on offline planning or single-agent tasks (Mu et al., 2023; OpenAI, 2023), this system enables live, asynchronous human input and evaluates its effect on coordination success, delay, and safety. These experiments contribute to the emerging literature on LLM-in-the-loop safety and the design of interpretable, robust instruction systems.

2.2. Coordination and Planning in Multi-Agent Systems

Most classical coordination frameworks (e.g., centralized planners, auction-based task allocation, swarm intelligence) prioritize performance guarantees under constrained assumptions. Recent work like Agashe et al. (2023) evaluates LLM-based coordination in comparison with reinforcement learning, highlighting LLMs' limitations in theory-of-mind reasoning and negotiated decision-making—key components in dynamic, human-supervised systems.

While Candra et al. (2020) demonstrates A*'s computational efficiency for single-agent scenarios, the current project extends this by observing A* performance under asynchronous agent-goal allocation*, where real-time interference and re-planning simulate practical constraints.

The work also contrasts with Dec-POMDP (Decentralized Partially Observable Markov Decision Process) approaches, which remain computationally intractable at scale but offer theoretical grounding for decision conflict resolution—an area not yet addressed in LLM-based systems.

2.3. Human–Robot Interaction and Language Interfaces

Längle & Wörn (2001) provide an early foundation in role negotiation within human-robot teams, while Mu et al. (2023) explore LLM-Knowledge Graph (KG) integration for richer task understanding. However, most current systems—including this one—omit the KG layer, leaving semantic gaps unaddressed.

This project contributes to this discussion by testing natural language to JSON command reliability, observing failure cases due to ambiguity or hallucination, and measuring how agent response varies with command timing, specificity, and recursion.

The reliance on GPT-4 builds on OpenAI (2023), but this review problematizes the dependence on vendor-specific models lacking transparent training data, benchmark reproducibility, or error diagnosis tools—key to trust in interactive robotics.

2.4. Physical Embodiment and Simulation Fidelity

Simulated embodiment is central to validating coordination schemes. Farley et al. (2022) position CoppeliaSim as a superior environment for simulating contact dynamics, which justifies its use here. This project leverages vision-based obstacle perception and wheel-level kinematics, extending the work of KUKA youBot (2020) to reflect realistic slip/friction constraints in omnidirectional robots.

Yet, while the current project remains in simulation, emerging real-world validation—e.g., LLM-Flock, HMCF (Li & Zhou, 2025)—points to the growing need for physical benchmarks, particularly regarding communication lag, safety boundaries, and control noise.

3. Identified Research Gaps

- Main Research Question:

H0: The implemented obstacle avoidance logic performs equally well across static and dynamic environments.
H1: Environmental complexity and the presence of dynamic obstacles significantly reduce task completion success and increase navigation time.

- Sub-Research Questions:

  1) Multi-Agent Coordination

  H0: Multi-agent coordination has no measurable impact on task efficiency or response time.

  H1: Coordination between agents improves task efficiency in low obstacle environments but degrades in cluttered/dynamic settings due to LLM latency and lack of intention modeling.

  2) Human-in-the-Loop Interference

  H0: LLM-based human commands issued mid-task do not affect the performance of nearby agents.

  H1: Human intervention for one robot (via GPT-4) leads to delay, rerouting, or collision risk for other agents in shared space.

  3) Obstacle Type Sensitivity

H0: The robot's navigation performance is unaffected by obstacle type.

H1: Dynamic obstacles and agent-like obstructions (other robots) lead to higher sidestepping frequency and increased rerouting time compared to static obstacles.

4. Methodology

This project implements a modular, simulated multi-robot navigation system combining classical path planning (A*), local obstacle avoidance, and human-in-the-loop task delegation via Large Language Models (LLMs). The aim is to assess system performance and inter-agent coordination in dynamic, partially observable environments, while comparing GPT-4–based goal assignment with non-LLM baselines.

4.1. Simulation Environment

The system is developed in CoppeliaSim, chosen for its high-fidelity physics engine, ZeroMQ support, and realistic omnidirectional robot dynamics. Compared to Gazebo and Webots, CoppeliaSim was rated highest in modularity and physics accuracy for multi-agent research (Farley et al., 2022). The environment is structured as a 2D occupancy grid containing:

- Static obstacles (walls, barriers)
- Dynamic obstacles (moving blocks, other robots)
- Multiple start/goal configurations

Each robot is equipped with four SICK S300 directional sensors to support local obstacle awareness (front, back, left, right), enabling reactive navigation based on proximity feedback (Gourley & Trivedi, 1994).

4.2. Robot Platform

Robots are modeled on the KUKA youBot using Mecanum wheels for planar omnidirectional motion. Wheel velocity mappings are adapted from real-world kinematic models (KUKA, 2020) to support lateral sidestepping, critical for navigating tight corridors and resolving inter-agent blockage scenarios.

4.3. Navigation and Control

Global Path Planning: A* algorithm operates on the occupancy grid, updating paths in real time based on new obstacle detections. A* was selected for its speed and optimality tradeoff over Dijkstra (Candra et al., 2020).

Local Obstacle Avoidance: Based on sensor input, robots dynamically reroute or sidestep when obstacles are detected along the current path.

4.4. Human-in-the-Loop Goal Assignment

Goal assignment is conducted via two contrasting methods:

Method	Description
GPT-4 LLM	Accepts natural language or speech input, parsed into structured JSON using the OpenAI API (OpenAI, 2023).
Rule-Based GUI	A baseline GUI with dropdowns and coordinate inputs; no LLM parsing involved.

This control comparison allows quantifying the impact of LLM latency, errors, or ambiguity on system performance. GPT-4 commands may include mid-task reassignment, emergency stops, or return-to-base orders, simulating realistic supervision scenarios.

4.5. Inter-Agent Coordination

Multiple robots operate concurrently. Task assignment is determined by:

- Nearest robot heuristic for single goals
- Task splitting for multiple goal inputs.

Coordination behaviors include:

- Path yielding when robots block one another.
- Dynamic rerouting if blockage persists.
- Communication delay simulation to model realistic lag.

This cooperative behavior draws inspiration from agent role-negotiation frameworks (Längle & Wörn, 2001).

4.6. Experimental Conditions and Matrix

To evaluate the hypotheses (see Section 1.2), the system is evaluated across a controlled experimental matrix:

Condition	Variables Manipulated
Obstacle Complexity	Static-only, Dynamic-only, Mixed
Robot Count	1, 2, 3, 5
Goal Assignment Method	LLM (GPT-4), GUI rule-based
Human Intervention Timing	None, Mid-task reassignment, Emergency stop
Task Type	Point-to-point, Multi-goal delivery, Return-to-base

4.7. Evaluation Metrics

Each scenario is measured using the following quantitative indicators:

- Task Completion Time (TCT): Time from goal assignment to final arrival.
- Path Deviation: Distance traveled vs. optimal A* path.
- Collision Rate: Number of contacts with obstacles or other robots.
- Blocking Duration: Time one robot remains blocked by another.
- LLM Parsing Time: Time from command to action execution.
- Command Success Rate: Proportion of commands resulting in correct task completion.
- Intervention Impact Score: Change in TCT or collision rate when a mid-task command is issued.

Using these metrics, the evaluation will compare scenarios based on obstacle density, number of robots, and goal assignment method. Parsing accuracy and trust calibration will be analyzed by measuring how well GPT-4 interprets natural language commands and how users respond to errors or misinterpretations. Robot response to real-time interventions, final goal precision, and adaptation to failed commands will be documented using logs and simulation visualizations.

Results will include means, standard deviations, and confidence intervals where applicable. Statistical tests (e.g., t-test, ANOVA) will determine the significance of differences between LLM vs. GUI setups and across obstacle configurations.

5. Proposed One-Month Methodology

This one-month implementation cycle ensures both system readiness and experimental rigor through a structured four-week plan:

Week 1 – Simulation Setup and Baseline Definition

- Build a 2D occupancy-grid-based simulation with 2–3 robots in CoppeliaSim.
- Introduce dynamic and static obstacles.
- Implement:
    - A* path planner
    - Sensor-based local avoidance
    - GPT-4 goal parsing.
    - GUI baseline controller
- Define tasks involving collaborative goal pursuit and real-time updates.

Week 2 – Instrumentation and Metric Logging

- Log quantitative indicators:
    - Coordination Success Rate (CSR)
    - Joint-plan adherence
    - LLM latency
    - Command parsing accuracy.
    - Hallucination frequency
    - Collision count
- Implement automated CSV/JSON log generation and timestamped event tracking.

Week 3 – Controlled Trials

- Run **≥30 trials per condition**, varying:
    - Number of robots (1, 2, 3+)
    - Obstacle density (none, static, dynamic, mixed)
    - Command timing (pre-task, mid-task, emergency)

- **Include both GPT-4 and GUI-based goal assignment.**
- **Capture** all logs, sensor data, and visuals.

Week 4 – Evaluation and Human-in-the-Loop Study

- Perform statistical analysis using ANOVA and t-tests with 95% CI.
- Evaluate how ambiguity and timing in LLM commands affect coordination.
- Conduct a small user study (3–5 participants) issuing mid-task instructions to evaluate LLM parsing robustness.
- Analyze failure cases, misinterpretations, and recovery behavior.

This plan will also include inter-robot clearance behavior to improve coordination, support for scaling the system to more than two robots, and the implementation of advanced LLM command capabilities such as mid-task stop and return commands. Extensive performance data will be collected across different obstacle layouts, task types, and human interventions. The findings will inform the system's scalability, responsiveness, and LLM reliability in dynamic environments.

6. Expected Outcomes and Contributions

This methodology enables a scientific evaluation of LLM-based multi-robot navigation, delivering:

- Quantified benchmarks for LLM reliability in agent coordination.
- Latency and hallucination impact models for human-in-the-loop robotics.
- Insights into coordination degradation under ambiguity or real-time interruption.
- A reproducible experimental design and simulator toolkit suitable for future academic extensions or teaching.

7. Project Artefact

The developed artefact is a comprehensive, modular Python-based simulation system integrated with CoppeliaSim via ZeroMQ, designed for intelligent multi-robot navigation, coordination, and human-LLM interaction.

Each omnidirectional robot in the simulation is equipped with four directional SICK S300 sensors for obstacle detection and avoidance. The robots can interpret human commands (spoken or written), navigate using A* path planning, and dynamically cooperate with each other to avoid collisions or blocking.

System Features:

- **Natural Language Interaction**: Commands are processed through GPT-4, enabling intuitive goal assignment.

  (This project does not involve supervised or unsupervised training of models but evaluates the integration of pretrained LLMs for decision-making in robotic control systems.)

- **Autonomous Navigation**: A* planning on a grid-based map with dynamic occupancy updates.

- **Obstacle Avoidance**: Real-time reactive behavior using SICK S300 sensor data.

- **Inter-Robot Coordination**: Dynamic reassignment, collision prevention, and path unblocking.

Modules developed include:

- main.py: Launches and manages the full simulation lifecycle.
- run.py: Entry point for execution, integrating LLM, navigation, and motion.
- LLM.py: Handles speech recognition, GPT-4 query interaction, and JSON goal formatting.
- astar.py & astar_env.py: Implement A* pathfinding and environment handling.
- map_builder.py: Constructs grid maps from simulated environment data.
- path_executor.py: Converts waypoint paths into movement actions.
- robot_controller.py: Manages robot decisions, task status, and behavior switching.
- robot_motion.py: Sends motion commands to the simulation for specific movement types.
- obstacle_awareness.py: Contains the real-time avoidance logic based on sensor input.
- sensor_fetch.py: Collects raw sensor readings from each SICK sensor.
- check_nearest_robot.py: Assigns tasks to the nearest available robot.
- plotter.py: Visualizes planned and executed paths for evaluation.
- shared.py: Defines constants, thresholds, and utility functions.
- sim_client.py: Manages the CoppeliaSim connection using ZeroMQ and sim.py API.

8. 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

9. Deliverables

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

10. Ethical, Legal, Professional, and Social Issues

In the development and execution of this project, several ethical, legal, professional, and social considerations have been considered 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.

11. Progress to Date

11.1. 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.

11.2. 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.

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.

12. Work 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

Task	Target Date
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

13. Bibliography

1. Adamov, B.I. & Saipulaev, G.R. (2020) 'Research on the dynamics of an omnidirectional platform taking into account real design of Mecanum wheels (as exemplified by KUKA youBot)', Russian Journal of Nonlinear Dynamics, 16(2), pp. 291–307. https://doi.org/10.20537/nd200205

2. Agashe, S., Fan, Y., Reyna, A. & Wang, X.E. (2023) LLM-Coordination: Evaluating and Analyzing Multi-Agent Coordination Abilities in Large Language Models. arXiv preprint arXiv:2310.03903. Available at: https://doi.org/10.48550/arXiv.2310.03903

3. Candra, A., Budiman, M.A. & 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 2020), Medan, Indonesia, 16–17 July, pp. 19–32. https://doi.org/10.1109/DATABIA50434.2020.9190342

4. Chen, J., Yang, Z., Xu, H.G., Zhang, D. & Mylonas, G. (2025) Multi-Agent Systems for Robotic Autonomy with LLMs. arXiv preprint arXiv:2505.05762. Available at: https://doi.org/10.48550/arXiv.2505.05762

5. Farley, A., Wang, J. & Marshall, J.A. (2022) 'How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion', Simulation Modelling Practice and Theory, 120, 102629. https://doi.org/10.1016/j.simpat.2022.102629

6. Längle, T. & Wörn, H. (2001) ‘Human–robot cooperation using multi-agent systems’, Journal of Intelligent and Robotic Systems, 32(2), pp. 143–160. https://doi.org/10.1023/A:1013901228979

7. Li, P. & Zhou, L. (2025) LLM-Flock: Decentralized Multi-Robot Flocking via Large Language Models and Influence-Based Consensus. arXiv preprint arXiv:2505.06513. Available at: https://doi.org/10.48550/arXiv.2505.06513

8. Li, P., An, Z., Abrar, S. & Zhou, L. (2024) Large Language Models for Multi-Robot Systems: A Survey. arXiv preprint arXiv:2502.03814. Available at: https://doi.org/10.48550/arXiv.2502.03814

9. Li, Z., Wu, W., Wang, Y., Xu, Y., Hunt, W. & Stein, S. (2025) HMCF: A Human-in-the-Loop Multi-Robot Collaboration Framework Based on Large Language Models. arXiv preprint arXiv:2505.00820. Available at: https://doi.org/10.48550/arXiv.2505.00820

10. Mu, Z., Zhao, W., Yin, Y., Xi, X., Song, W., Gu, J. & Zhu, S. (2023) ‘KGGPT: Empowering robots with OpenAI’s ChatGPT and Knowledge Graph’, in Yang, H., Liu, H., Zou, J., Yin, Z., Liu, L., Yang, G., Ouyang, X. & Wang, Z. (eds.) Intelligent Robotics and Applications – 16th International Conference, ICIRA 2023, Hangzhou, China, July 5–7, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14271. Singapore: Springer, pp. 340–351. https://doi.org/10.1007/978-981-99-6495-6_29

11. Zhang, J., Tai, L., Liu, M., Boedecker, J. & Burgard, W. (2017) Neural SLAM: Learning to Explore with External Memory. arXiv preprint arXiv:1706.09520. Available at: https://doi.org/10.48550/arXiv.1706.09520

14. Appendices

1) Code

Everything is in the GitHub public repository: https://github.com/mohamaddirani/MSc-FInal-Project

2) Appendix B – Robot Navigation Paths (Planned vs. Executed)
    - Figure B1 – Planned Path (Grid-Based A*)

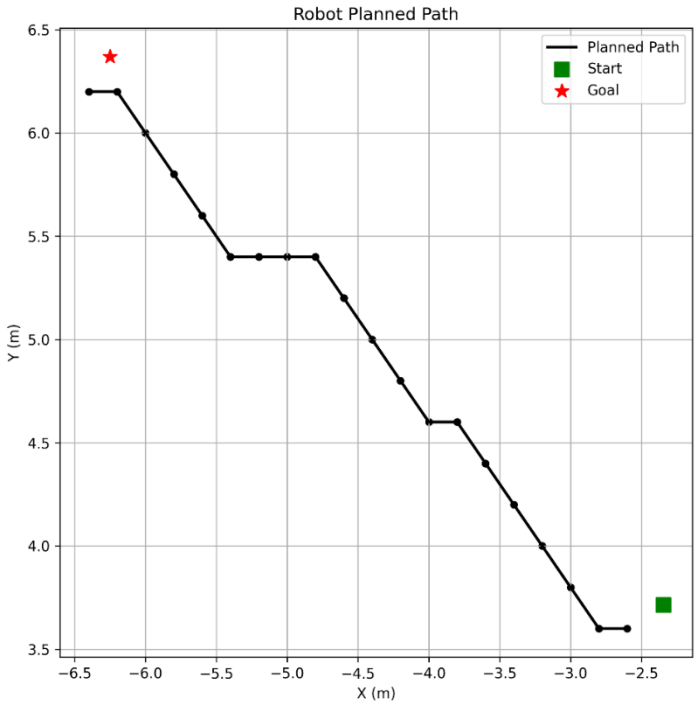


Figure 1 Planned Path (Grid-Based A)*

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

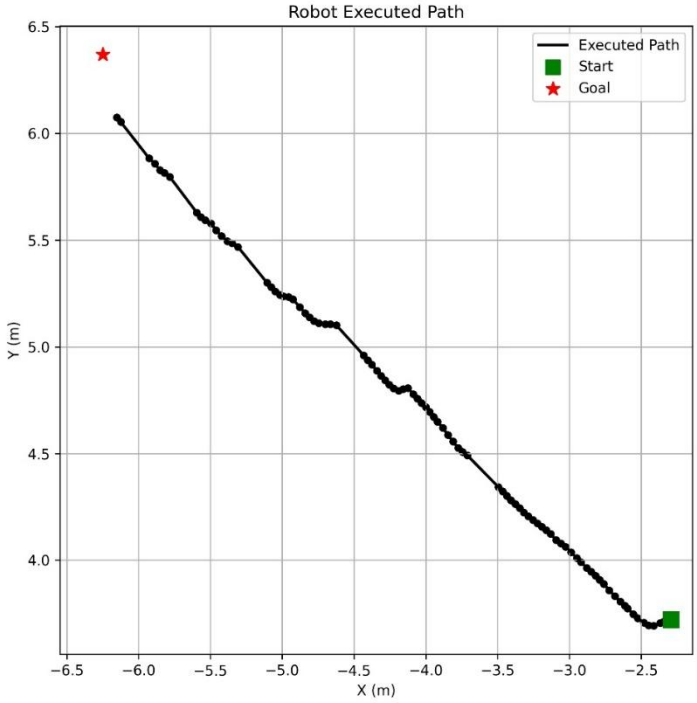


Figure 2 Executed Path Without Obstacles

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

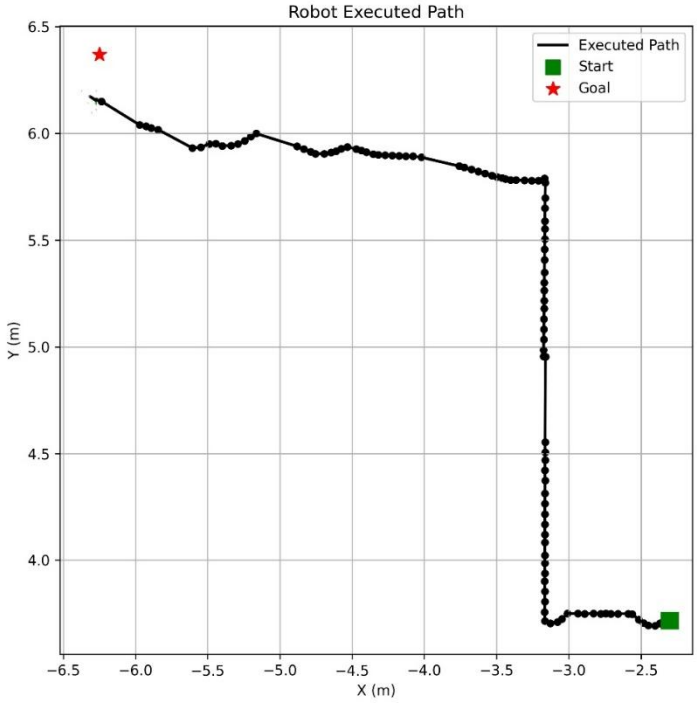


Figure 3 Executed Path with Obstacle Avoidance

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.

3) Sample GPT prompts and outputs.

```
🎤 Waiting for your voice command...
🎙️Say your command...
📝 You said: what are you doing now
🤖 GPT Output: {"robot_id": null, "destination": null}
⚠️No destination provided. Waiting for a valid command.
❌ Unknown destination: None
🔁 Waiting for a valid command. Try again...

🎙️Say your command...
📝 You said: robot 1 what are you doing now
🤖 GPT Output: {"robot_id": "Robot1", "destination": null}
⚠️No destination provided. Waiting for a valid command.
❌ Unknown destination: None
🔁 Waiting for a valid command. Try again...
```

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.

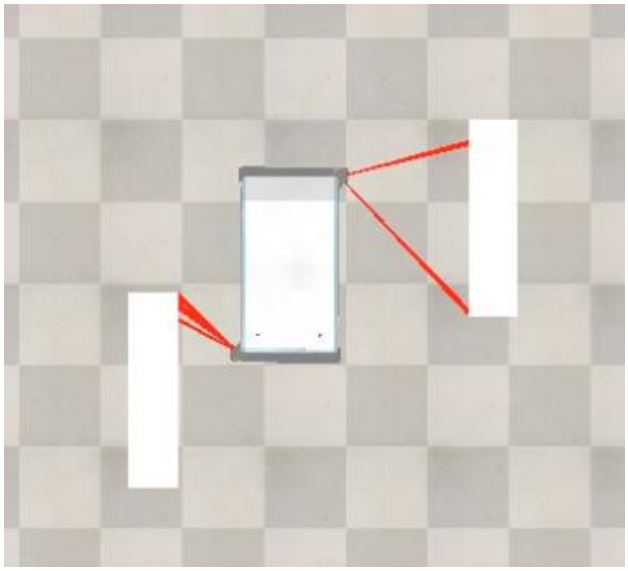
```
🎤 Waiting for your voice command...
🎙️Say your command...
📝 You said: go to point c
🤖 GPT Output: {"robot_id": null, "destination": "point C"}
🚀 Sending Nearest Robot to coordinates (8.125, 8.15)
✅ Voice command parsed. Starting simulation...
⏳ Waiting for goal from LLM...
✅ Goal received: [8.125, 8.15]
✅ Connected to CoppeliaSim
```

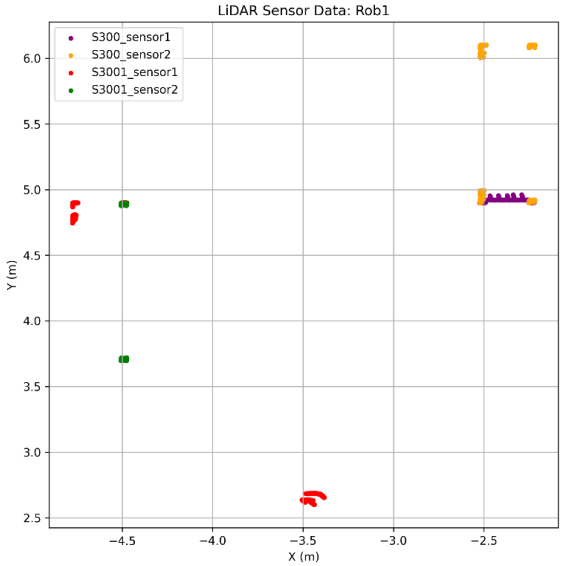
The user command *"go to point C"* is correctly parsed by the LLM, which assigns the task to the nearest robot and converts it to coordinates [8.125, 8.15]. The system acknowledges the command and starts the simulation. This shows successful goal assignment without specifying a robot.

```
🎤 Waiting for your voice command...
🎙️Say your command...
📝 You said: Road zero go to point a
🤖 GPT Output: {"robot_id": "Robot0", "destination": "point A"}
🚀 Sending Robot0 to coordinates (1.5, 3.0)
✅ Voice command parsed. Starting simulation...
⏳ Waiting for goal from LLM...
✅ Goal received: [1.5, 3.0]
✅ Connected to CoppeliaSim
```

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.

4) Sensors Execution





This LiDAR visualization shows real-time obstacle detection from all four SICK S300 sensors on Rob1. Each color corresponds to a specific direction, validating the robot's awareness of its surroundings. This sensor feedback is used in the recursive obstacle avoidance and axis-alignment logic to navigate safely in cluttered environments.

5) Gantt Chart

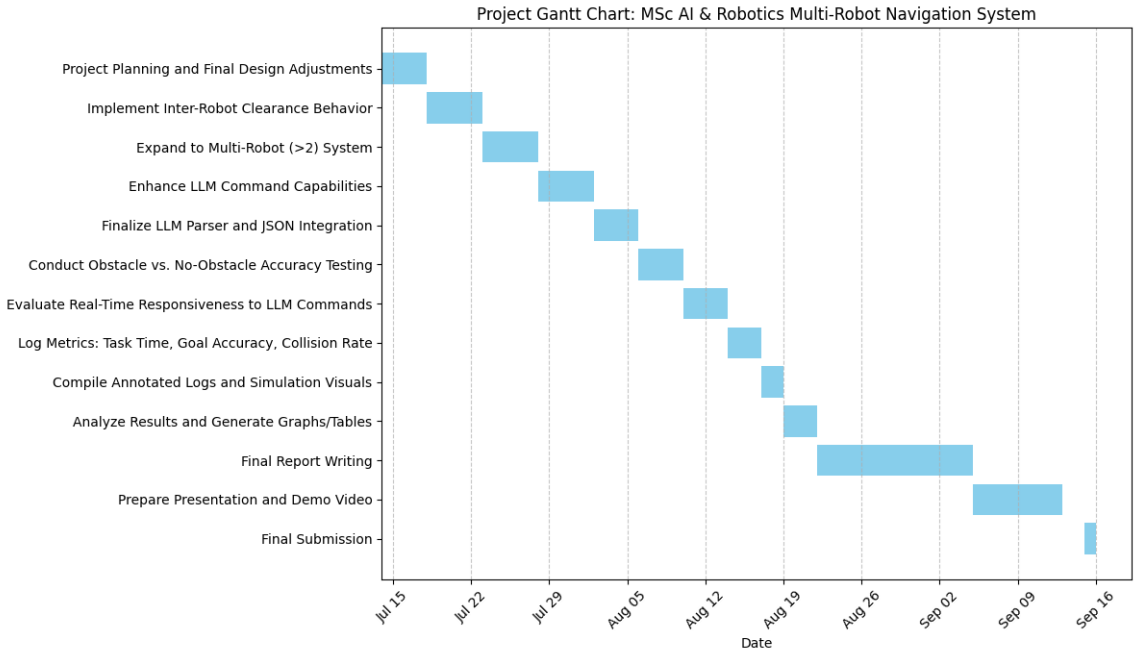


Figure 4 Gantt Chart for the planned work.