UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie
master in de informatica

Masterthesis

Large language models as drivers for robotic control

David Kellens
Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Raf RAMAKERS

2024
2025

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Large language models as drivers for robotic control

David Kellens

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Raf RAMAKERS

Acknowledgements

I would like to express my sincere gratitude to my promotor, Professor Raf Ramakers, and my supervisor, Dries Cardinaels, for their invaluable guidance and support throughout this thesis. Their insights and encouragement were essential to achieving the results and ensuring the success of this work.

I would also like to thank Jeroen Ceyssens, Tom Veuskens, Danny Leen, Mannu Lambrichts, Lennert Meurs, and Brecht Heeren for their contributions during the group meetings. Their feedback and suggestions were instrumental in overcoming challenges and exploring new approaches.

My heartfelt thanks go to my parents for their support throughout my studies. They provided me with the opportunity to pursue my education and were always there to listen and encourage me during the difficult moments.

Lastly, I want to thank the UHasselt Digital Future Lab for providing the equipment and resources necessary for conducting my research.

Summary

0.1 Introduction

The automation of manual tasks has progressed significantly since the mid-20th century, with robotics playing a key role in sectors such as manufacturing, transportation, and household chores. Traditionally, robot control relied on rule-based and hard-coded logic. However, the past decade has seen a shift toward AI-driven automation using Large Language Models (LLMs), Vision-Language Models (VLMs), and other specialized AI tools. These models have enabled more adaptive and intelligent behaviors in systems like self-driving cars and robots such as Boston Dynamics' "Spot".

Despite these advances, programming robots remains a complex challenge, especially for tasks involving reasoning, spatial awareness, and planning. While LLMs and VLMs offer new possibilities by mimicking human reasoning and being more user-friendly, they are primarily designed for conversational tasks, not robotic control. Integrating them into robotics requires overcoming limitations in planning and control.

Numerous prior works have integrated LLMs into robotic systems, but they typically rely on additional models or engineered frameworks to support or constrain the language model. Such frameworks include using learned value function to filter and select robotic skills generated by an LLM, enhancing the memory capabilities of LLMs to improve their teachability based on human feedback and using LLMs to define reward parameters, allowing a robot to optimize its behavior by selecting actions that meet those goals. While all of these approaches demonstrate promising results, they depend on extensive supporting infrastructure to guide or correct the LLM. This raises the question of whether recent advancements in LLMs are sufficient to enable effective robot control without such auxiliary components.

The central research question is: "**Can large language models navigate a mobile robot through an area, using a minimal supporting framework?**". The goal is natural language interaction with robots, where an LLM interprets and executes user instructions directly. A major technique used is prompt engineering, which helps guide the LLM's responses. The framework is intentionally kept minimal, with most logic and control delegated to the LLM, supplemented only by simple feedback from the robot's environment.

0.2 Related Work

Large Language Models (LLMs) are powerful yet opaque systems whose full behavior remains poorly understood. Their general-purpose capabilities make them valuable across many domains, including robotics. However, they also present significant limitations and risks, particularly in high-stakes applications. It is therefore essential to understand their drawbacks before deploying them in real-world scenarios. LLMs do not "think" or "reason" like humans; instead, their outputs are generated based on probabilistic patterns in text. This can lead to hallucinations, which are plausible but factually incorrect outputs, which may pose safety concerns in robotics. Several studies have proposed methods to detect and mitigate these hallucinations.

Additionally, LLMs are highly sensitive to prompt variations, often producing drastically different outputs with minor changes. Other concerns include computational cost, environmental impact, and security and privacy vulnerabilities. Human-Robot Interaction (HRI) is a key aspect of robotics, where predictability and transparency are critical for effective collaboration. LLMs can contribute positively to HRI due to their human-like language understanding. Prior work has explored their use in simulating parts of systems in HRI studies and modeling human behavior.

This thesis focuses on using LLMs for robotic navigation, an area previously explored in both known and unknown environments. In known settings, LLMs have achieved performance comparable to traditional planners such as A*, but displayed increased processing speed. In unknown environments, two main approaches have been studied: continuous mapping during exploration and building reusable maps. These systems often rely on Visual Language Models (VLMs) with camera and sensor input. In contrast, this thesis uses a minimal framework and a text-only LLM to assess navigation capabilities in isolation. HRI remains relevant in navigation tasks, both in terms of interaction with the operator and with the environment. Effective interfaces are needed for operator-LLM communication, and robots must navigate safely around humans. Some works have explored using LLMs to reason about pedestrian behavior and social connection to guide robot actions.

A key challenge in LLM-based robotics is long-horizon task planning, where models may exceed token limits or forget earlier instructions. Proposed solutions include decomposing tasks into subtasks, using structured memory systems to reduce the number of things that need to be remembered, incorporating environment graphs that store information and making LLMs remember details about the environment in their own output. While this thesis focuses primarily on short-horizon tasks, related memory issues did arise and were partially mitigated through prompt reminders and providing additional context.

Finally, prompt engineering, which is optimizing the instructions given to the LLMs, remains the most effective method for improving LLM performance. Prior studies have identified strategies such as structured input-output templates, persona assignment, few-shot prompting, contextual information inclusion, and chain-of-thought reasoning. These techniques were employed extensively throughout this work to enhance navigation outcomes.

0.3 Initial Feasibility Testing

To start, a series of feasibility tests were conducted using GPT-4o (OpenAI), Gemini-2.0-Flash (Google DeepMind), DeepSeek-V3 and DeepSeek-R1, which are among the most advanced publicly accessible LLMs. A simplified environment, a labeled chessboard grid, was chosen to test spatial understanding in a controlled setting. This setup provided a discrete, easy-to-interpret space for the models to reason about navigation, movement, and obstacles.

In the tests, a robot, representing a mobile agent, could move one square at a time in any direction, with the added requirement of turning to face the correct direction before moving forward. Obstacles on the grid added complexity, requiring the models to avoid invalid paths. These experiments offered early insight into each model's strengths and limitations in reasoning about movement and space, helping to establish whether LLMs alone can form the basis for autonomous robot control.

0.3.1 Tests and Results

To evaluate the navigation capabilities of LLMs, three types of tests were designed: executing single-step movements, planning full paths in a single response, and navigating interactively step-by-step with real-time feedback. Each test was conducted in both obstacle-free and obstacle-rich environments, with obstacles being either known (explicitly described) or unknown (inferred through failed movements). LLMs were also asked to generate ASCII-based

visualizations of the environment for clarity and traceability.

The results showed that current LLMs, GPT-4o, Gemini-2.0-Flash, and DeepSeek-V3, are capable of basic spatial reasoning and can handle simple navigation tasks when instructions are clear. They performed reasonably well in executing direct commands and could remember obstacle locations in guided scenarios. However, their consistency declined in more complex environments, especially during full-path planning where collisions or repetitive loops sometimes occurred. Diagonal paths were often ignored in favor of simpler orthogonal movement. Step-by-step navigation, where models received feedback after each move, produced better outcomes than generating entire paths at once. It allowed the LLMs to adjust to unexpected obstacles and refine their strategies. Nonetheless, hallucinations and logical inconsistencies still emerged, particularly in ambiguous situations. Among the models, DeepSeek-R1 displayed stronger performance in reasoning and obstacle handling but suffered from extremely long response times, limiting its practicality. DeepSeek-V3 improved on speed but lagged behind ChatGPT and Gemini, which responded faster overall. Visual representations were another weak point: most models struggled with consistently accurate ASCII or image-based depictions of the robot's position and environment. DALL-E 3 produced more coherent visuals than earlier versions but still misrepresented certain spatial details.

0.4 Concept

The feasibility tests demonstrated that LLMs possess the foundational capabilities for navigation, but further experimentation was needed in more controlled settings with improved prompting strategies. All interactions with the LLMs were conducted via text-based input, with the primary models being GPT-4o and Gemini-2.0-Flash, both accessed through their official APIs. Additional exploratory testing was performed using DeepSeek-R1 and LLaMA 3.1, either via third-party APIs or through local deployment. The robotic platform used was the "TurtleBot3 Waffle Pi", which is capable of basic movement commands: forward, backward, and in-place rotation in both clockwise and counterclockwise directions. Experiments were carried out in Python and Unity-based simulations as well as in physical real-world environments. The robot operated in a flat, two-dimensional space, with its position represented using Cartesian coordinates and its orientation expressed in degrees, following either Unity's or the standard mathematical convention. The environment could contain both known and unknown obstacles. Unknown obstacles had to be inferred by the LLMs themselves by running into them. In simulation, tracking the robot's position was straightforward. In real-world experiments, tracking was accomplished using an infrared camera-based motion capture system from Qualisys.

0.4.1 Input and Output

LLMs received structured Task Prompts containing instructions, environment descriptions, output formatting rules, and information about obstacle handling, error handling and output formatting. Additionally, there was a brief reminder that was iteratively updated based on the perceived weaknesses of the LLMs. For each navigation task, the LLMs were given a start point, destination, and sometimes orientation. They issued one move per message, aided by real-time feedback after each command. This feedback was in the form of the robot's updated position after executing a forward or backward move and a confirmation when turning. Prompt styles were tested with and without reasoning allowed in the response (similar to chain-of-thought prompting).

0.4.2 Prompting and Navigation Strategies

To evaluate effective prompting techniques for robotic navigation, two strategies were tested for handling robot orientation: either explicitly providing the orientation at the start or requiring the LLMs to deduce it by observing the result of a forward move. The latter aligns with the

thesis goal of minimizing external support. LLMs were also tested under two output styles: returning just the result or showing their reasoning step-by-step along with the formula.

For turning, both relative and absolute strategies were explored. Relative turning required the LLMs to track orientation changes through commands like "turn left 90 degrees" or "turn -45 degrees", using different phrasings. Absolute turning involved specifying a final orientation (e.g., "face 90 degrees," "face North" or "positive-x"), with the control system translating this into a relative turn. While direction names were effective in constrained environments, they became impractical for arbitrary angles. These tests helped identify which phrasing and strategies best supported spatial reasoning in LLM-based navigation.

0.4.3 Experimental Scenarios

Tests were conducted across three distinct scenarios, each evaluating different aspects of the LLMs' reasoning capabilities. The first scenario that was tested was a grid-based movement environment, in which the robot was constrained to move on a fixed grid similar to a chessboard. Movement was restricted to discrete steps: 1 meter for orthogonal directions and $\sqrt{2}$ meters for diagonal ones. The robot could rotate in 45-degree increments, allowing for movement in eight possible directions. This setup provided a simplified testbed to evaluate whether LLMs could perform basic spatial reasoning and control a robot accordingly. All experiments for this scenario were conducted in a Unity simulation, as the real-world TurtleBot3 lacks the motion precision required to reliably conform to grid constraints.

The second scenario involved a more realistic form of navigation, where the robot was no longer restricted to a grid and could move any distance in any direction between 0° and 360°. Experiments were conducted both in simulation, using Unity, and in the real world with the TurtleBot3. In contrast to the precise control afforded by simulation, the real-world robot was subject to physical inaccuracies such as wheel slippage and motor drift. Consequently, a tolerance of 0.5 meters from the target location was allowed for successful navigation in real-world experiments. In this scenario, absolute turning was expressed exclusively in degrees, as named directions are impractical in continuous movement. Although this scenario introduced more complex orientation calculations, the underlying optimal movement strategy was conceptually simple: the robot should rotate to face the destination and move directly toward it.

In many real-world applications, obtaining the exact coordinates of a robot at all times is not always feasible due to sensor limitations or environmental constraints. To simulate such scenarios, a final set of experiments was conducted in which the LLMs were no longer provided with precise location data. Instead, they were only given the current distance to the endpoint, challenging them to navigate with significantly reduced information. These experiments were carried out using grid-based movement in a Python simulation to reduce the complexity of the task while maintaining structure. The underlying strategy for this scenario was to attempt moves in various directions and observe whether the distance to the target decreased. While this task can be trivially solved using a simple trial-and-error algorithm, the objective was to assess whether LLMs could demonstrate spatial reasoning to identify and converge on the correct path more efficiently. Obstacles were not tested here, as they would be difficult to detect and showed poor results in other scenarios.

0.5 Implementation

The temperature and top-p parameters for both ChatGPT and Gemini were set to 0.1 to ensure more deterministic output while maintaining enough variability to enable error correction. Communication with these models was handled via their official APIs using Python code. LLaMA 3.1 and DeepSeek-R1 were run locally using the program Ollama, while DeepSeek-R1 was also accessed through a third-party API provider called OpenRouter, again using Python for integration.

The primary simulation environment was built in Unity, where a virtual model of the TurtleBot3 navigated within a modeled room. This setup included two main components: one module for controlling the robot's movement and another for establishing a socket connection to the Python backend that interfaced with the LLMs. Robot movement was implemented using linear interpolation for smooth transitions in both translation and rotation. In addition, a simplified simulation, without visual representation, was implemented in Python for the distance-based movement tests, replicating the logic of the Unity environment in a lightweight form.

For real-world experiments, the TurtleBot3 was controlled via Python scripts that sent motor commands. As in the simulation, socket communication was used to interface with the Python code handling the LLM interactions. However, in the real-world setup, the robot's position was not derived from its internal sensors but from an external tracking system. Specifically, a Qualisys infrared camera-based motion capture system was employed. Tracking markers were attached to the TurtleBot3, and the area was calibrated using the Qualisys Track Manager software, allowing for precise real-time location data. The code requesting positional data ran on the same machine as the LLM interface.

0.6 Evaluation

This chapter presents the evaluation of LLMs in robotic navigation tasks. It begins with the results of the exploratory experiments that formed the structure of the subsequent quantitative tests, which were conducted to obtain more consistent, data-driven insights into model performance across various navigation challenges.

0.6.1 Exploratory Results

A series of exploratory experiments were conducted to refine the prompt design, output formats, and navigation strategies before quantitative testing. Multiple LLMs were evaluated, including ChatGPT, Gemini, DeepSeek-R1, and LLaMA. Local models (DeepSeek and LLaMA) struggled due to limited resources, with DeepSeek via OpenRouter showing promise but suffering from excessive latency. Consequently, later tests focused on ChatGPT and Gemini.

Two output styles were compared: command-only and reasoning-based. Without reasoning, LLMs performed poorly, often misinterpreting directions and failing to navigate or avoid obstacles. When allowed to show reasoning, performance improved significantly. Among orientation conventions, switching from Unity's to the standard mathematical system reduced errors, particularly in calculating angles from movement. However, systematic 180 degree errors persisted until prompt reminders about quadrant corrections were added.

For turning, both relative and absolute methods worked well with reasoning, though absolute turning was more reliable. Signed degree values slightly outperformed other relative formats. Obstacles posed persistent challenges, LLMs could only handle one at a time and failed to use path history or adapt strategies, often getting stuck in loops. In distance-based movement, inconsistent behavior led to circling or direction reversals, and while improved prompting reduced some errors, full resolution was not achieved.

0.6.2 Methodology for Quantitative Tests

The quantitative tests focused exclusively on GPT-4o and Gemini-2.0-Flash, with both models operating at a temperature and top-p value of 0.1 to ensure consistent yet flexible behavior. The tests covered the same four core scenarios explored during experimentation: orientation calculation, grid-based movement, free movement, and distance-based movement. All tests allowed reasoning in the LLMs' responses, as earlier experiments demonstrated that this led to significantly better outcomes. Each scenario included multiple test cases to capture a range of navigation setups, with and without obstacles. To evaluate consistency, all tests, except orientation tests, were repeated three times per model. Each tests was done in a separate

conversation with the LLMs to provide a consistent knowledge baseline. Performance metrics included overall and per-case success rates, number of moves taken, and final distance to the destination in free movement scenarios.

0.6.3 Scenarios and Results

Orientation. Orientation calculation was tested within both the grid-based and free movement scenarios, under conditions where the LLMs were either required to show their calculation steps or not. In the grid-based setup, 10 test cases were evaluated with step-by-step calculations included. Both LLMs achieved a 100% success rate. In an additional three test cases without visible calculations, both models again succeeded. This indicates high reliability in grid scenarios. In free movement, performance slightly declined. With the formula shown, ChatGPT succeeded in 7 out of 10 test cases (70%), while Gemini succeeded in 9 (90%). Notably, ChatGPT's failures were not due to incorrect calculations but rather from assuming an orientation instead of performing the required forward move. Gemini's single failure was also procedural rather than computational. When the formula was omitted, ChatGPT succeeded in all three test cases, whereas Gemini failed one, rounding the orientation to the nearest multiple of 45°. These results suggest both models are capable of correctly computing orientations, but consistently better results are obtained when prompting them to explicitly show their calculation steps.

Grid-Based Movement. The grid-based movement scenario was tested under varying conditions: with and without obstacles, and using two turning strategies: relative (positive/negative degrees) and absolute (explicit direction in degrees). Test complexity ranged from simple straight-line movements to routes requiring one or more turns and reorientations.

In obstacle-free settings, there were 10 test cases per turning strategy. For both relative and absolute turning, each LLM successfully completed all test cases in at least one of the three attempts, demonstrating general capability for basic navigation. Occasional failures did occur, ChatGPT due to output formatting issues, and Gemini due to stopping prematurely. However, both models struggled with path optimality. When the destination was not reachable via a direct line, they tended to travel along separate x and y axes or overshoot diagonally. Interestingly, when using absolute turning, both LLMs often ignored the instruction to report orientation before issuing the first move, highlighting sensitivity to minor prompt variations.

Obstacle inclusion significantly reduced success rates. While both LLMs could navigate around a single obstacle, the introduction of multiple obstacles frequently led to infinite loops, repeated path attempts, or hallucinated locations. ChatGPT handled obstacles slightly better overall, whereas Gemini more often generated implausible or incorrect outputs. These results confirm that although LLMs can effectively manage grid-based movement in open environments, complex spatial reasoning with dynamic obstacle avoidance remains a considerable challenge, pointing to limitations in the models' working memory and their lack of an internal state representation or backtracking mechanism.

Free Movement. This scenario evaluated LLM performance in a free-movement setting, both in simulation and with a real TurtleBot3 robot, where movement was continuous rather than grid-based. Tests were performed with both relative turning and absolute turning. The movement in the simulation was perfectly accurate, while the movement in the real world was influenced by the errors of the TurtleBot3, meaning the robot only had to get within half a meter of the destination.

In simulation, both ChatGPT and Gemini achieved high success rates in basic navigation tasks without obstacles, although small issues like hallucinations, inefficient movements, and misinterpretations of orientation (especially with relative turning) were observed. ChatGPT generally performed more reliably and efficiently, while Gemini occasionally made 180° errors in orientation and exhibited less stable behavior after obstacle encounters. Obstacle navigation remained

a significant challenge for both models, particularly in more complex configurations (e.g., walls or U-shapes), where both LLMs failed consistently. The increased complexity of the movement also made the single-obstacle scenario more challenging, as the LLMs frequently failed to steer the robot far enough to the side to fully bypass the obstacle, resulting in repeated collisions. However, this did not necessarily prevent the LLMs from reaching the destination.

Real-world tests showed similar patterns, with ChatGPT showing higher consistency and Gemini repeating previous orientation errors. Additionally, both LLMs attempted to further reduce the distance to the destination, despite the robot already being within the required half-meter threshold, increasing movement time. Overall, while LLMs handled simple free movement navigation well, they struggled with consistent instruction-following and avoiding obstacles.

Distance-Based movement. The distance-based movement scenario tested the LLMs' ability to navigate unknown environments using only Euclidean distance to the destination as feedback. Without access to the orientation, only relative turning was used. In a simple setup, both LLMs managed to reach the destination in all attempts. Gemini consistently outperformed ChatGPT by reliably identifying a correct path with minimal variation. ChatGPT, however, showed inconsistent behavior and was occasionally inefficient or repetitive, but did demonstrate signs of higher-level spatial reasoning in one complex case, where it adapted its path based on detecting parallel movement relative to the destination. In a more challenging scenario requiring multi-step planning, Gemini again showed solid consistency across all trials, albeit with slightly longer paths. ChatGPT failed in two out of three attempts but had the most efficient successful run, indicating potential for deeper understanding when successful. Overall, Gemini was more reliable, while ChatGPT showed greater, though inconsistent, reasoning depth. Both models' strategies were heavily influenced by the prompt design, with only limited evidence of autonomous spatial inference.

Across all tests and experiments, ChatGPT showed better performance but Gemini had a much shorter response time, forcing a sleep to be added in the code to not run into the request limit. This is mainly because Gemini-2.0-Flash was used, which is a version designed for speed instead of reasoning power, while GPT-4o is a more balanced model.

0.7 Conclusion & Future Work

0.7.1 Conclusion

This thesis set out to evaluate whether large language models (LLMs) can autonomously navigate mobile robots with minimal external support. The investigation followed an iterative approach, beginning with exploratory experimentation to refine prompting strategies, followed by quantitative testing to assess consistency and performance in both simulated and real-world settings. The results indicate that LLMs like GPT-4o, Gemini-2.0-Flash, and DeepSeek-V3 are capable of interpreting commands, reasoning through navigation tasks, and executing movement plans using step-by-step interaction with real-time feedback. However, their success depends heavily on clear task prompts and structured environments. All models struggled with obstacle handling, often repeating failed strategies, forgetting previous paths, and becoming stuck in loops. While DeepSeek-R1 demonstrated relatively strong reasoning, its long response times made it impractical for real-time use. A key finding was the importance of open reasoning. When models were allowed to explain their thinking, performance improved notably. Still, serious challenges remained: models failed to track orientation consistently, misinterpreted turning commands, and struggled with pathfinding and distance-based reasoning. Their inability to generate accurate spatial representations or adapt heuristics also limited their effectiveness. Inconsistent output, even under identical conditions, further undermined reliability. In conclusion, while LLMs show promise for low-support navigation in simple environments, they are not yet dependable for general-purpose robotic autonomy. To be viable for real-world deployment, they require external support, such as memory aids, abstraction layers, or more advanced

reasoning modules, to overcome their current limitations.

0.7.2 Limitations and Future Work

This thesis primarily focused on GPT-4o and Gemini-2.0-Flash, with limited trials using DeepSeek-R1 and LLaMA3.1, leaving many strong alternatives like Claude, Mistral and Grok unexplored. Broader benchmarking across models could offer clearer insights as capabilities evolve. A major limitation was the short-horizon nature of tasks, which do not reflect the complexity of real-world applications where long-term memory, token limits, and state tracking become more critical. Future work should explore extended navigation scenarios using conversation management or persistent memory mechanisms. Improving obstacle handling and distance-based navigation is also essential; while this study minimized external aids to evaluate raw LLM capability, adding lightweight support such as sensor inputs, verification models, or simple memory could significantly boost performance, especially in dynamic or complex environments. A recurring issue was the models' inconsistent adherence to output formatting, often leading to execution errors despite sound reasoning. Adopting structured formats like JSON could improve reliability and integration with control systems, though care must be taken not to overly constrain model behavior. Finally, future research could expand beyond navigation into robotic manipulation using grid-based representations to explore simple tasks like grasping and object placement, while assessing how such models perform in less controlled, real-world conditions.

0.7.3 Reflection

Over the course of this thesis, I developed a clearer understanding of the limitations of LLMs in domains like robotic navigation and spatial reasoning. While their conversational fluency initially impressed me, their failure to follow explicit spatial instructions revealed that their reasoning is often shallow and unreliable in unfamiliar contexts. A major turning point in the work was the realization that allowing models to include reasoning and intermediate calculations in their responses led to significantly better outcomes. Unfortunately, this insight came relatively late in the process, and a considerable amount of time was spent attempting to improve performance under restricted output settings. Another misstep was relying too heavily on iterative prompt tuning with only a small number of models. This project also underscored the critical importance of maintaining structured, detailed records throughout all stages of experimentation. In early phases, testing was highly exploratory, and results were often logged only in brief notes or informal summaries. This lack of structured documentation made it difficult to trace errors, repeat tests, or draw precise conclusions later during analysis. As the project progressed, more rigorous data collection practices helped reduce these issues, but the earlier gaps inevitably slowed progress and required certain experiments to be redone.

Samenvatting

0.8 Inleiding

De automatisering van manuele taken is sinds het midden van de twintigste eeuw sterk geëvolueerd, waarbij robotica een sleutelrol speelt in sectoren zoals productie, transport en huishoudelijke taken. Traditioneel was de aansturing van robots gebaseerd op regelgebaseerde en hardgecodeerde logica. In het afgelopen decennium is er echter een verschuiving opgetreden richting AI-gedreven automatisering, met het gebruik van Large Language Models (LLM's), Vision-Language Models (VLM's) en andere gespecialiseerde AI-tools. Deze modellen hebben gezorgd voor meer adaptief en intelligent gedrag in systemen zoals zelfrijdende auto's en robots zoals Boston Dynamics' "Spot".

Ondanks deze vooruitgang blijft het programmeren van robots een complexe uitdaging, vooral voor taken die redenering, ruimtelijk inzicht en planning vereisen. Hoewel LLM's en VLM's nieuwe mogelijkheden bieden door menselijk redeneren na te bootsen en gebruiksvriendelijker te zijn, zijn ze in eerste instantie ontworpen voor conversatietoepassingen, niet voor robotbesturing. Hun integratie in robotica vereist dan ook het overwinnen van beperkingen op het gebied van planning en aansturing.

Verschillende eerdere studies hebben LLM's geïntegreerd in robotsystemen, maar doen daarbij meestal een beroep op aanvullende modellen of speciaal ontwikkelde frameworks om het taalmodel te ondersteunen of te beperken. Voorbeelden hiervan zijn het gebruik van aangeleerde value functions om robotvaardigheden gegenereerd door een LLM te filteren en te selecteren, het verbeteren van het geheugen van LLM's om hun leerbaarheid via menselijke feedback te vergroten, en het inzetten van LLM's voor het definiëren van reward parameters zodat een robot zijn gedrag kan optimaliseren door acties te kiezen die aan die doelen voldoen. Hoewel al deze benaderingen veelbelovende resultaten opleveren, zijn ze sterk afhankelijk van uitgebreide ondersteunende infrastructuur om het LLM te begeleiden of te corrigeren. Dit roept de vraag op of de recente vooruitgang in LLM's voldoende is om effectieve robotbesturing mogelijk te maken zonder zulke bijkomende componenten.

De centrale onderzoeksvraag luidt: "**Kunnen grote taalmodellen een mobiele robot door een gebied navigeren met een minimaal ondersteunend framework?**" Het doel is natuurlijke taalinteractie met robots, waarbij een LLM gebruikersinstructies direct interpreteert en uitvoert. Een belangrijke techniek die hierbij gebruikt wordt, is prompt engineering, die helpt om de reacties van het LLM te sturen. Het raamwerk wordt bewust minimaal gehouden, waarbij de meeste logica en controle wordt overgelaten aan het LLM, aangevuld met slechts eenvoudige feedback uit de omgeving van de robot.

0.9 Gerelateerd Werk

Large Language Models (LLM's) zijn krachtige maar weinig transparante systemen, waarvan het volledige gedrag nog steeds niet goed wordt begrepen. Hun algemene toepasbaarheid maakt ze waardevol in uiteenlopende domeinen, waaronder robotica. Tegelijk brengen ze ook aanzien-

lijke beperkingen en risico's met zich mee, vooral in toepassingen met hoge risico's. Het is daarom essentieel om hun tekortkomingen te begrijpen voordat ze in reële situaties worden ingezet. LLM's "denken" of "redeneren" niet zoals mensen; hun output is gebaseerd op probabilistische patronen in tekst. Dit kan leiden tot hallucinaties: ogenschijnlijk plausibele maar feitelijk onjuiste uitingen, wat veiligheidsrisico's met zich mee kan brengen in robotica. Verschillende studies hebben methoden voorgesteld om dergelijke hallucinaties te detecteren en te beperken. Daarnaast zijn LLM's zeer gevoelig voor variaties in prompts, waarbij kleine wijzigingen in de input vaak leiden tot sterk afwijkende resultaten. Andere zorgen omvatten de hoge rekentijd, de milieu-impact en kwetsbaarheden op het gebied van beveiliging en privacy. Mens-Robot Interactie (Human-Robot Interaction, HRI) is een cruciaal aspect van robotica, waarin voorspelbaarheid en transparantie belangrijk zijn voor effectieve samenwerking. LLM's kunnen hierin positief bijdragen dankzij hun mensachtige taalbegrip. Eerder onderzoek heeft hun inzet verkend bij het simuleren van onderdelen van systemen in HRI-studies en bij het modelleren van menselijk gedrag.

Deze thesis richt zich op het gebruik van LLM's voor robotnavigatie, een gebied dat eerder is onderzocht in zowel bekende als onbekende omgevingen. In bekende omgevingen hebben LLM's prestaties bereikt die vergelijkbaar zijn met traditionele planners zoals A*, maar met een hogere verwerkingssnelheid. In onbekende omgevingen zijn twee hoofdbenaderingen onderzocht: continue mapping tijdens verkenning en het opbouwen van herbruikbare kaarten. Deze systemen maken vaak gebruik van Visual Language Models (VLM's) die camerabeelden en sensorinput verwerken. In tegenstelling hiermee gebruikt deze thesis een minimaal framework en een tekst-only LLM om de navigatiemogelijkheden in isolatie te beoordelen. HRI blijft relevant binnen navigatietaken, zowel in de interactie met de operator als met de omgeving. Er zijn doeltreffende interfaces nodig voor communicatie tussen de operator en het LLM, en robots moeten zich veilig kunnen bewegen in de nabijheid van mensen. Enkele studies hebben onderzocht hoe LLM's kunnen redeneren over gedrag van voetgangers en sociale interacties om robotacties te sturen.

Een belangrijke uitdaging in LLM-gebaseerde robotica is langetermijnplanning van taken, waarbij modellen hun tokenlimiet kunnen overschrijden of eerdere instructies vergeten. Voorgestelde oplossingen hiervoor zijn onder andere het opsplitsen van taken in subtaken, het gebruik van gestructureerde geheugensystemen om het aantal te onthouden elementen te beperken, het toevoegen van omgevingsgrafen om informatie op te slaan, en het laten opnemen van contextuele details in de output van het LLM zelf. Hoewel deze thesis zich hoofdzakelijk richt op kortetermijntaken, kwamen verwante geheugenproblemen toch naar voren en werden deze deels opgevangen met herhaalde promptherinneringen en extra contextuele input.

Tot slot blijkt prompt engineering, het optimaliseren van instructies aan LLM's, de meest effectieve methode om hun prestaties te verbeteren. Eerdere studies hebben strategieën geïdentificeerd zoals gestructureerde input-outputformaten, toekenning van persona's, few-shot prompting, toevoegen van contextuele informatie en chain-of-thought redeneren. Deze technieken zijn uitgebreid toegepast in dit werk om de navigatieresultaten te verbeteren.

0.10 Initiële Haalbaarheidstesten

Om te beginnen werd een reeks haalbaarheidstesten uitgevoerd met GPT-4o (OpenAI), Gemini-2.0-Flash (Google DeepMind), DeepSeek-V3 en DeepSeek-R1, enkele van de meest geavanceerde publiek toegankelijke LLM's. Er werd gekozen voor een vereenvoudigde omgeving, een gelabeld schaakbordraster, om het ruimtelijk begrip van de modellen in een gecontroleerde context te testen. Deze opzet bood een discrete en gemakkelijk te interpreteren ruimte waarin de modellen konden redeneren over navigatie, beweging en obstakels.

In de testen kon een robot, die een mobiele agent voorstelde, zich telkens één vakje verplaatsen in eender welke richting, met de bijkomende vereiste dat hij eerst in de juiste richting moest draaien alvorens vooruit te bewegen. Obstakels op het raster verhoogden de complexiteit,

aangezien de modellen hierdoor ongeldige paden moesten vermijden. Deze experimenten boden vroege inzichten in de sterktes en beperkingen van elk model wat betreft het redeneren over beweging en ruimte, en hielpen bepalen of LLM's zelfstandig de basis kunnen vormen voor robotbesturing.

0.10.1 Testen en Resultaten

Om de navigatiemogelijkheden van LLM's te evalueren, werden drie soorten testen ontworpen: het uitvoeren van enkele bewegingen per instructie, het plannen van een volledig pad in één respons, en interactieve stap-voor-stap navigatie met real-time feedback. Elke test werd uitgevoerd in zowel obstakelvrije als obstakelrijke omgevingen, waarbij obstakels ofwel bekend waren (expliciet beschreven) of onbekend (afgeleid uit mislukte bewegingen). LLM's kregen ook de opdracht ASCII-gebaseerde visualisaties van de omgeving te genereren voor duidelijkheid en traceerbaarheid.

De resultaten toonden aan dat de huidige LLM's, GPT-4o, Gemini-2.0-Flash en DeepSeek-V3, in staat zijn tot basisvormen van ruimtelijk redeneren en eenvoudige navigatietaken aankunnen, mits duidelijke instructies. Ze presteerden redelijk goed bij het uitvoeren van directe commando's en konden obstakellocaties onthouden in begeleide scenario's. Hun consistentie nam echter af in complexere omgevingen, vooral bij het plannen van volledige paden, waar soms botsingen of herhalende lussen optraden. Diagonale bewegingen werden vaak genegeerd ten voordele van eenvoudigere orthogonale bewegingen. Stap-voor-stap navigatie, waarbij de modellen feedback kregen na elke zet, leverde betere resultaten op dan het in één keer genereren van een volledig pad. Dit stelde de LLM's in staat zich aan te passen aan onverwachte obstakels en hun strategieën bij te sturen. Niettemin bleven hallucinaties en logische inconsistenties optreden, vooral in dubbelzinnige situaties. Van de geteste modellen vertoonde DeepSeek-R1 de sterkste prestaties qua redenering en obstakelverwerking, maar kampte met extreem lange responstijden, wat de bruikbaarheid inperkte. DeepSeek-V3 was sneller, maar bleef achter op ChatGPT en Gemini, die gemiddeld de snelste reacties gaven. Visuele representaties bleken een zwakker punt: de meeste modellen hadden moeite met het consequent accuraat weergeven van ASCII visualisaties van de robot en zijn omgeving.

0.11 Concept

De haalbaarheidstesten toonden aan dat LLM's over de fundamentele capaciteiten voor navigatie beschikken, maar verdere experimenten waren nodig in meer gecontroleerde omgevingen met verbeterde promptingstrategieën. Alle interacties met de LLM's gebeurden via tekstinvoer, met als primaire modellen GPT-4o en Gemini-2.0-Flash, beide benaderd via hun officiële API's. Aanvullende verkennende testen werden uitgevoerd met DeepSeek-R1 en LLaMA 3.1, via externe API's of via lokale implementatie. Het gebruikte robotplatform was de "TurtleBot3 Waffle Pi", die in staat is tot basale bewegingscommando's: vooruit, achteruit, en rotatie op de plaats met de klok mee en tegen de klok in. Experimenten werden uitgevoerd in Python- en Unity-gebaseerde simulaties en in fysieke omgevingen. De robot bewoog zich in een vlakke tweedimensionale ruimte, waarbij zijn positie werd weergegeven met Cartesische coördinaten en zijn oriëntatie in graden, volgens de conventie van Unity of de standaard wiskundige notatie. De omgeving kon zowel bekende als onbekende obstakels bevatten. Onbekende obstakels moesten door de LLM's zelf worden afgeleid door er tegenaan te botsen. In simulatie was het bijhouden van de positie van de robot eenvoudig. In de echte wereld werd tracking uitgevoerd met een infrarood motion capture-systeem van Qualisys.

0.11.1 Input en Output

LLM's ontvingen gestructureerde taakprompts met instructies, beschrijvingen van de omgeving, regels voor outputformaat en informatie over obstakelverwerking, foutafhandeling en outputstructuur. Daarnaast was er een korte herinnering die iteratief werd bijgewerkt op basis van de

waargenomen zwaktes van de LLM's. Voor elke navigatietaak kregen de LLM's een beginpunt, bestemming en soms oriëntatie. Ze gaven één beweging per bericht, geholpen door realtime feedback na elk commando. Deze feedback bestond uit de geüpdatete positie van de robot na een vooruit- of achteruitbeweging en een bevestiging bij het draaien. Promptstijlen werden getest met en zonder redenering in de respons, vergelijkbaar met chain-of-thought prompting.

0.11.2 Prompting- en Navigatiestrategieën

Om effectieve promptingtechnieken voor robotnavigatie te evalueren, werden twee strategieën getest voor het omgaan met robotoriëntatie: ofwel de oriëntatie expliciet meegeven aan het begin, ofwel de LLM's deze laten afleiden door het resultaat van een voorwaartse beweging te observeren. Dit laatste sluit aan bij het doel van de thesis om externe ondersteuning te minimaliseren. LLM's werden ook getest met twee outputstijlen: enkel het resultaat retourneren of hun redenering stap voor stap tonen samen met de formule.

Voor draaien werden zowel relatieve als absolute strategieën onderzocht. Relatief draaien vereiste dat de LLM's oriëntatiewijzigingen bijhielden via commando's zoals "draai 90 graden naar links" of "draai -45 graden", met verschillende formuleringen. Absoluut draaien hield in dat een eindoriëntatie werd gespecificeerd, zoals "kijk naar 90 graden", "kijk naar het noorden" of "positieve x", waarbij het controlesysteem dit vertaalde naar een relatieve draai. Richtingsnamen waren effectief in beperkte omgevingen, maar werden onpraktisch bij willekeurige hoeken. Deze testen hielpen bepalen welke formuleringen en strategieën het ruimtelijk redeneren in LLM-gebaseerde navigatie het best ondersteunden.

0.11.3 Experimentele Scenario's

Testen werden uitgevoerd in drie verschillende scenario's, elk gericht op het evalueren van verschillende aspecten van het redeneervermogen van de LLM's. Het eerste scenario was een rastergebaseerde bewegingsomgeving, waarin de robot zich bewoog op een vast raster vergelijkbaar met een schaakbord. Beweging was beperkt tot discrete stappen: 1 meter voor orthogonale richtingen en $\sqrt{2}$ meter voor diagonale. De robot kon draaien in stappen van 45 graden, wat beweging in acht mogelijke richtingen toeliet. Deze opzet vormde een vereenvoudeld testplatform om te evalueren of LLM's basaal ruimtelijk redeneren konden uitvoeren en een robot dienovereenkomstig konden besturen. Alle experimenten in dit scenario werden uitgevoerd in een Unity-simulatie, aangezien de echte TurtleBot3 niet de bewegingsprecisie bezit die nodig is om betrouwbaar aan de rasterbeperkingen te voldoen.

Het tweede scenario betrof een realistischer vorm van navigatie, waarbij de robot niet langer aan een raster gebonden was en in eender welke richting en afstand kon bewegen tussen 0° en 360°. Experimenten werden uitgevoerd zowel in simulatie via Unity als in de echte wereld met de TurtleBot3. In tegenstelling tot de nauwkeurige controle die simulatie bood, had de echte robot te maken met fysieke onnauwkeurigheden zoals het slippen van wielen en motordrift. Daarom werd een tolerantie van 0,5 meter rond het doel toegestaan voor succesvolle navigatie in echte experimenten. In dit scenario werd absoluut draaien uitsluitend uitgedrukt in graden, aangezien richtingsnamen onpraktisch zijn bij continue beweging. Hoewel dit scenario complexere oriëntatieberekeningen introduceerde, was de onderliggende optimale bewegingsstrategie conceptueel eenvoudig: de robot moest naar de bestemming draaien en er rechtstreeks naartoe bewegen.

In veel echte toepassingen is het niet altijd mogelijk om voortdurend de exacte coördinaten van een robot te verkrijgen vanwege sensorbeperkingen of omgevingsfactoren. Om zulke situaties te simuleren, werd een laatste reeks experimenten uitgevoerd waarin de LLM's niet langer exacte locatiegegevens kregen. In plaats daarvan ontvingen ze enkel de huidige afstand tot het eindpunt, wat hen uitdaagde om te navigeren met aanzienlijk minder informatie. Deze experimenten werden uitgevoerd met rastergebaseerde beweging in een Python-simulatie om de taak beheersbaar te houden met behoud van structuur. De onderliggende strategie in dit scenario was om bewegingen in verschillende richtingen te proberen en te observeren of de

afstand tot het doel verminderde. Hoewel deze taak eenvoudig opgelost kan worden met een trial-and-error algoritme, was het doel om te evalueren of LLM's ruimtelijk redeneren konden vertonen om sneller het juiste pad te vinden. Obstakels werden hier niet getest, aangezien detectie moeilijk zou zijn en in andere scenario's slechte resultaten opleverde.

0.12 Implementatie

De parameters temperatuur en top-p voor zowel ChatGPT als Gemini werden ingesteld op 0,1 om meer deterministische output te garanderen, terwijl er genoeg variabiliteit bleef om foutcorrectie mogelijk te maken. De communicatie met deze modellen verliep via hun officiële API's met Python-code. LLaMA 3.1 en DeepSeek-R1 werden lokaal uitgevoerd met het programma Ollama, terwijl DeepSeek-R1 ook toegankelijk was via een derdepartij API-provider genaamd OpenRouter, ook geïntegreerd met Python.

De primaire simulatieomgeving werd gebouwd in Unity, waar een virtueel model van de TurtleBot3 navigeerde binnen een gemodelleerde kamer. Deze opzet bestond uit twee hoofdbestanddelen: een module voor het aansturen van de beweging van de robot en een andere voor het opzetten van een socketverbinding met de Python-backend die de interface met de LLM's verzorgde. De robotbeweging werd geïmplementeerd met lineaire interpolatie voor vloeiende overgangen in zowel translatie als rotatie. Daarnaast werd een vereenvoudigde simulatie zonder visuele weergave uitgevoerd in Python voor de afstand-gebaseerde bewegingstests, waarbij de logica van de Unity-omgeving lichtgewicht werd nagebootst.

Voor de experimenten in de echte wereld werd de TurtleBot3 aangestuurd via Python-scripts die motorcommando's verstuurden. Net als in de simulatie werd socketcommunicatie gebruikt om te koppelen met de Python-code die de interacties met de LLM's verzorgde. In de echte wereld kwam de positiebepaling van de robot niet voort uit de interne sensoren, maar uit een extern volgsysteem. Hiervoor werd een infrarood motion capture-systeem van Qualisys ingezet. Er werden trackingmarkers bevestigd aan de TurtleBot3 en het gebied werd gekalibreerd met de Qualisys Track Manager-software, wat precieze locatiegegevens in realtime mogelijk maakte. De code die de positiegegevens opvroeg draaide op dezelfde machine als de interface met de LLM.

0.13 Evaluatie

Dit hoofdstuk presenteert de evaluatie van LLM's in robotnavigatietaken. Het begint met de resultaten van de verkennende experimenten die de basis vormden voor de daaropvolgende kwantitatieve tests, die werden uitgevoerd om meer consistente, data-gedreven inzichten te verkrijgen in de prestaties van de modellen bij verschillende navigatie-uitdagingen.

0.13.1 Verkennende Resultaten

Er werden een reeks verkennende experimenten uitgevoerd om het ontwerp van de prompts, outputformaten en navigatiestrategieën te verfijnen vóór de kwantitatieve tests. Meerdere LLM's werden geëvalueerd, waaronder ChatGPT, Gemini, DeepSeek-R1 en LLaMA. Lokale modellen (DeepSeek en LLaMA) hadden moeite vanwege beperkte middelen, waarbij DeepSeek via OpenRouter veelbelovend was maar last had van te hoge latentie. Daarom richtten latere tests zich op ChatGPT en Gemini.

Twee outputstijlen werden vergeleken: alleen commando's en redeneervorm. Zonder redeneervorm presteerden de LLM's slecht, ze interpreteerden richtingen vaak verkeerd en faalden in navigatie of het vermijden van obstakels. Bij toestemming om redeneringen te tonen, verbeterde de prestatie aanzienlijk. Van de oriëntatieconventies verminderde de overgang van Unity's naar het standaard wiskundige systeem fouten, vooral bij het berekenen van hoeken vanuit beweging.

Systematische fouten van 180 graden bleven echter bestaan totdat prompttherinneringen over kwadrantcorrecties werden toegevoegd.

Voor draaien werkten zowel relatieve als absolute methoden goed met redeneervorm, hoewel absoluut draaien betrouwbaarder was. Ondertekende gradenwaarden presteerden iets beter dan andere relatieve formaten. Obstakels bleven een blijvende uitdaging, LLM's konden er maar één tegelijk aan, gebruikten geen padgeschiedenis en pasten strategieën niet aan, waardoor ze vaak vastliepen in lussen. Bij afstand-gebaseerde beweging leidde inconsistent gedrag tot cirkelen of richtingsomkering, en hoewel verbeterde prompting sommige fouten verminderde, werd volledige oplossing niet bereikt.

0.13.2 Methodologie voor Kwantitatieve Tests

De kwantitatieve tests richtten zich exclusief op GPT-4o en Gemini-2.0-Flash, waarbij beide modellen draaiden met een temperatuur en top-p waarde van 0,1 om consistente maar flexibele prestaties te garanderen. De tests behandelden dezelfde vier kernthema's die tijdens de experimenten werden onderzocht: oriëntatieberekening, grid-gebaseerde beweging, vrije beweging en afstand-gebaseerde beweging. In alle tests werd redeneervorm in de antwoorden van de LLM's toegestaan, aangezien eerdere experimenten aantoonden dat dit leidde tot significant betere resultaten. Elk scenario omvatte meerdere testgevallen om een reeks navigatieopstellingen te bestrijken, met en zonder obstakels. Om consistentie te evalueren werden alle tests, behalve oriëntatietests, driemaal per model herhaald. Elke test werd uitgevoerd in een aparte conversatie met de LLM's om een consistente kennisbasis te waarborgen. Prestatiemaatstaven omvatten de algemene en per-geval succespercentages, aantal gemaakte bewegingen en de uiteindelijke afstand tot de bestemming in scenario's met vrije beweging.

0.13.3 Scenario's en Resultaten

Oriëntatie. Oriëntatieberekening werd getest binnen zowel de grid-gebaseerde als de vrije bewegingsscenario's, onder de voorwaarden dat de LLM's ofwel hun berekeningsstappen moesten tonen, of niet. In de grid-gebaseerde opzet werden 10 testgevallen geëvalueerd met stap-voorstap berekeningen inbegrepen. Beide LLM's behaalden een succespercentage van 100%. In nog eens drie testgevallen zonder zichtbare berekeningen slaagden beide modellen opnieuw. Dit duidt op een hoge betrouwbaarheid in grid-scenario's. Bij vrije beweging daalde de prestatie iets. Met getoonde formule slaagde ChatGPT in 7 van de 10 testgevallen (70%), terwijl Gemini in 9 (90%) slaagde. Opvallend was dat de fouten van ChatGPT niet voortkwamen uit onjuiste berekeningen, maar uit het aannemen van een oriëntatie in plaats van de vereiste voorwaartse beweging uit te voeren. De ene fout van Gemini was ook procedureel in plaats van computationeel. Wanneer de formule werd weggelaten, slaagde ChatGPT in alle drie de testgevallen, terwijl Gemini er één faalde, waarbij de oriëntatie naar de dichtstbijzijnde veelvoud van 45° werd afgerond. Deze resultaten suggereren dat beide modellen in staat zijn oriëntaties correct te berekenen, maar dat consequent betere resultaten worden behaald wanneer ze expliciet worden aangespoord hun berekeningsstappen te tonen.

Grid-gebaseerde Beweging. Het grid-gebaseerde bewegingsscenario werd getest onder wisselende condities: met en zonder obstakels, en met twee draai-strategieën: relatief (positieve/negatieve graden) en absoluut (expliciete richting in graden). De testcomplexiteit varieerde van eenvoudige rechte bewegingen tot routes die één of meerdere bochten en heroriëntaties vereisten.

In obstakelvrije omgevingen waren er 10 testgevallen per draai-strategie. Voor zowel relatief als absoluut draaien voltooide elk van de LLM's alle testgevallen met succes in ten minste één van de drie pogingen, wat de algemene bekwaamheid voor basale navigatie aantoont. Af en toe traden fouten op, ChatGPT vanwege problemen met outputformattering, en Gemini doordat het voortijdig stopte. Beide modellen hadden echter moeite met padoptimaliteit. Wanneer de bestemming niet via een rechte lijn bereikbaar was, neigden ze ertoe langs afzonderlijke x- en

y-assen te reizen of diagonaal te overschieten. Interessant was dat bij absoluut draaien beide LLM's vaak de instructie negeerden om de oriëntatie te rapporteren vóór de eerste beweging, wat hun gevoeligheid voor kleine promptvariaties illustreert.

Het toevoegen van obstakels verminderde de succespercentages aanzienlijk. Hoewel beide LLM's rond een enkel obstakel konden navigeren, leidde de introductie van meerdere obstakels vaak tot oneindige lussen, herhaalde paden of gefantaseerde locaties. ChatGPT ging over het algemeen iets beter om met obstakels, terwijl Gemini vaker onwaarschijnlijke of incorrecte outputs genereerde. Deze resultaten bevestigen dat hoewel LLM's grid-gebaseerde beweging in open omgevingen effectief kunnen beheren, complexe ruimtelijke redenering met dynamische obstakelvermijding een aanzienlijke uitdaging blijft, wat wijst op beperkingen in het werkgeheugen van de modellen en hun gebrek aan een interne statusrepresentatie of terugspoelmechanisme.

Vrije Beweging. Dit scenario evalueerde de prestaties van LLM's in een vrije bewegingsomgeving, zowel in simulatie als met een echte TurtleBot3-robot, waarbij de beweging continu was in plaats van grid-gebaseerd. Tests werden uitgevoerd met zowel relatief als absoluut draaien. De beweging in de simulatie was perfect nauwkeurig, terwijl de beweging in de echte wereld beïnvloed werd door fouten van de TurtleBot3, waardoor de robot slechts binnen een halve meter van de bestemming hoefde te komen.

In simulatie behaalden zowel ChatGPT als Gemini hoge succespercentages bij basale navigatietaken zonder obstakels, hoewel kleine problemen zoals hallucinaties, inefficiënte bewegingen en misinterpretaties van oriëntatie (vooral bij relatief draaien) werden waargenomen. ChatGPT presteerde over het algemeen betrouwbaarder en efficiënter, terwijl Gemini af en toe 180° fouten in oriëntatie maakte en minder stabiel gedrag vertoonde na obstakelontmoetingen. Obstakelnavigatie bleef een grote uitdaging voor beide modellen, vooral in complexere configuraties (bijvoorbeeld muren of U-vormen), waar beide LLM's consequent faalden. De toegenomen complexiteit van de beweging maakte ook het scenario met één obstakel uitdagender, omdat de LLM's er vaak niet in slaagden de robot ver genoeg naar de zijkant te sturen om het obstakel volledig te passeren, wat resulteerde in herhaalde botsingen. Dit weerhield de LLM's er echter niet noodzakelijk van om de bestemming te bereiken.

Tests in de echte wereld toonden vergelijkbare patronen, waarbij ChatGPT meer consistentie liet zien en Gemini eerdere oriëntatiefouten herhaalde. Daarnaast probeerden beide LLM's de afstand tot de bestemming verder te verkleinen, ondanks dat de robot al binnen de vereiste halve meter was, wat de bewegingstijd verlengde. Over het geheel genomen konden LLM's eenvoudige vrije bewegingsnavigatie goed afhandelen, maar ze worstelden met het consequent opvolgen van instructies en het vermijden van obstakels.

Afstandsgebaseerde Beweging. Het afstandsgebaseerde bewegingsscenario testte het vermogen van LLM's om onbekende omgevingen te navigeren met alleen Euclidische afstand tot de bestemming als feedback. Zonder toegang tot oriëntatie werd alleen relatief draaien gebruikt. In een eenvoudige opzet slaagden beide LLM's erin de bestemming in alle pogingen te bereiken. Gemini presteerde consequent beter dan ChatGPT door betrouwbaar een correct pad met minimale variatie te identificeren. ChatGPT vertoonde echter inconsistent gedrag en was af en toe inefficiënt of repetitief, maar toonde wel tekenen van hoger niveau ruimtelijk redeneren in één complex geval, waarin het zijn pad aanpaste op basis van het detecteren van parallelle beweging ten opzichte van de bestemming. In een uitdagender scenario dat meerstapsplanning vereiste, toonde Gemini opnieuw solide consistentie in alle pogingen, zij het met iets langere paden. ChatGPT faalde in twee van de drie pogingen, maar had de meest efficiënte succesvolle uitvoering, wat duidt op potentieel voor diepgaander begrip wanneer het slaagt. Over het geheel genomen was Gemini betrouwbaarder, terwijl ChatGPT een grotere, zij het inconsistente, diepgang in redeneren liet zien. De strategieën van beide modellen werden sterk beïnvloed door het promptontwerp, met slechts beperkte aanwijzingen voor autonome ruimtelijke inferentie.

In alle tests en experimenten vertoonde ChatGPT betere prestaties, maar Gemini had een veel kortere responstijd, waardoor een pauze in de code moest worden ingebouwd om niet tegen het verzoeklimiet aan te lopen. Dit komt vooral doordat Gemini-2.0-Flash werd gebruikt, een versie die is ontworpen voor snelheid in plaats van redeneerkracht, terwijl GPT-4o een meer gebalanceerd model is.

0.14 Conclusie & Toekomstig Werk

0.14.1 Conclusie

Deze thesis had als doel om te onderzoeken of grote taalmodellen (LLM's) zelfstandig mobiele robots kunnen navigeren met minimale externe ondersteuning. Het onderzoek volgde een iteratieve aanpak, beginnend met verkennende experimenten om de prompting-strategieën te verfijnen, gevolgd door kwantitatieve tests om consistentie en prestaties te beoordelen in zowel gesimuleerde als echte omgevingen. De resultaten geven aan dat LLM's zoals GPT-4o, Gemini-2.0-Flash en DeepSeek-V3 in staat zijn om commando's te interpreteren, te redeneren over navigatietaken en bewegingsplannen uit te voeren via stapsgewijze interactie met real-time feedback. Hun succes is echter sterk afhankelijk van heldere taakbeschrijvingen en gestructureerde omgevingen. Alle modellen hadden moeite met het omgaan met obstakels, waarbij ze vaak mislukte strategieën herhaalden, eerdere routes vergaten en vastliepen in loops. Hoewel DeepSeek-R1 relatief sterke redeneerprestaties liet zien, maakten de lange reactietijden het onpraktisch voor real-time gebruik. Een belangrijke bevinding was het belang van open redeneren. Wanneer modellen hun denkproces mochten toelichten, verbeterde de prestatie aanzienlijk. Toch bleven er serieuze uitdagingen: modellen faalden in het consequent bijhouden van oriëntatie, interpreteerden draaicommandos verkeerd en hadden moeite met het vinden van paden en afstandsgebaseerde redenering. Hun onvermogen om nauwkeurige ruimtelijke representaties te genereren of heuristieken aan te passen, beperkte ook hun effectiviteit. Inconsistente output, zelfs onder identieke omstandigheden, ondermijnde de betrouwbaarheid verder. Samenvattend tonen LLM's potentie voor navigatie met weinig ondersteuning in eenvoudige omgevingen, maar zijn ze nog niet betrouwbaar genoeg voor algemene robotautonomie. Om bruikbaar te zijn voor praktische toepassingen, hebben ze externe ondersteuning nodig, zoals geheugenhulpmiddelen, abstractielagen of geavanceerdere redeneermodules, om hun huidige beperkingen te overwinnen.

0.14.2 Limitaties en Toekomstig Werk

Deze thesis richtte zich voornamelijk op GPT-4o en Gemini-2.0-Flash, met beperkte tests met DeepSeek-R1 en LLaMA3.1, waardoor veel sterke alternatieven zoals Claude, Mistral en Grok buiten beschouwing bleven. Een bredere benchmark over meerdere modellen zou duidelijkere inzichten kunnen opleveren, zeker naarmate de capaciteiten van modellen verder evolueren. Een belangrijke beperking was de focus op kortetermijntaken, die niet de complexiteit van toepassingen in de echte wereld weerspiegelen, waar zaken als langetermijngeheugen, tokenlimieten en statustracking veel belangrijker worden. Toekomstig onderzoek zou zich moeten richten op uitgebreidere navigatiescenario's, bijvoorbeeld met behulp van gespreksbeheer of mechanismen voor persistent geheugen. Ook het verbeteren van obstakelvermijding en afstandsgebaseerde navigatie is essentieel. Hoewel deze studie externe hulpmiddelen tot een minimum beperkte om de ruwe capaciteiten van LLM's te evalueren, zou het toevoegen van lichte ondersteuning, zoals sensorinvoer, verificatiemodellen of eenvoudige geheugensystemen, de prestaties aanzienlijk kunnen verbeteren, vooral in dynamische of complexe omgevingen. Een terugkerend probleem was de inconsistente naleving van outputformaten door de modellen, wat vaak leidde tot uitvoeringsfouten ondanks correcte redenering. Het gebruik van gestructureerde formaten zoals JSON zou de betrouwbaarheid en integratie met besturingssystemen kunnen verbeteren, al moet daarbij worden opgelet dat het gedrag van het model niet te veel wordt beperkt. Tot slot zou toekomstig onderzoek zich kunnen uitbreiden naar robotmanipulatie, door gebruik te maken van grid-gebaseerde representaties om eenvoudige taken zoals grijpen en objectplaats-

ing te verkennen. Hierbij kan dan worden beoordeeld hoe zulke modellen presteren in minder gecontroleerde, realistische omstandigheden.

0.14.3 Reflectie

Tijdens het verloop van deze thesis heb ik een beter begrip gekregen van de beperkingen van LLM's in domeinen zoals robotnavigatie en ruimtelijke redenering. Hoewel hun gespreksvaardigheid aanvankelijk indruk maakte, toonde hun falen om expliciete ruimtelijke instructies te volgen aan dat hun redenering vaak oppervlakkig en onbetrouwbaar is in onbekende contexten. Een belangrijk keerpunt in het werk was het besef dat het toestaan van redenering en tussenliggende berekeningen in de antwoorden leidde tot aanzienlijk betere resultaten. Helaas kwam dit inzicht relatief laat in het proces, en werd er veel tijd besteed aan het proberen te verbeteren van prestaties onder beperkter outputgebruik. Een andere misstap was het te zwaar leunen op iteratieve prompt tuning met slechts een klein aantal modellen. Dit project benadrukte ook het cruciale belang van het bijhouden van gestructureerde, gedetailleerde documentatie gedurende alle fasen van het experimenteren. In de vroege fases was het testen sterk verkennend en werden resultaten vaak slechts in korte notities of informele samenvattingen vastgelegd. Dit gebrek aan gestructureerde documentatie maakte het moeilijk om fouten te traceren, tests te herhalen of precieze conclusies te trekken tijdens de analyse. Naarmate het project vorderde, hielpen strengere dataverzamelingspraktijken deze problemen te verminderen, maar de eerdere hiaten vertraagden het proces onvermijdelijk en vereisten het opnieuw uitvoeren van bepaalde experimenten.

Contents

Chapter 1

Introduction

Since the mid-20th century, automation has transformed everyday tasks and entire industries, ranging from factory assembly lines to household chores with robotic vacuum cleaners. One of the most impactful areas of this transformation is automation in robotics, where machines now perform tasks once handled by human labor. Traditionally, these systems relied on rule-based algorithms, pre-programmed routines, and fixed logic. But in recent years, Artificial Intelligence (AI) has dramatically advanced the field. From helping robots make autonomous decisions to enabling more flexible interactions with their environment, AI is redefining robotic capabilities. These advances include the use of Large Language Models (LLMs), Vision-Language Models (VLMs), and task-specific AI such as image recognition systems. Well-known examples include self-driving cars and Boston Dynamics' robot, "Spot".

These LLMs and VLMs offer a new approach to automation that builds on the challenges of traditional programming. Programming robots has long been a complex task, especially when it comes to high-level reasoning required for navigation, decision-making, or interacting with dynamic environments, challenges that are difficult to solve with fixed algorithms. Unlike classical methods, LLMs can emulate aspects of human reasoning, while VLMs provide visual context by interpreting images. Together, they enable more adaptive, intelligent behavior in robots. These models also lower the barrier to entry for non-experts, as many are freely available and increasingly user-friendly. However, applying them to robotics isn't without challenges. Because LLMs were primarily designed for language tasks, they must be adapted to handle robotics-specific requirements like path planning, spatial reasoning, and real-time control. VLMs excel at environmental awareness through visual input but still require integration with LLMs or task-specific models to drive behavior effectively.

Numerous work has investigated using LLMs and VLMs to control robots already and something most of these have in common is that they trained specific models or implemented frameworks around the LLM to aid or correct it. For example, Ahn et al. [1] designed "SayCan" which uses an LLM to output robotic skills via a learned value function that filters candidate skills. Liang et al. [2] use their framework "LMPC" to increase the memory capabilities of an LLM to improve teachability based on human feedback. Lastly, Yu et al. [3] utilize LLMs to define reward parameters. By optimizing these rewards, or goals, actions to execute a task can be selected. While these approaches achieve positive results, all of these depend on supplemental models or engineered training procedures to correct, constrain, or extend the LLM's native reasoning. While this might still be the best approach, the advancements in LLMs increased their capabilities, leaving the question to be asked wether or not a bare bone LLM can achieve similar results, eliminating the use of additional models or extensions.

In this thesis, the central research question is **"Can large language models navigate a mobile robot through an area, using a minimal supporting framework?"**. The ultimate aim is to enable users to command robots using natural language, with the LLM acting

as both interpreter and operator. For instance, when a user instructs a robot car to move to a specific location, the LLM interprets this request and generates a set of executable navigation commands for the robot. This approach could significantly lower the barrier for robot control, allowing users without programming or AI expertise to interact effectively with mobile robots.

A key technique in this effort is prompt engineering, which is the careful construction and refinement of prompts (i.e., task instructions, examples, and contextual cues) to guide the LLM's output. Throughout this work, various prompting strategies are explored and evaluated to identify effective methods. Feedback mechanisms, which relay the robot's state back to the LLM after each move, are also essential to ensure accurate situational awareness. Nonetheless, the surrounding framework must remain minimal; most of the computational and logical burden should be handled by the LLM itself.

Initial feasibility tests were conducted using the web-based interfaces of several leading LLMs: ChatGPT-4o (OpenAI), Gemini 2.0-Flash (Google), DeepSeek-V3, and DeepSeek-R1. These tests assessed baseline navigational capabilities in a controlled, text-based format. The main body of experimentation was carried out using two models: ChatGPT-4o and Gemini 2.0-Flash. Limited testing was also performed with DeepSeek-R1 and LLaMA 3.1 for comparison purposes. All interactions with the LLMs were conducted via a command-line interface, with the LLM connected to a mobile robot using socket-based communication. Testing was performed in a simulated and physical environment. The Unity game engine provided a virtual testing ground, while real-world tests were conducted using the "TurtleBot3 Waffle Pi" [1]. This robot operates under the Robot Operating System (ROS), ensuring compatibility with other ROS-based mobile platforms. The robot was tasked with navigating an environment that could be either known, where the coordinates of the robot are provided to the LLM, or unknown, where the coordinates are withheld. In the known environment, the robot's position was tracked using the Qualisys motion capture system [2], which uses markers to provide precise coordinate feedback. Obstacles could be present in these environments but were not necessarily tracked in advance; they could also be discovered dynamically through collision feedback. In the unknown environment, positional information was limited, and only the estimated distance to the goal was provided to the LLM.

The feasibility tests demonstrate that while LLMs are capable of basic navigation, they struggle with environmental understanding and handling more complex scenarios. In simple point-to-point navigation tasks within a known environment and without obstacles, both ChatGPT and Gemini performed well. However, when obstacles were introduced, even when their positions and the path they followed were explicitly provided, both models encountered difficulties. They were often unable to apply backtracking or consistently remember obstacle locations, and could only reliably avoid a single obstacle. Turning performance improved when LLMs were instructed to use absolute directions (e.g., "face north") instead of relative turns (e.g., "turn left"). In unknown environments, ChatGPT and Gemini managed to reach the destination, but their behavior suggested reliance on trial-and-error rather than reasoning-based planning. The models typically tried various directions without forming an internal representation or understanding of the environment. This lack of genuine decision-making became evident as many of their "intelligent" choices were simply restatements of information already provided in the task prompt. An important observation during testing was that enabling the LLMs to reason explicitly within their responses significantly improved performance across all test scenarios. This means that rather than being forced to output only the final instruction, they were encouraged to write out their thought process. Despite this, the inclusion of reasoning was still insufficient to fully handle obstacle-related tasks. A recurring issue throughout testing was the failure to follow specific instructions, particularly related to output formatting.

[1] https://www.turtlebot.com/turtlebot3
[2] https://www.qualisys.com/

Chapter 2

Related Work

This section reviews prior research relevant to the application of Large Language Models (LLMs) in robotics. It begins by examining how LLMs function, including limitations and inherent trade-offs when applied to complex tasks such as robot control. Following this, existing work on leveraging LLMs for human-robot interaction (HRI) is explored, highlighting how natural language capabilities facilitate communication but also introduce challenges. Next, frameworks designed to enhance LLM-driven robotic control are introduced. The discussion then addresses navigation strategies in both known and unknown environments, along with HRI in navigational tasks. Research on long-horizon task planning with LLMs is reviewed, including difficulties in maintaining coherence over extended sequences and potential solutions like hierarchical planning and memory augmentation. The chapter concludes with an introduction to prompt engineering techniques proposed to improve LLM performance and reliability.

2.1 Understanding Large Language Models

LLMs are powerful but complex systems. Their ability to solve a wide range of tasks makes them valuable across many fields. However, they also come with significant limitations and risks, especially in high-stakes applications like robotics. It is essential to be aware of these considerations before relying on LLMs in real-world scenarios.

2.1.1 How do LLMs work?

LLMs, as discussed in the work of Shanahan [4], do not "think" or "reason" in the human sense. While they may appear intelligent in conversation, their behavior is grounded in statistical pattern matching learned from vast amounts of textual data. These models generate responses by predicting the most probable sequence of words following a given prompt, based on their training data. This lack of genuine understanding explains why LLMs can fail to follow seemingly straightforward instructions, especially in tasks like navigation that require spatial awareness and consistent state tracking. This limitation became apparent throughout the experiments in this thesis. Despite providing detailed instructions and feedback, LLMs often produced incorrect or nonsensical actions. Their apparent human-like responses can lead users to overestimate their comprehension, making failures more surprising and harder to diagnose [5]. However, in some aspects, they do behave similarly to humans, for instance, they tend to generate better responses when prompted to explain their reasoning, given clear examples, or when the prompt is phrased with greater precision. This underlines the importance of careful prompt engineering to maximize performance.

2.1.2 Hallucinations in LLMs

LLMs have a tendency to hallucinate, which is confidently generating information that is false or fabricated. In the field of robotics, these hallucinations could potentially cause severe issues. For instance, if a robot arm mistakes a person's hand for an object that needs to be manipulated, it could result in harm to the individual. As such, systems capable of identifying and mitigating hallucinations are critical in LLM-based robotics applications. Leiser et al. [6] explored hallucinations in the context of human–LLM interactions. They introduced HILL (Hallucination Identifier for Large Language Models), a tool that detects hallucinations in ChatGPT outputs and provides users with greater transparency. Their work involved the development of several visual design prototypes for HILL, which were then evaluated through user testing to find the most effective and intuitive design. The HILL system offers a variety of feedback: a confidence score for the generated response, a political classification score (ranging from left to right), a score estimating the probability of paid/promotional content, specific hallucination indicators, and a self-assessment score generated by ChatGPT indicating how accurate it believes its answer is. The user study conducted in this research showed that HILL successfully identifies hallucinations and increases user confidence in assessing ChatGPT's reliability, without creating excessive visual noise.

Other researchers have studied hallucinations in the domain of robotics, particularly in planners used for navigation and manipulation. In the work of Ren et al. [7], the authors introduced KnowNo, a system designed to detect uncertainty in LLM responses and request additional user input when needed. The system begins with a natural language instruction and uses a pre-trained LLM to generate multiple possible actions to fulfill the instruction. Then, it applies conformal prediction, a statistical framework for uncertainty quantification, to select a subset of actions that are most likely to be correct. If this subset contains only one action, the system proceeds with it; otherwise, it pauses and prompts the user for clarification or to select one of the available options. The challenge lies in finding the right balance: the system must ask for enough help to avoid incorrect actions, but not so much that it becomes overly reliant on the user. Their experiments, performed on a range of simulated and real robot tasks with varying ambiguity levels, demonstrated that KnowNo improves both the efficiency and autonomy of LLM-based planners compared to existing baselines.

In terms of robotic manipulation, Duan et al. [8] proposed AHA, a vision-language model (VLM) designed to detect and reason about task failures. To generate training data, the authors created a pipeline named FailGen, which modifies tasks from an existing dataset to introduce various types of failures. A total of seven failure types were included, such as incomplete grasp, incorrect rotation, and choosing the wrong target object. To train AHA, each manipulation task is broken down into sub-tasks, which are executed sequentially. After each sub-task, an image of the environment is sent to the VLM to determine whether the task was completed successfully. If not, the model provides a reasoning explanation describing why it interpreted the task as a failure. AHA was benchmarked against six state-of-the-art VLMs, including GPT-4o, Gemini-1.5, and LLaVA-v1.5, and was found to outperform the second-best model (GPT-4o) in all evaluations. Additionally, the authors found that AHA facilitates reward synthesis for reinforcement learning, improves the accuracy of task and motion planning, and enhances zero-shot task verification using synthetic robot data.

2.1.3 Prompt vulnerabilities

LLMs are also vulnerable to prompt modifications, a phenomenon where user input, either intentionally or unintentionally, alters the behavior of the model in undesired ways. In malicious cases, users can craft prompts that cause the LLM to disregard its original instructions or adopt new, potentially harmful behaviors. In conversational assistants, this might result in the LLM providing incorrect information or refusing to respond. However, in the context of LLM-controlled robotics, such vulnerabilities can lead to much more serious consequences, such as autonomous vehicles deviating from their intended paths, or robot arms manipulating objects

they should not. These actions could result in property damage, injury, or even death [9, 10]. Importantly, prompt modifications do not always arise from malicious intent. Seemingly minor changes in phrasing or prompt structure can significantly affect performance. While such variability might be tolerable in general-purpose language applications, it presents a serious concern in robotic control, where reliability and safety are paramount.

The work of Wu et al. [11] investigates unintentional prompt variations in LLM and VLM-controlled robotic systems. They examine various types of prompt alterations and their impacts. For LLMs, vulnerabilities arise from changes such as synonym replacement, word reordering, paraphrasing, and adding overly descriptive or irrelevant details. For VLMs, issues can result from inserting non-task-related objects into the visual scene, applying small image transformations, or reducing visual quality. Their experiments, conducted on a range of robotic manipulation tasks, demonstrated that even minor prompt modifications can lead to performance drops of 19.4% in task execution success rate in the tested systems. These findings highlight the critical need for robust input preprocessing and validation mechanisms. However, care must also be taken to avoid overly restricting user inputs, as this could negatively affect usability and user satisfaction.

2.1.4 Considerations and Trade-Offs of Using LLMs

While LLMs are powerful and flexible tools, their use, particularly in robotics, comes with a number of important considerations and trade-offs. Kasneci et al. [12] explore these concerns in the context of education, but many of their insights are equally relevant for robotic applications. One key issue is the lack of understanding among end users. Although LLMs can convincingly mimic human reasoning, they function in fundamentally different ways. Users without technical expertise may find it difficult to comprehend why the model behaves in certain ways, especially when it makes mistakes. This lack of understanding can also make it difficult to fix problems or adapt the system to specific needs. Even experienced users may struggle to interpret or debug LLM outputs due to the black-box nature of these models. Another major consideration is cost and accessibility. Running powerful LLMs requires significant computational resources. Local deployment offers the greatest control and reliability but is prohibitively expensive for most users, particularly when dealing with the largest or most advanced models. An alternative is cloud-based access, which reduces upfront cost and hardware requirements but introduces other limitations, such as latency, dependence on internet connectivity, and potential service outages. Cloud access also raises concerns about data privacy and security. In robotic applications, users may upload text descriptions, sensor data, or camera images from their environment. Depending on the service provider, this data could be stored, analyzed, or even inadvertently leaked. This poses serious privacy risks, especially in sensitive or personal settings. Finally, there is the issue of environmental sustainability. Training and running LLMs at scale consumes vast amounts of electricity, and the infrastructure that supports them is not always powered by renewable energy sources. As the demand for LLM applications grows, so too does their environmental footprint. These ethical and practical trade-offs must be taken into account when considering the deployment of LLMs and VLMs in real-world robotic systems.

2.1.5 LLMs in Human-Robot Interaction

Human-robot interaction (HRI) is a component of robotics that involves ensuring seamless cooperation between humans and robots. This includes aspects such as clear communication, shared task understanding, trust, safety, and adaptability. Recent work has investigated how the integration of LLMs into robotic systems affects HRI and what design considerations this integration introduces. Kim et al. [13] explore user perceptions of three types of LLM-powered agents, text-based, voice-based, and physically embodied robot agents, across four different tasks. Their study found that users expressed the highest level of satisfaction with the text agent, followed by the robot agent, and finally the voice agent. The voice-based agents, both standalone and embedded within the robot, were perceived as more error-prone. Participants struggled with recognizing appropriate timing for interaction and experienced frustration due

to voice recognition failures. These issues were less pronounced in the robot agent due to its additional communicative signals, such as gaze direction and gestures, which helped users understand when the robot was processing input or ready for a response. The robot agent offered unique advantages in situations requiring multitasking, as users were able to issue commands while attending to another task, relying on the robot's nonverbal feedback for coordination. In contrast, the simplicity and predictability of the text agent made it preferable for purely conversational tasks, as users appreciated the ease of use and faster response generation. The study concludes that enhancing LLM agents with nonverbal communication cues, carefully fine-tuning models for specific domains, and implementing safeguards such as training on curated datasets and maintaining human oversight can significantly improve user experience and safety in HRI settings.

In a different study, Williams et al. [14] propose leveraging LLMs for rapid prototyping of interactive robotic systems, similar to the Wizard-of-Oz (WoZ) methodology. Traditionally, WoZ studies involve humans simulating intelligent robot behavior during early-stage research, particularly when the desired functionality has not yet been fully implemented. Williams et al. argue that LLMs can serve a similar role by standing in for incomplete components of the system, such as speech recognition or natural language generation. Their work discusses how LLMs can be used both for parsing user input and generating appropriate system responses. They acknowledge the risks of using LLMs in this context, especially regarding hallucinations, inconsistent behavior, and bias stemming from the models' training data. However, they also emphasize the value LLMs bring in terms of flexibility, scalability, and realism in simulated interactions. The authors recommend detailed documentation of LLM use in HRI studies, including justification for the chosen model, ethical considerations, and plans for transitioning from prototype to deployable system.

Zhang et al. [15] investigate the use of LLMs to model human behavior in robotic planning. Such human models are essential in enabling robots to anticipate human reactions and act accordingly in socially appropriate ways. Traditional models, whether manually engineered or data-driven, often lack generalizability or require vast amounts of task-specific data. The authors explore the potential of using pre-trained LLMs, specifically text-davinci-003 and FLAN-T5-XXL, as general-purpose task-level human behavior models. They evaluate these models using three datasets: MANNERS-DB, which focuses on social appropriateness; Trust-Transfer, which involves measuring the effectiveness of building and transferring trust; and SocialIQA, which benchmarks commonsense reasoning in social situations. Their findings indicate that text-davinci-003 performs comparably to existing baselines across all datasets, while FLAN-T5-XXL performs well overall but underperforms on trust-related tasks. The models showed particular weaknesses in reasoning about spatial and numerical information, especially in scenarios that involved personal space or geometric relationships. The researchers also found that model performance improved with the use of chain-of-thought prompting and adjustments to how input information was structured. To validate their findings, the authors integrated the LLM-based models into an existing robot planner, previously reliant on custom-built models, and observed similar levels of performance. This suggests that LLMs are viable as task-level human models in HRI, although some limitations remain. Tasks involving nuanced spatial reasoning or requiring precision still present challenges that may necessitate the use of complementary models or further advances in LLM architecture.

2.1.6 Other use-cases of LLMs in robotics

In addition to real-world deployment, LLMs are proving valuable in simulation-based robot training. Training robots in physical environments is often time-consuming and resource-intensive, while simulations can expedite the process and reduce costs. However, manually constructing realistic simulation environments remains a major bottleneck. To address this, Wang et al. [16] developed RoboGen, a fully automated system for simulation environment generation and robot training using a propose-generate-learn cycle, powered primarily by LLMs. RoboGen begins by either selecting a robot-object pair (e.g., a robot arm and a microwave)

or a few example tasks from a predefined task list. Based on this, an LLM generates novel learning tasks, such as placing an object inside the microwave. The LLM then compiles a list of necessary scene assets, which are retrieved from existing databases or generated via a pipeline that includes text-to-image synthesis and image-to-3D model conversion. A vision-language model (VLM) is employed to verify the validity of these generated assets. Once the assets are verified, the LLM scales and positions them to construct a coherent 3D simulation environment. Following scene creation, the LLM decomposes the overall task into sub-tasks and assigns suitable learning algorithms to each one, choosing among methods such as reinforcement learning, gradient-based trajectory optimization, and motion planning using action primitives. When reinforcement learning is selected, the LLM also designs reward functions tailored to each subtask. These sub-tasks are learned in sequence, with the results from each stage feeding into the next. This architecture enables the system to be fully autonomous, while remaining easily upgradable as newer LLMs and VLMs become available. The performance of RoboGen was evaluated on several metrics, including task diversity, scene validity, the quality of training supervision, and the success of learned skills. The system outperformed baseline methods in generating diverse tasks and achieved high success rates in both training and skill execution. Incorporating VLMs into the asset verification process significantly improved the accuracy and realism of the generated scenes. In total, RoboGen successfully created and learned over 100 diverse robotic skills, encompassing manipulation, locomotion, and interactions with soft-body objects.

2.2 Frameworks for Assisting LLMs in Robotic Control

To enhance the performance of LLMs in robotic applications, large frameworks are often developed to provide structured guidance and ensure that the LLM outputs are more consistently accurate and executable. One such framework is "SayCan," introduced by Ahn et al. [1]. This system aims to ground the outputs of an LLM so that they become more contextually appropriate for robotic control scenarios. A known limitation of LLMs is their difficulty in decomposing high-level tasks into actionable sub-tasks without explicit knowledge of the robot's capabilities or the current state of the environment. SayCan addresses this issue by integrating the generative reasoning of LLMs with learned value functions that represent the robot's operational context. When given a high-level instruction, the LLM is tasked with proposing a list of likely useful skills to achieve the goal. Concurrently, the value function evaluates the feasibility of executing each of these skills based on the current environmental and robot states. By combining the LLM's semantic probabilities with the value function's feasibility estimates, the system identifies the most appropriate skill for the robot to execute. SayCan was evaluated in both real and simulated environments, which were an office kitchen and a replica office kitchen setup, across a total of 101 distinct tasks. It successfully created an executable plan in 84% of the test cases and successfully executed the plan in 74% of the cases. Moreover, the system demonstrated the ability to handle long-horizon tasks, successfully planning and executing an eight-step sequence. This study underscores the importance of providing LLMs with sufficient contextual information to support reliable task planning and execution. While their approach relies on a dedicated framework to provide this context, the work in this thesis takes a different path by ensuring that the prompt includes detailed explanations of all possible actions and environmental events, as well as continuously updating the robot's precise location to the LLM.

Another example is the Language Model Predictive Control (LMPC) framework [2] that improves the learnability of an LLM. The framework enables LLMs to remember not only the immediate conversational context, but also prior user interactions and feedback. This extended memory makes it easier to teach a model how to correctly complete robot control tasks based on natural language feedback from users. Instead of treating each interaction as a short-term prompt that is forgotten once it falls out of the model's context window, LMPC treats the process of teaching robots as a kind of sequential decision-making problem. Specifically, they formulate the interaction between humans and robots as a partially observable Markov decision

process, where the human's language serves as observations and the robot's generated code serves as actions. This allows them to train the LLM to predict future interactions based on past ones. To implement this, they fine-tune the LLM to model human-robot conversations, training it to simulate likely continuations of the interaction and choose actions that would reduce the number of user corrections needed. They introduce two variants: LMPC-Rollouts, which predicts full sequences of future interactions using a planning-based approach at inference time, and LMPC-Skip, which is trained to directly jump to the final successful action. Experiments on 78 robot tasks across five different embodiments, both simulated and real, show that LMPC significantly improves performance compared to the base model. It reduces the number of required corrections, improves success rates on unseen tasks, and generalizes better across different robots and APIs. LMPC-Rollouts, in particular, proves more robust in multi-turn teaching scenarios, while LMPC-Skip performs better when the model is likely to get things right on the first try.

Yu et al. [3] utilize LLMs to define reward parameters that can be optimized to accomplish a variety of robotic tasks. Given a user instruction, the LLM translates it into a reward function, which acts as a high-level specification of the robot's goal. These rewards are then passed to a motion controller, based on MuJoCo's model predictive control (MPC), which optimizes them in real time to generate the appropriate low-level actions for the robot. This setup allows for an interactive loop where users can give high-level corrections in natural language, and the robot can adjust its behavior accordingly. The key insight is that, rather than relying on predefined control primitives or hand-written policies, the system leverages the LLM's ability to define meaningful reward structures that can guide complex behavior. The approach is evaluated on 17 tasks using a simulated quadruped and dexterous manipulator, achieving a 90% task completion success rate. It substantially outperforms a baseline method that relied on code-generated control primitives, which succeeded on only 50% of tasks. They also show sim-to-real transfer on a real robot arm, where the system can successfully generate grasping and pushing behaviors, demonstrating that learning through reward translation can enable more flexible and generalizable robot skills.

2.3 Navigation

The use of LLMs in robot navigation presents an exciting research area and forms the central focus of this thesis. Leveraging LLMs can enable practical, nearly fully automated navigation solutions that are designed to be usable even by users without expertise in complex algorithms or specialized robotic tools. Navigation tasks can be divided into two broad categories: known environments and unknown environments. Each category presents distinct advantages and challenges.

2.3.1 Navigating in known environments

Latif [17] investigates navigation within a fully mapped environment. They propose an LLM-based path planning algorithm that utilizes states and actions. The algorithm iteratively computes the best next action based on the current state to move toward the goal state, updating the current state accordingly until the goal is reached. This method was implemented using ChatGPT-3.5-turbo and benchmarked against two established path planning algorithms: A* and Rapidly Exploring Random Tree (RRT). The comparison focused on processing time, path correctness, and path length. The LLM-based planner demonstrated the fastest processing time, being approximately twice as fast as RRT and seven times faster than A*. In terms of path correctness, A* achieved the highest accuracy, with the LLM and RRT following closely behind. Regarding path length, A* produced the shortest paths, with the LLM planner achieving nearly equivalent lengths, while RRT generated the longest paths. These results illustrate the potential of LLMs for efficient path planning in fully known environments, where the layout and obstacles are perfectly mapped. In this thesis, the majority of tests will be conducted in known environments, although obstacles may be either known or unknown.

2.3.2 Navigating in unknown environments

While navigation in known environments is a practical use case and generally yields more reliable results, mapping such environments can be time-consuming, costly, and sometimes unfeasible, especially when environments change dynamically. Consequently, exploring how LLMs perform in unknown or partially unknown environments is equally important and constitutes a significant topic of this thesis.

On-The-Fly Mapping. Some approaches perform mapping of the immediate surroundings at each movement step but do not retain memory of previously mapped areas. Dorbala et al. [18] introduce Language-Guided Exploration (LGX), a novel algorithm for Language-Driven Zero-Shot Object Goal Navigation (L-ZSON), in which an AI agent must navigate toward an object without prior knowledge of the environment. Their system employs LLMs to make sequential navigation decisions and uses a pre-trained Vision-Language Model (VLM) for object recognition. The robot performs a 360-degree scan, capturing RGB images and depth data to generate a cost map of the current area. These images are processed by either an object detection or an image captioning model, whose outputs feed into an LLM that decides the robot's next move. This cycle repeats until the target object is located. The authors investigate different prompt engineering strategies to optimize performance, experimenting with varying narrative perspectives (first person, third person, assistant description), ordering of information within prompts, and natural language scene captions. Evaluation on the RoboTHOR simulated environment shows a 27% navigation success rate improvement over the previous state-of-the-art OWL-ViT CLIP on Wheels (CoW) baseline [19]. Real-world tests on a TurtleBot 2 achieved a 54.2% navigation success rate, demonstrating the complementary strengths of VLMs and LLMs in navigation. Among the prompting strategies, natural language captions performed the worst, likely due to the restricted action space they imply, as they only describe the four cardinal directions as possible paths. No significant differences were found between different narrative perspectives; however, information within the prompt affected outcomes, as LLMs often forgot instructions that appeared earlier in the prompt after being followed by large amounts of other data. This aligns with observations in this thesis, where adding concise reminders at the end of prompts has proven beneficial.

Another relevant example is Zhang et al. [20], who develop NaVid, a navigation system relying solely on monocular RGB input to address Sim2Real transfer gaps commonly encountered with other sensors. Built on the video-based VLM LLaMA-VID, NaVid accepts human instructions alongside continuous RGB frames and outputs movement commands executed sequentially, with updated frames provided after each move until reaching the destination. Trained on extensive navigation data combined with large-scale web data, NaVid achieves state-of-the-art results on the VLN-CE benchmark, offering continuous low-level control within photorealistic indoor scenes, and surpasses baseline methods in multiple real-world tests.

Biggie et al. [21] propose an approach called Navigation with Context (NavCon), which introduces an intermediate layer translating the LLM's knowledge into executable API instructions. This layer mitigates the LLM's inherent lack of real-world awareness and unpredictability in output by parsing inputs into grounded commands. NavCon takes as input an RGB image of the robot's surroundings, a 3D volumetric environmental map, and a natural language command. Their experiments reveal that concatenating the RGB images into a single input improves performance compared to separate, labeled images. GPT-3.5 is used to generate Python code that extracts and processes all input information, converting it into a target 3D location for navigation. A graph-based planner then computes the path to the destination. Tested on a Boston Dynamics Spot equipped with a custom sensor suite, the system attains 90% accuracy in controlled experiments involving 50 different command types, including generic, specific, relational, and contextual inputs, and 70% accuracy in expansive outdoor environments. The framework demonstrates the capacity to identify and navigate to specific objects in relevant scenarios, such as locating a fire extinguisher to respond to an emergency.

Reusable Maps. Some approaches focus on creating reusable maps of the environment to accelerate navigation in areas the robot has previously explored. Huang et al. [22] address navigation in unknown environments by introducing spatial visual-language maps, called VLMaps. These maps are generated using standard Vision-Language Models (VLMs) combined with 3D reconstruction libraries. VLMaps integrate RGB images, depth data, and semantically rich features detected by the VLM to construct a 3D representation of the environment. Unlike purely geometric maps, VLMaps include object names derived from the VLM's detection capabilities, providing a meaningful semantic layer. Leveraging these maps, an LLM can interpret navigation instructions referencing known objects, such as "move between the table and the couch," and decompose them into actionable subgoals. The LLM further translates these subgoals into executable Python code. Huang et al. evaluate their system in both simulated and real-world environments. In simulation, their approach consistently outperforms three baseline methods, including CLIP on Wheels. In real-world tests, it successfully completes 10 out of 20 navigation tasks.

All the systems described so far rely heavily on cameras to capture environmental data, which are then analyzed by VLMs to provide rich contextual input for the LLM. This significantly eases the reasoning burden on the LLM. To specifically assess the reasoning capabilities of LLMs in navigation, my thesis deliberately excludes the use of VLMs or camera input. Instead, navigation will be conducted within a mapped area where only the robot's coordinates, and optionally obstacle locations, are provided. This design restricts the LLM's input to location data alone, compelling it to independently infer and reason about the environment.

2.3.3 Human-Robot Interaction in navigation tasks

Effective navigation depends not only on the robot's movement capabilities but also on the interaction method with human operators and others sharing the space. The operator must be able to communicate with the robot clearly and intuitively, enabling efficient task completion while minimizing misunderstandings or errors. Additionally, robots operating in shared environments must detect and respond appropriately to bystanders, whether by recalculating paths, signaling intentions, or requesting assistance.

Interaction With the Operator. Macdonald et al. [23] introduce the Context-observant LLM-Enabled Autonomous Robots (CLEAR) platform, which provides LLM-enabled robot autonomy through a modular microservice architecture connected via REST APIs. The system is designed around several interconnected components that each serve a distinct role. One part interprets user commands given in natural language, while another processes visual data from the robot's camera. A central module handles the more computationally intensive tasks, and a user interface, accessible via browser or voice, allows people to interact with the robot. Coordinating all of this is a central controller that keeps track of objects, generates natural language summaries, and ensures smooth communication between components. CLEAR relies on pre-trained language models, enabling even non-expert users to control robots effectively through simple language commands. Tested on a simulated quadcopter and a Boston Dynamics Spot, GPT-4 outperformed GPT-3.5 and LLaMA2, achieving an overall task execution success rate of 97%.

Nwankwo et al. [24] explore natural language interaction with autonomous robots, addressing the complexity and steep learning curve of traditional robot programming languages. While prior work on natural language interfaces often involves costly and time-consuming reinforcement learning training, they propose a system integrating LLMs and VLMs to bypass this requirement. Users communicate with the robot through a chat-based GUI linked to an LLM that interprets the commands. Simultaneously, a VLM processes camera and sensor data. Their robot execution node generates movement commands based on this multimodal input. Tested in both simulated and real-world settings, the system achieves 99.13% command recognition accuracy and 97.96% success in command execution.

Interaction with People in the Environment. Luo et al. [25] propose a group-based social navigation framework (GSON) that enables mobile robots to perceive and leverage social relationships among pedestrians to navigate more naturally in shared spaces. Their robot uses RGB cameras and a 2D lidar sensor to scan the environment and detect humans, assigning unique identifiers for subsequent processing. A Large Multimodal Model (LMM), which is a combination of an LLM and a VLM, then analyzes these detections to identify social groups, such as queues or clusters. The robot's planner incorporates this social grouping information to navigate around groups appropriately; for instance, it avoids passing between people standing in a queue. The authors evaluated GSON across 50 simulated scenarios using four different LMMs: GPT-4v, GPT-4o, Gemini 1.5pro, and Claude 3.5-sonnet. All models achieved over 50% accuracy in correctly grouping people, with GPT-4o performing best at 73%. Subsequent testing with GPT-4o in both simulation and real-world environments showed that GSON outperforms existing baseline methods. A real-world long-range navigation trial further validated the approach, demonstrating the robot's ability to traverse a complex path while passing multiple groups without disturbing them, highlighting its effectiveness in diverse social settings.

In contrast, Zu et al. [26] developed LIM2N, a system that allows users to guide the robot through a combination of natural language descriptions and user-generated sketches of the environment. User input, provided as text or voice, is processed by an LLM to extract tasks and environmental details. Meanwhile, an intelligent sensing model transforms 2D laser scans into a visual map, on which users can add annotations, such as drawing navigation paths or marking endpoints, to refine instructions. Pedestrian detection is performed via cameras. This multimodal information feeds into a reinforcement learning controller that manages three possible tasks: point-to-point navigation, following humans, and guiding humans. LIM2N was evaluated in both simulated and real-world scenarios and compared against manual remote control. It outperformed manual control when navigating fixed obstacle environments but showed slightly reduced performance in dynamic settings with pedestrians or unpredictable obstacles. User satisfaction surveys indicated a generally positive experience with LIM2N, with only participants familiar and comfortable with manual control favoring the traditional approach.

2.4 Planning

When navigating over longer distances, an LLM must remember the overall goal, as well as obstacles and other relevant environmental details encountered along the way. However, with extended interactions, LLMs may forget rules or instructions given early in the conversation. Continuously passing all relevant information in every prompt is not always practical, since increasing token count leads to higher computational costs and slower response times.

The problem of LLMs overlooking rules in complex, long-horizon task planning is investigated by Zhang et al. [28]. These tasks usually involve more rules to ensure that the generated plan can be executed correctly, requiring the LLM to retain extensive information and perform complex reasoning. However, due to limited memory and reasoning capacity, LLMs sometimes fail to follow all rules. Their framework, FLTRNN, addresses this by using an LLM to break a task into smaller, manageable subtasks, and then using recurrent neural networks (RNNs) to solve each subtask. RNNs, being designed for sequential data processing such as text, naturally integrate memory management, which reduces the burden on the LLM. The LLM then aggregates all subtask plans to form the overall task plan. To reduce memory load further, they introduce a long-term and short-term memory separation: long-term memory stores basic task information relevant throughout, while short-term memory contains details specific to each subtask. This way, the LLM only needs to remember information about the current subtask rather than the entire task at once. To improve reasoning, the system uses two strategies: Chain-of-Thought prompting, where the LLM is encouraged to explicitly reason through its answers, and a memory graph, a storage structure where the LLM can offload intermediate reasoning information so it doesn't need to keep everything in active memory. The system was tested in a

virtual household environment with a robot capable of manipulating objects. Across all metrics and datasets, FLTRNN consistently outperformed other methods and demonstrated the lowest standard deviation, indicating robustness and stability. They also note that for shorter tasks, a pure planning approach without explicit reasoning performs better.

Rana et al. [29] propose a solution to the same problem by grounding plans generated by LLMs in larger, complex environments using 3D scene graphs (3DSGs). These are hierarchical multigraphs that represent an environment at multiple levels. For example, the highest level might be the floor of a building, with rooms underneath, and objects inside the rooms at the lowest level. The 3DSGs they use add an extra level for assets, which are immovable objects, placed between rooms and objects. Their approach assumes pre-constructed 3DSGs of the environment and consists of two phases. First is the semantic search phase, where the LLM searches through the 3DSG to find the relevant subgraph based on the instruction. This reduces the amount of information the LLM needs to consider, mitigating memory overload and token limit issues. The second phase is iterative re-planning: the LLM generates a plan by selecting nodes from the subgraph to visit, and a path planner such as Dijkstra generates paths between these nodes. The plan is iteratively verified by a scene graph simulator that checks feasibility within the subgraph, allowing corrections if needed. They tested the semantic search phase comparing GPT-3.5 and GPT-4; GPT-3.5 frequently failed while GPT-4 succeeded, illustrating the improved reasoning capabilities of newer models. The full system was tested in a real-world office floor environment spanning 36 rooms and containing 150 assets. The results show the robot can successfully navigate and interact with objects, outperforming baselines due to the iterative re-planning.

Wake et al. [27] propose a method to translate natural-language instructions into executable robot actions using ChatGPT. They design customizable prompts containing six key pieces of information, including the instruction and textual environmental context. Importantly, they found ChatGPT performs more robustly when information is provided separately across a conversation, rather than all at once. Based on these prompts, ChatGPT outputs a sequence of user-defined robot actions with explanations in JSON format, while also reasoning about the environment and estimating post-operation states to support subsequent planning. This essentially lets ChatGPT "reason in its output," removing the need to provide updated environmental info on every input. To reduce incorrect instructions, the system incorporates user feedback. When tested on a simulated robot arm, the system initially achieved a 36% success rate in creating an executable plan without feedback. Analysis of failures revealed two main issues: incorrect verb selection due to ambiguous instructions, and omission of necessary steps because ChatGPT struggled with task granularity. After renaming some possible instructions and providing more examples for granularity, performance improved. With multiple rounds of feedback, ChatGPT generated successful action sequences in all scenarios.

In this thesis, these approaches are less applicable as only short-horizon tasks are performed. Additionally, the simpler environment means less information needs to be remembered. Challenges with LLMs forgetting rules or goals have been addressed by adding short reminders at the end of prompts. However, LLMs still struggle to recall obstacles or paths they have already visited, and this issue will be explored further in later chapters.

2.5 Prompt Engineering

To interact effectively with LLMs or VLMs, one must carefully craft prompts, which are the instructions or queries given to the model. The exact phrasing and structure of these prompts can dramatically influence the quality and relevance of the model's output. Consequently, prompt engineering has evolved into a specialized discipline focused on designing prompts that maximize model performance on specific tasks. White et al. [30] provide a comprehensive catalog of prompt patterns intended to enhance prompt engineering with ChatGPT. Analogous to software development, prompt engineering can be viewed as a form of programming, which makes it suitable for documentation and reuse through standardized patterns. They propose

16 distinct prompting strategies, grouped into 6 categories. Below, I discuss the patterns most relevant to my thesis:

- **Meta Language Creation Pattern:** This involves creating a custom "language" or symbolic format to communicate specific information to the LLM. In my thesis, this pattern is used to define the format of coordinate data that the LLM receives and returns. By explicitly setting this format, the LLM reliably interprets the coordinates and generates appropriate navigation moves. However, care must be taken to avoid ambiguities in the custom language, as these can degrade performance.
- **Persona Pattern:** Here, the LLM is assigned a persona or role to guide the style and content of its output. For instance, the LLM is instructed to act as a robot operator controlling a robot capable of a defined set of movements. This helps the model tailor its responses to the expected context and vocabulary.
- **Template Pattern:** Some tasks require the LLM's output to follow a strict format. This pattern involves providing explicit templates or examples to the LLM to constrain its responses. In this thesis, the LLM must output navigation moves in a specific, easy-to-parse format, which is achieved by demonstrating possible moves and instructing the model to delimit the moves between particular symbols. However, overly restrictive templates can harm output creativity and quality, so it is important to balance constraints with allowing the LLM some freedom. For example, placing strict format requirements only at the end of the response.
- **Infinite Generation Pattern:** This pattern enables ongoing conversations without reinputting the initial instructions every time. It achieves this by giving the LLM a template for how the interaction should proceed. In this thesis, the LLM is prompted to generate one move per turn and then wait for a location update before proceeding, allowing the dialogue to continue indefinitely while maintaining context.
- **Reflection Pattern:** The LLM is prompted to explain the reasoning behind its answers. This not only aids user understanding and validation but often enhances output quality by encouraging the model to "think through" its responses explicitly.

They also emphasize the rapid evolution of LLM capabilities, which can render existing prompt patterns obsolete and create demand for novel prompting strategies. Complementing this, Marvin et al. [31] underscore the critical role of prompt engineering in unlocking the full potential of LLMs. They discuss several optimization techniques relevant to my thesis:

- **Few-shot Prompting:** Including a few examples of the desired task within the prompt helps the LLM better grasp the task requirements and improves response accuracy, contrasting with zero-shot prompting where only an explanation is provided.
- **Chain-of-Thought Prompting:** When reasoning is important, the LLM is encouraged to output its step-by-step thought process alongside the answer, which enhances performance and interpretability.
- **Knowledge Generation Prompting:** The LLM is tasked with generating new knowledge based on given information. For example, when navigating an unknown environment, the LLM might be asked to recognize and remember obstacles based on previous moves and responses.
- **Contextual Prompting:** This involves passing additional context in the prompt to improve decision-making. For instance, providing a list of known obstacle locations helps the LLM plan paths that avoid them.
- **Dynamic Prompting:** The prompt is iteratively updated based on previous model outputs. If the LLM repeatedly makes a particular mistake, this error can be explicitly added to the prompt to guide future responses toward correctness.

Chapter 3

Initial Feasibility Testing

LLMs are primarily designed and optimized for language-based tasks such as answering questions, giving advice, solving problems, assisting with coding, and generating text. However, navigation and spatial planning require structured reasoning about the physical world, which remains a significant challenge for LLMs. These tasks involve a type of reasoning that is fundamentally different from linguistic processing, and LLMs often struggle with the complexities of spatial understanding. To address these limitations, existing research typically uses supportive frameworks around LLMs to enhance their spatial capabilities. These frameworks either support the LLMs by assisting with spatial reasoning or offload that component entirely to other systems, achieving promising results. Nevertheless, as LLMs continue to evolve and improve across various domains, it is becoming increasingly plausible that they may eventually handle spatial reasoning tasks independently. Before evaluating their performance in complex navigation tasks, it is essential to conduct feasibility studies to determine whether LLMs are capable of solving simpler spatial reasoning problems. These preliminary tests not only provide an overview of each model's abilities but also offer insight into the specific failure points in more advanced tasks. They help identify the limitations of current models and guide future work in refining test cases and selecting appropriate support mechanisms.

3.1 Testing Setup

The feasibility tests were conducted using the web-based interfaces of ChatGPT (by OpenAI), Gemini (by Google DeepMind), and DeepSeek (by DeepSeek-Vision). For ChatGPT, the OpenAI Developer Playground was used, which requires paid access and allows users to interact directly with various model versions. Prompts were entered manually, and the responses were evaluated qualitatively based on correctness, reasoning, and consistency. The models used in this study were GPT-4o, Gemini-2.0-Flash, and DeepSeek-V3. Although GPT-4o was accessed via the OpenAI Developer Playground, it was also available for free through the web interface, as were the other two models. However, this free access comes with certain usage limitations. Additionally, an evaluation was carried out using DeepSeek-R1 to explore its potential performance. These models represent the current state-of-the-art in publicly accessible large language models. While they are primarily designed for natural language understanding and generation, they also demonstrate increasingly strong reasoning capabilities. Access to the APIs of both ChatGPT and Gemini also influenced their selection for this study. DeepSeek was included due to its emerging reputation for strong reasoning performance.

3.1.1 Environment

To help the LLMs understand both the nature of the tasks and the structure of the environment they are navigating, a clear and intuitive setup was essential. A chessboard-style grid,

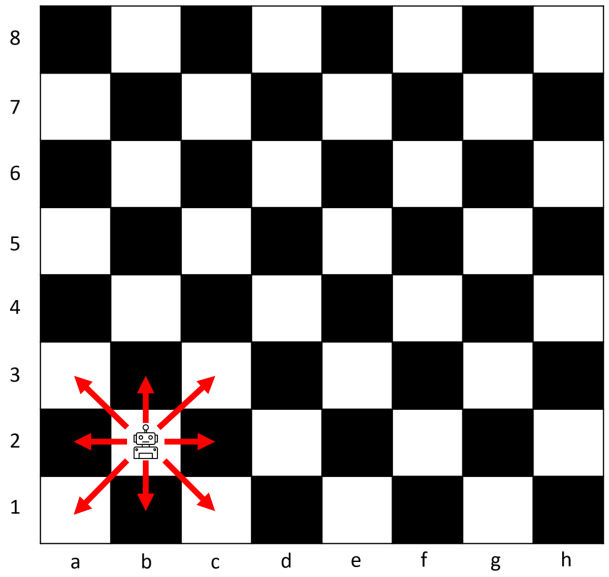

Figure 3.1: Overview of the robot's movement environment and capabilities.

illustrated in Figure 3.1, was chosen as the representation of the environment due to its clarity and simplicity. The chessboard offers a limited number of discrete, labeled squares, each of which can be individually referenced and reasoned about. This makes it well-suited for assessing spatial understanding while reducing ambiguity. The restricted number of locations minimizes environmental complexity, limiting possible paths. It also tests whether the LLMs respect spatial constraints. Movement on a chessboard can be easily described in horizontal, vertical, or diagonal terms, unlike real-world navigation, where any direction between 0° and 360° is possible. This constraint simplifies reasoning about direction while still requiring the models to understand and plan movement.

3.1.2 Robot and Moves

With the environment defined, the next step is to introduce an entity that can move across the board, along with the set of possible moves it can execute. This entity serves as the abstract representation of a navigational agent. While chess pieces such as pawns or queens can move across a board in specific patterns, this study instead uses a robot as the moving entity, given the thesis' focus on robot navigation. The exact identity of the entity is less critical than the movement capabilities it embodies, which must generalize to real-world robotic systems. Most navigational robots, including the TurtleBot3 used in this thesis, can perform basic movements such as moving forward and backward and turning to face a new direction. To navigate toward a target square on the board, the robot must first orient itself in the correct direction and then move forward until it reaches the destination. Functionally, this results in movement capabilities similar to those of a king in chess: one square in any direction, as shown in Figure 3.1. However, unlike a chess piece, the robot must explicitly turn to face its intended direction of movement before proceeding forward. This added constraint more accurately reflects the operational limitations of physical robots.

3.1.3 Obstacles

Obstacles are included in the environment to introduce additional complexity and test the models' ability to reason around constraints. Obstacles occupy entire squares and render them inaccessible from all directions. A square containing an obstacle cannot be moved onto or

traversed, regardless of the direction from which it is approached. If the robot attempts to move forward into a square occupied by an obstacle, it will remain in its current position. Attempting to "jump" over an obstacle by issuing multiple forward movements is similarly invalid; the robot will still not change location. This models physical constraints in the real world, where robots cannot pass through or leap over solid barriers.

3.2 Test Scenarios and Results

To thoroughly assess the capabilities of large language models (LLMs) in navigation tasks, three distinct test scenarios were designed. Each scenario targets different aspects of spatial reasoning and planning:

- **Single-step movement execution:** Evaluates whether the model can correctly understand and execute individual movement commands.
- **Multi-step pathfinding:** Assesses the model's ability to plan a complete route to the target square within a single response.
- **Interactive step-by-step navigation:** Examines the model's capacity to adapt its strategy dynamically based on feedback after each movement, simulating real-time navigation.

In each scenario, tests were conducted both in obstacle-free and obstacle-rich environments. Moreover, in the interactive navigation scenario, obstacles were either known (explicitly provided within the prompt) or unknown (requiring the LLM to infer their presence). In the case of unknown obstacles, the model was expected to detect barriers by attempting a forward move; if the robot's position remained unchanged following this action, it was interpreted as the presence of an obstacle blocking the path. To enhance clarity and facilitate evaluation, the LLMs were instructed to produce textual, ASCII-based representations of the chessboard environment. These visualizations included indicators for the robot's current position and, when applicable, obstacle locations. Examples of such ASCII chessboards, both with and without obstacles, are presented in Figure 3.2. Multiple tests were conducted within each scenario, and all tests in a scenario were executed within a single conversation with the LLMs. Unless otherwise specified, references to DeepSeek refer to the DeepSeek-V3 model.

3.2.1 Limitations of Consistency in LLM Testing

Before discussing the tests and results, it is important to note that when an LLM fails in a certain scenario, it does not imply the LLM is incapable of ever passing it. LLMs are inherently non-deterministic. They generate outputs by sampling from a probability distribution over possible tokens, which can result in different outputs even when the input remains unchanged. While adjusting generation parameters such as temperature and top-p can reduce variability and make the model more consistent, full determinism is generally not achievable in typical usage. In theory, full determinism could be achieved when you have control over all influencing factors, including decoding parameters, model version, tokenization, and prompt formatting. However, in practice when using hosted APIs or web interfaces, external variables such as model updates or hidden system prompts can lead to changes in behavior. This was observed during testing, where the behavior of the LLMs would change on different days, likely due to internal changes. Together with their tendency to hallucinate, this means that it might perfectly pass the test when you repeat it. For this reason, it is essential to test each scenario multiple times to be certain the LLM can pass it. The percentage of successful attempts is a key performance metric, particularly in the context of robotic navigation, where consistent task completion is essential for practical deployment. For these initial feasibility tests, each scenario was only run once, as the goal was not to measure the precise accuracy of the LLMs but rather to gain a general understanding of their capabilities.

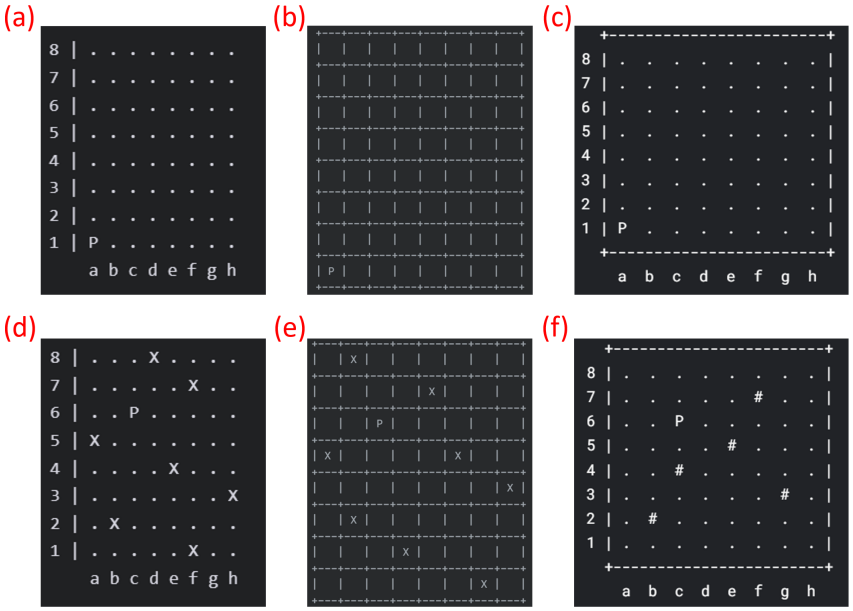

Figure 3.2: ASCII chessboard representations generated by different LLMs, where the robot is denoted by "P". (a–c) Boards without obstacles: (a) ChatGPT, (b) Gemini, and (c) DeepSeek. (d–f) Boards with obstacles: (d) ChatGPT using "X", (e) Gemini using "X", and (f) DeepSeek using "#" to represent obstacles.

3.2.2 Task Prompt

At the beginning of each test, the LLMs were presented with a prompt outlining the task. This prompt included a description of the environment, conceptualized as a chessboard, along with an instruction to render the board in ASCII format and to place the robot at the bottom-left corner. It also specified the scenario-specific objective, detailed the available movement options, and instructed the LLM to redraw the board after each move. In the multi-step pathfinding and step-by-step navigation scenarios, the prompt further included information about the potential presence of obstacles, as well as the locations of any known ones. The task instructions followed the persona prompting pattern as described in the related work (see Chapter 2). In the step-by-step navigation scenario, the meta-language creation pattern was also employed to inform the LLM about the format of the feedback provided after each action.

3.2.3 Single-Step Movement Execution

The first test evaluates whether LLMs can accurately execute spatial movement commands based on user input. In this setup, the user functions as the navigator, providing instructions, while the LLMs are responsible for interpreting and executing them. This task primarily tests spatial reasoning, short-term memory within a conversation, and adherence to explicit instructions. At the start of each test, the LLMs are provided with the robot's initial position, orientation, and a specific movement command. For instance, if the robot is at "a1" and facing upward and instructed to move forward, the correct response would be "a2," the square directly in front. If an obstacle is present at the target square (e.g., "a2"), and the robot is instructed to move forward from "a1," the LLM must recognize the obstruction and respond with "a1," indicating that the move could not be completed. In this test, all obstacles were predefined and explicitly stated in the prompt. Since the LLMs were not responsible for navigation, they were not required to detect unknown obstacles themselves.

Results Without Obstacles. In obstacle-free scenarios, all tested LLMs consistently followed movement instructions correctly. They were able to track the robot's position and orien-

tation and provide accurate responses regarding its final location after a command. However, a common issue was observed with the visual board representations included in their responses. While the textual response indicated the correct resulting square (e.g., "a3" after two forward moves from "a1"), the ASCII board often showed the robot at an incorrect position. This discrepancy suggests that although the LLMs could logically process movement instructions and update position correctly, they struggled to maintain consistency in visual spatial representations.

Result With Obstacles. When obstacles were introduced, the results were more varied. ChatGPT initially produced incorrect outputs by attempting to move the robot over or onto obstacles, suggesting a misunderstanding of the obstacle constraints. After receiving clarifying feedback, it adjusted its behavior and began stopping at obstacles as expected. Gemini correctly recognized and avoided the first obstacle but failed to account for a second one later in the test sequence. DeepSeek, in contrast, demonstrated accurate performance, handling all obstacle constraints correctly. It also provided more detailed and logically structured reasoning in its responses, indicating a more methodical approach. However, similar to the non-obstacle tests, all models exhibited difficulties with accurately rendering the obstacles on the visual board. This further reinforces the observation that while the models can process and reason about spatial instructions, they are less reliable when tasked with maintaining a consistent spatial visualization.

3.2.4 Multi-Step Pathfinding

In the second test, the roles were reversed: the LLMs now acted as navigators. This test evaluates whether the models possess basic path planning and spatial reasoning capabilities. The LLMs were given a start position, orientation, and destination, each defined as a specific square on the chessboard (e.g., moving from "a1" facing up, to "c4"). Their task was to return a set of movement commands that would lead the robot from the start to the destination. For instance, a correct response could involve moving forward three times to reach "a4," then turning 90 degrees to the right, and moving forward twice to reach "c4." If obstacles were present, the LLMs were expected to plan a path around them. As this test is non-iterative, all obstacles were known and provided in advance.

Results Without Obstacles. All models succeeded in reaching the destination without obstacles, but their chosen paths varied in quality and efficiency. ChatGPT and Gemini both followed a similar strategy: first resolving one axis (either horizontal or vertical), then the other. For example, to move from "a1" to "c4," they would first move right to "c1" and then up to "c4." This method is correct but not optimal, as diagonal movement would be more efficient. DeepSeek initially followed a similar strategy but introduced an inefficient detour. For a move from "a1" to "b4," it first moved to "c1," then up to "c4," and finally corrected left to reach "b4." This route is functionally correct but unnecessarily long. When prompted to provide optimal diagonal paths, all LLMs showed significant difficulties. ChatGPT, for example, attempted to move diagonally from "a1" to "b4" by turning 45 degrees to face northeast and moving forward three times, ending at "d4," an overshoot in the horizontal direction. After being corrected, it reverted to the earlier, non-optimal axis-by-axis strategy. On a subsequent correction, it attempted a diagonal move again, but this time ended at "c3," still an incorrect square. Gemini performed worse in this scenario, failing to produce any correct path when prompted for optimality. DeepSeek consistently returned correct but non-optimal paths, and could not successfully implement diagonal movement when explicitly requested to do so.

Results With Obstacles. Obstacle navigation proved to be a significant challenge for all LLMs, with none of them managing the reach the destination without breaking the rules and traversing over obstacles. The test scenario included two staggered rows of five obstacles each: one on the fourth row skewed rightward, and one on the sixth row skewed leftward, with an

overlapping center. The task required moving from the bottom of the board to the top, forcing the robot to navigate around both rows. ChatGPT successfully identified and avoided the first row of obstacles but failed to avoid the second, proceeding onto an obstructed square as if it were clear. DeepSeek showed similar behavior: it avoided the first obstacle row and acknowledged the second but misjudged its position. It attempted to sidestep the obstacles after reaching them but found itself on top of the blocked squares and became stuck. Gemini exhibited particularly erratic behavior. After reaching the first obstacle, it entered a loop in which it repeatedly alternated between two paths, both of which led into obstacles. This cycle caused it to get stuck in an infinite loop, reaching the output limit and terminating.

DeepSeek-R1. Significant improvements were observed when using DeepSeek-R1. The model successfully avoided both rows of obstacles and followed an optimal path to the destination. Furthermore, its board representation was consistently accurate, with both obstacles and robot positions correctly visualized. However, this came at a substantial cost in response time. Generating the initial board alone required approximately 60 seconds. Complex tasks, such as navigating past obstacles or computing optimal routes, took five to six minutes to complete. Additionally, the model outputted its full internal reasoning process, which in one case was nearly 5000 words for a single task. Although DeepSeek-R1 delivered excellent navigation results, its impractical runtime made it unsuitable for further feasibility tests, particularly for interactive tasks requiring multiple sequential moves. If each individual move took several minutes to process, completing even a simple multi-step navigation would become infeasible.

3.2.5 Interactive Step-By-Step Navigation

The final test builds upon the previous one but introduces interactivity by requiring the LLMs to provide navigation instructions one step at a time. In this scenario, the models could only output a single movement command per message. After each move, feedback was provided indicating the resulting square. For example, if the model started at "a1" facing up and responded with "forward," the reply would be "a2." Obstacles in this test could be either known or unknown. This test assesses the LLMs' ability to remember previous instructions, track their position and orientation over time, and incorporate feedback dynamically. Interactive navigation is a more realistic simulation of robotic control, as it allows error correction mid-process and is thus used in later stages of this research.

Results Without Obstacles. Initial results mirrored those of the multi-step pathfinding task. Both Gemini and DeepSeek initially failed to comply with the one-step-at-a-time format, returning full navigation sequences instead. This was resolved through additional instruction and examples. All three models ultimately reached the destination, but consistently followed suboptimal paths. When asked to provide optimal routes, results diverged. Simple diagonal paths (e.g., "a1" to "d4") were executed correctly in most cases. However, slightly more complex routes requiring mixed movement (e.g., "a1" to "b3," which combines diagonal and vertical movement) were generally mishandled. DeepSeek, for instance, correctly moved diagonally from "a1" to 'b2," but then incorrectly assumed another diagonal move would result in "b3,' a misunderstanding of movement direction. ChatGPT and Gemini also began correctly but soon veered off course, missing the target and sometimes hallucinating moves or navigating randomly.

Results With Obstacles. Obstacle handling in this interactive format posed difficulties initially. ChatGPT and Gemini both struggled to correctly incorporate obstacle rules, but showed notable improvement after receiving targeted feedback. ChatGPT, for example, failed to recognize blocked squares at first but adapted once the movement rules were re-explained. Gemini exhibited a similar issue, misclassifying known obstacles as unknown and misunderstanding their behavior. This too was improved with further clarification. DeepSeek, by contrast, demonstrated robust reasoning from the start. It proactively explored the environment by

testing squares and analyzing the feedback to infer the obstacle mechanics. When faced with uncertainty, it would ask for the result of a move before assuming its success, effectively simulating an exploratory learning process. As a result, DeepSeek was able to reach the destination while successfully avoiding both known and unknown obstacles.

3.2.6 Conclusion

The results of the tests indicate that ChatGPT, Gemini, and DeepSeek possess a basic level of spatial reasoning, enabling them to navigate simple environments. However, their performance is often inconsistent. All models demonstrated proficiency in following explicit navigation instructions and remembering known obstacles when properly informed of task rules and conditions. Multi-step path planning was feasible but became less reliable in the presence of obstacles, while the LLMs remembered obstacle locations, they occasionally collided with them or became stuck in repetitive loops. Additionally, the models frequently favored orthogonal movement paths over optimal diagonal routes. DeepSeek-R1 showed significant improvements in handling these challenges and correctly executed tasks involving obstacles and path optimization. However, it incurred long response times, sometimes several minutes per move, rendering it impractical for real-time navigation scenarios. Interactive step-by-step navigation, where moves were executed one at a time with feedback, yielded better results than generating the entire path at once. This approach allowed LLMs to incorporate feedback, crucial for detecting and responding to unknown obstacles. Nonetheless, the models sometimes exhibited hallucinations or erratic behavior when unexpected situations arose. Visual representation posed a consistent challenge for all models except DeepSeek-R1. The positioning of the robot and obstacles on ASCII boards frequently did not match the actual state, limiting the usefulness of visual aids. Regarding response times, ChatGPT and Gemini were highly responsive, while DeepSeek's models were noticeably slower. DeepSeek-V3 offered improved speed compared to R1 but remained slower than the other models. Gemini's superior speed can be attributed to the 2.0-Flash model's design, which prioritizes faster response times at some cost to reasoning performance.

Chapter 4

Concept

Initial feasibility tests demonstrated that LLMs are capable of performing basic navigation tasks. They could interpret and execute movement commands, and generate navigation instructions, either step-by-step or as complete sequences, when given a destination. However, these early experiments also revealed limitations, particularly in handling obstacles and producing consistent outputs. Based on these initial insights, a more structured evaluation was conducted to rigorously test the models' capabilities under controlled conditions and with refined prompting strategies. The goal was to explore the performance boundaries of the LLMs and assess their consistency and reliability when provided with well-crafted inputs. This was approached through iterative prototyping and quantitative testing, which is discussed in Chapter 6. Both simulated and real-world environments were used during the evaluation to ensure practical relevance and controlled experimentation. This chapter outlines the system setup, the input and output formats, the prompting and navigation strategies used, and the different scenarios that are tested.

4.1 System Overview

This section provides an overview of the system that is used in this thesis. It describes the key components of the setup, including the language models used, the interaction mechanism between the user and the model, the hardware platform itself, and the characteristics of the environment in which the robot operates. Together, these components form the basis for the evaluation.

4.1.1 Interaction Method and Models

Text-based interaction with the LLMs was used throughout the evaluation. Users provided input via the terminal, specifying the robot's current location, destination, and, when necessary, orientation. During the feasibility testing phase, ChatGPT, Gemini, and DeepSeek were accessed via their respective web clients. For subsequent testing, however, API access was required to automate the process and improve efficiency. Therefore, ChatGPT and Gemini were primarily used, as API access was available for both OpenAI and Google's Gemini. That said, Gemini imposes a daily and per-minute token limit, though these limits were broad enough not to impact the evaluation. The specific models used were GPT-4o and Gemini-2.0-Flash, as these were the most up-to-date models available via API at the beginning of the thesis. It is possible that the models received updates during the course of the thesis, which may have influenced results. Brief experimentation was also conducted with other models, specifically Meta's LLaMA3.1 and DeepSeek-R1. DeepSeek-R1 was selected due to its strong performance during feasibility testing, while LLaMA was chosen for its popularity in related work. Since these models do not have publicly available APIs, and LLaMA is not directly accessible in this

Used Models, Parameters and Platform			
Model	Temperature	Top-p	Platform
GPT-o4	0.1	0.1	OpenAI API
Gemini-2.0-Flash	0.1	0.1	Gemini API
DeepSeek-R1	Default/0.1	Default	OpenRouter/Ollama
Llama3.1	0.1	Default	Ollama

Table 4.1: Overview of the language models used in the experiments, including their temperature and top-p sampling values, as well as the platform through which each model was accessed.

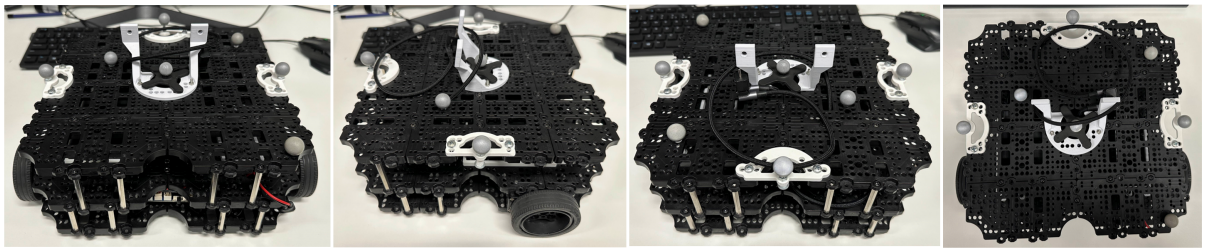

Figure 4.1: The TurtleBot3 Waffle Pi seen from different angles and with infrared tracking markers (gray spheres) attached to it. From left to right: front view, right-side view, rear view, and top view.

region, alternative access methods were used. These included the third-party API platform OpenRouter and running the models locally using the Ollama framework. The implementation details of these methods are discussed further in Chapter 5. Table 4.1 provides an overview of the models used, including the temperature and top-p values, as well as the platforms through which the models were accessed. The temperature and top-p, which influence the randomness of LLM outputs, are described in detail in Chapter 5.

4.1.2 Robot Platform

The first step in the experimentation process was selecting a suitable robot platform, which influenced the set of possible moves. The robot selected for this thesis is the "TurtleBot3 Waffle Pi[1]", shown in Figure 4.1. The TurtleBot3 is compact and highly maneuverable. It features two independently controlled wheels positioned near the front of the robot. This configuration allows the robot to move forward and backward, as well as rotate in place by driving the wheels in opposite directions. These movements are consistent with the moves used in the earlier feasibility tests (see Chapter 3). The robot is powered by an 11.1V 1800mAh rechargeable Li-Po battery. While the TurtleBot3 supports flexible movement, it is not highly precise. Inaccuracies in execution can result from factors such as motor inconsistencies, wheel slippage, and the limitations of the onboard motion controller. For example, small differences in motor speed can cause the robot to drift slightly during straight-line motion. Additionally, due to the wheels being offset from the robot's center, rotating in place causes a minor shift in the robot's central position. A 3D simulation model of the TurtleBot3 was also used during development. The simulated version mirrors the real robot's movement capabilities, with one notable exception: in simulation, the robot rotates precisely around its central axis, meaning its coordinates remain unchanged during turns. This differs slightly from the physical robot's behavior but provides a useful approximation for testing control logic.

[1]https://www.turtlebot.com/

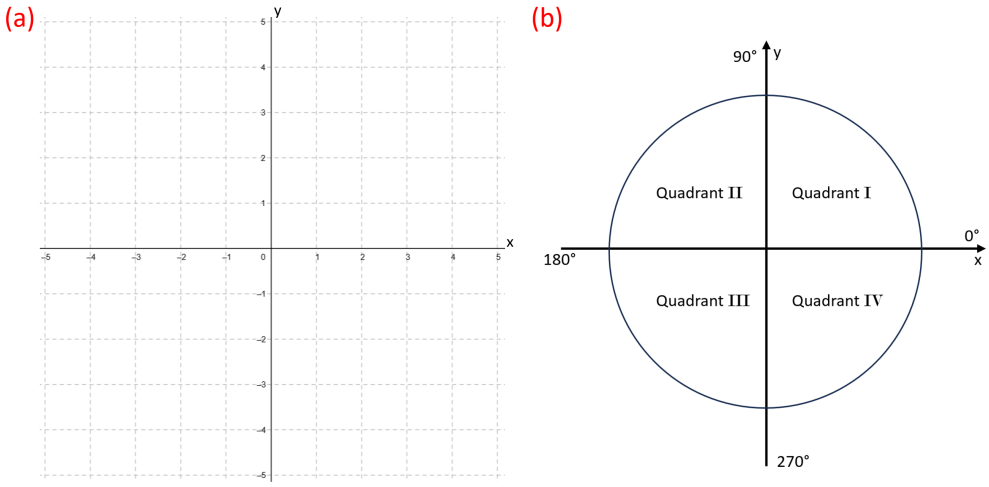

Figure 4.2: The navigation environment setup: (a) a 2D Cartesian coordinate system used for positioning, and (b) the mathematical orientation convention showing the four quadrants.

4.1.3 Environment Setup

The next step was to define the environment in which the robot would navigate, including its characteristics such as coordinate representation and obstacles. The chessboard-based setup used during the feasibility tests (see Chapter 3) served as a useful starting point but was ultimately too limited in terms of space and flexibility. Although the concept of an "infinite chessboard", meaning one with an unlimited number of squares, was considered, it still restricts movement to a fixed grid and a limited set of directions. A more versatile alternative is to use continuous coordinates. This approach provides a theoretically unlimited navigable area and allows the robot to orient and move in any direction. A two-dimensional Cartesian coordinate system (x,y) was selected for this purpose, as the robot does not move along the vertical (z-axis) and the environment is assumed to be flat. A schematic of this coordinate system is shown in Figure 4.2 (a). For representing orientation, degrees were chosen since they allow for expressing all possible headings from 0° to 360°, and this format integrates easily with Unity's simulation environment. However, it is important to note that there are multiple conventions for defining orientation, which can lead to inconsistencies. In Unity, 0° is aligned with the positive y-axis, and positive rotations proceed in the clockwise direction. In contrast, the standard mathematical convention defines 0° along the positive x-axis, with positive rotations occurring counterclockwise. Both conventions were tested in the simulation environment, while only the mathematical convention was used for the real-world experiments. The rationale behind this decision will be discussed in Chapter 6. A visualization of the mathematical orientation convention is provided in Figure 4.2 (b). Obstacles may also be present in the environment. These obstacles are considered impassable by the robot and may be either known in advance or unknown, depending on the specific experimental setup.

4.1.4 Tracking the Robot

Most experiments in this thesis were conducted in a Unity-based simulation. Unity offers direct access to the position and orientation of objects, making robot tracking straightforward. Some experiments were also performed in a Python-based simulation, which, while lacking a visual component, offers similar access to positional and orientation data. In the real world, however, tracking the TurtleBot3 requires additional hardware. All real-world experiments were conducted in a room equipped with the "Qualisys" motion capture system. This camera-based system uses infrared tracking to determine the precise location and orientation of objects with attached reflective markers. When applied to the TurtleBot3, this setup provides real-time

positional and orientation data, similar to what is available in simulation. Implementation details for both the simulations and the Qualisys system are discussed in Chapter 5.

4.2 Input and Output

To carry out any navigation task, LLMs require structured input and must return output in a usable format. In this thesis, the input includes the initial task prompt, details necessary to initiate a navigation task (e.g., position, orientation, and goal), and updates about the robot's progress. Based on this input, the LLMs generate commands that are parsed and executed. This section explains the design of the input-output framework.

4.2.1 Task Prompt

Crafting an effective task prompt is essential for optimizing the LLMs' performance. The prompt must clearly describe the task, the robot's environment, possible actions, and rules, while also addressing challenges identified during earlier experiments. It is structured in several logical sections. First, a task and environment description outlines the robot's role, the objective of reaching a destination, and the layout of the environment, including the coordinate system and orientation convention. This is followed by detailed instructions, which provide rules for interpreting input and deducing orientation, along with specific guidance aimed at known LLM limitations. If obstacle detection is relevant, the prompt includes a section on obstacle handling, where the LLM is instructed to infer obstacles based on whether the robot's position changes after a forward move. If the displacement is smaller than expected or unchanged, an obstacle is assumed. In some cases, the prompt also describes error handling strategies to account for inaccuracies due to rounding or hardware limitations of the TurtleBot3. A section on output formatting then specifies how responses should be structured, typically requesting step-by-step instructions and waiting for feedback before continuing. Finally, a reminder reinforces the core objective and key constraints, often updated based on observations from previous LLM performance.

4.2.2 Input

After the task prompt, which remains fixed for each scenario, additional input is needed to provide the LLMs with the current state of the robot and to ensure it can keep track of progress throughout the navigation task. This input includes the robot's current position, orientation, and destination. Since the objective of this thesis is to delegate most of the task logic to the LLMs, the input is designed to be minimal yet sufficient. Experiments were conducted in two types of environments: a tracked environment, where the robot's exact coordinates and orientation are always known, and an unknown environment, where only the distance to the destination is available.

In the tracked environment, the robot receives precise (x, y) coordinates for both its current location and the destination. The orientation is either explicitly provided or left for the LLM to infer, depending on the experimental setup. If the orientation is provided, the input is phrased as: "You are at (x, y), move to (x', y'). Orientation: θ". If the orientation must be inferred, the same message is sent without the orientation, and instructions for deducing it are included in the task prompt. Additionally, if obstacles are present and known, their locations are appended to the message in the format: "Obstacles: (x1, y1), (x2, y2), ...". This allows the LLM to incorporate environmental constraints into its planning and navigation decisions." To help the LLM keep track of its position during navigation, it is given feedback after each move. If the robot moves forward or backward, the new position is sent to the LLM. Initially, the same approach was used after turning, but since the robot's position does not change during in-place rotations in the simulation, this sometimes resulted in input that resembled an unsuccessful movement. To avoid ambiguity in subsequent reasoning, the simulation was adjusted to send a confirmation message like "Successfully turned" instead. In the real-world scenario, since

the physical TurtleBot3's position does change slightly after a turn, its new coordinates were always provided, even after rotations. In the unknown environment setup, the robot receives less precise input. Instead of exact coordinates, only the distance to the destination is given. This form of abstraction was used to test the LLMs' reasoning ability and adaptability when dealing with uncertainty or limited feedback. In scenarios with obstacles, an additional strategy was tested in which the LLM received not only the current position but the full path traversed so far. This was intended to help the model remember previously visited locations, recognize when it was looping, and improve its ability to backtrack after encountering blocked paths.

4.2.3 Output

To move the robot, the LLMs must generate commands that can be interpreted and executed by the system. Throughout all experiments, the available commands were limited to forward, backward, and turn, reflecting the capabilities of the TurtleBot3 platform. In scenarios where the LLM needed to determine the orientation before navigating, it also had to output an orientation value. Two different output styles were evaluated during testing. In the first style, the LLMs were instructed to provide only the necessary value, which was either the movement command or the orientation, without any additional explanation. This made parsing the response straightforward and reduced the risk of misinterpretation. In the second style, inspired by the reflection pattern and Chain-of-Thought (CoT) prompting method discussed in the related work (see Chapter 2), the LLMs were encouraged to explain their reasoning. This often resulted in richer, more detailed output that included the logic behind each decision. To make parsing still possible in this format, the LLMs were asked to follow a specific convention: orientation values were prefixed with "Orientation:" and movement commands were enclosed in dollar signs, such as "$forward$" or "$turn 90$". This structure allowed the system to extract the necessary commands using simple regular expressions, while still benefiting from the LLMs' intermediate reasoning.

4.3 Prompting and Navigation Strategies

Various prompting and navigation strategies were employed to evaluate which approaches yielded the most reliable results. This section outlines the different prompting techniques used to make the LLMs aware of the robot's orientation, as well as the methods tested for executing turns during navigation.

4.3.1 Orientation

Two approaches were evaluated for informing the LLMs of the orientation of the robot. The first involves explicitly providing the orientation to the LLMs at the beginning of the task. This method is straightforward in the simulation, where the orientation can easily be accessed through code, but less practical in real-world scenarios, where obtaining the orientation may require additional hardware or estimation. The second approach delegates this responsibility to the LLMs by requiring them to deduce the orientation. This is achieved by moving the robot forward once and comparing its new position to the original one. In addition to simplifying the input required from the user, this method aligns with the overall goal of this thesis: minimizing the supporting framework and maximizing the role of the LLMs. Overall, the LLMs were tested using two different output strategies. In the first, they were instructed to return only the final orientation value. In the second, they were asked to show their reasoning process step by step, including the calculations used to determine the orientation.

4.3.2 Turning Methods

Multiple turning strategies were explored during experimentation, broadly categorized into relative and absolute turning. In relative turning, the LLM specifies the angle the robot must turn relative to its current orientation. For instance, if the robot is facing 90 degrees and is

instructed to turn 90 degrees in the positive direction, it will end up at 180 degrees. This approach is direct and requires the LLMs to keep track of the robot's orientation themselves. Three different variants of relative turning were explored:

- **Directional commands with angles:** The LLMs are instructed to specify turns using terms like "left" or "right" along with a degree value (e.g., "Turn left 90 degrees"), where "left" corresponds to a counter-clockwise turn and "right" to a clockwise one.
- **Clockwise/counter-clockwise phrasing:** Instead of using "left" or "right," the LLMs are instructed to use "clockwise" or "counter-clockwise" with the angle (e.g., "Turn counter-clockwise 45 degrees").
- **Signed degree values:** Turns are expressed using numeric values with positive or negative signs (e.g., "Turn 90 degrees" or "Turn -45 degrees"). The direction of the turn depends on the used orientation convention.

In absolute turning, the robot is instructed to face a specific global orientation regardless of its current direction. Since the physical robot executes only relative turns, this method requires the control code to track the robot's current orientation and compute the shortest relative turn to reach the desired global direction. Two absolute strategies were explored:

- **Numeric angles:** The LLM specifies the target orientation as a global angle (e.g., "Face 90 degrees").
- **Named directions:** The LLM uses direction labels instead of angles. These can include compass directions (e.g., "North" or "Southwest") or coordinate-based descriptors (e.g., "positive-x" or "negative-y") that indicate movement along one or more axes. For diagonal directions, compound labels such as "positive-x positive-y" are used to indicate combined movement. However, such naming conventions are only feasible when movement is limited to a discrete set of standard directions (e.g., multiples of 45 degrees). When arbitrary orientations are allowed, such as any angle between 0° and 360°, these label-based methods become impractical, as there are no intuitive or standardized names for all possible directions.

These various strategies were developed to explore how format and phrasing affect the LLM's ability to guide robot navigation, and to assess which forms of instruction align best with spatial reasoning tasks.

4.4 Experimental Scenarios

Experimentation and testing were conducted across three distinct scenarios: grid-based movement, free movement, and distance-based movement. Each scenario was designed to evaluate different aspects of the LLMs' reasoning capabilities. This section outlines the characteristics of each scenario, including the specific rules, objectives, and whether the tests were performed in simulation, the real world, or both.

4.4.1 Grid-Based Movement

Following the initial chessboard-based feasibility tests, a natural progression was to explore navigation in a grid-based environment. In this setup, movement was restricted to whole-number coordinates, meaning the robot could only move to locations where the grid lines illustrated in Figure 4.2 (a) intersect. This approach is conceptually similar to the chessboard test but offers a much larger area for navigation. Consequently, forward and backward movement was always executed in unit steps, effectively locking the robot to the grid. While translational movement adhered strictly to unit distances, turning was constrained to discrete increments of 45 degrees. This meant the robot could only face and move in orthogonal (e.g., 0°, 90°, 180°, 270°) or diagonal (e.g., 45°, 135°, 225°, 315°) directions. For orthogonal directions, each forward movement covered a distance of exactly 1 meter. In contrast, for diagonal orientations,

the robot moved $\sqrt{2}$ meters forward in order to arrive at a new location with whole-number coordinates. This value corresponds to the diagonal of a 1-by-1 meter square and was necessary to preserve the grid structure. This grid-based movement setup offered a balance between simplicity and meaningful experimentation, allowing for clear evaluation of the LLMs' ability to interpret orientation, plan paths, and issue correct commands. To determine orientation in this context, the LLMs were instructed to move forward by one unit and infer their direction from the resulting change in coordinates. All experiments within this scenario were conducted in a simulated Unity environment. The physical TurtleBot3 was not used due to its limited ability to consistently perform precise movements required for a strict grid layout.

4.4.2 Free Movement

While grid-based movement offers a controlled framework for navigation testing, it lacks realism due to the rigidity of its constraints. Therefore, additional experiments were conducted in a more flexible scenario: free movement. In this scenario, the robot was allowed to traverse the coordinate system continuously. Forward and backward movement could be executed over any distance with centimeter-level precision, and turns were permitted to any angle between 0 and 360 degrees. Although this setup increases the complexity of individual computations, it often reduces the total number of actions required to reach a destination. In contrast to the grid-based scenario, where movement directions are constrained, optimal navigation here typically consists of orienting directly toward the destination and moving the precise distance. More extensive sequences of actions were only required when either the orientation or distance was incorrectly estimated, or when obstacles were introduced. To determine the initial orientation, the robot was instructed to move forward by 20 centimeters, which was a compromise between minimal movement and sufficient displacement to calculate direction accurately. A smaller move may not provide adequate coordinate change for reliable orientation estimation. Experiments were conducted in both simulated and real-world environments.

4.4.3 Distance-Based Movement

In contrast to previous environments where precise coordinates were available via motion capture systems, distance-based navigation offers a simpler alternative that avoids the need for global localization. To evaluate whether LLMs could reason under such sparse input, a simplified Python-based simulation was created in which the robot received only the Euclidean distance to the goal, no coordinate or orientation data were provided. Grid-based movement was used to simplify the task and reduce the number of possible directions, with steps of 1 meter and turns in multiples of 45 degrees. Only relative turning was possible, since absolute orientation could not be determined without coordinate data. Obstacles were excluded from these tests due to the already limited information available and the challenges they posed in more informative scenarios. The baseline strategy under these conditions is to iteratively move forward and monitor whether the distance to the destination decreases. A naive algorithm would check all directions repeatedly, but a more intelligent agent could use spatial reasoning to infer general directionality, which could reduce the amount of moves needed to reach the destination. For instance, if a forward move decreases the distance, it can be inferred that the opposite directions are unlikely to be optimal. Therefore, the goal is to see if the LLMs possess this kind of spatial reasoning. This approach was not extended to real-world testing due to unsatisfactory results in the simulation. Nonetheless, it served as an interesting conceptual baseline for reasoning under minimal input conditions.

Chapter 5

Implementation

This chapter will go further into detail on all the systems and software that were used to perform the experimenting and testing. It will include how the communication with the LLMs happened, the configuration of the models, how data is passed between the LLMs and the robot, how the simulation is built and how the TurtleBot3 is tracked and controlled. All software components, excluding those executed directly on the TurtleBot3, were run on an MSI laptop equipped with an AMD Ryzen 7 8845HS processor, integrated Radeon 780M graphics, and a dedicated Nvidia GeForce RTX 4070 Laptop GPU. The system has 16 GB of RAM and operates on Windows 11.

5.1 Model Parameters

Several model parameters can be adjusted to influence the output of large language models (LLMs), including temperature, top-p and top-k. The most used ones are temperature and top-p, commonly referred to as nucleus sampling. These are also the most widely available across LLM platforms and APIs.

Temperature. The temperature parameter controls the degree of randomness in the model's output. LLMs select from multiple candidate tokens to output, based on probability. More likely words will have a higher probability and less likely words will have a lower probability. The temperature changes the probability of these words. With a lower temperature, the gap in probability between the most and least likely words is increased. Consequently, tokens with the highest likelihood are selected more frequently, resulting in more deterministic and predictable outputs from the LLM. With a higher temperature, the gap in probability is decreased, meaning less likely words also have a higher chance of getting chosen. This increases generative diversity but may lead to less coherent or contextually appropriate outputs. temperature can usually have a value between 0 and 2.

Top-p. In contrast, top-p determines the amount of words that are considered for the output. It does this by choosing the smallest set of words whose cumulative probability mass exceeds p. A higher top-p means more possible words are considered, making the text more diverse. A lower top-p means less considered words and more deterministic and repetitive answers. Top-p can have a value between 0 and 1.

For a robot navigation task, the optimal values of these parameters are dependent of the specific situation, but overall the LLM should provide predictable and deterministic output. This is due to the presence of strict formatting constraints that the LLMs must adhere to, and excessive generative variability may compromise these requirements. Moreover, navigation tasks often involve a single optimal path, and taking any other route is inefficient. For obstacles, there is

maybe a bit more creativity required to find a good solution. However, an obstacle can also be avoided in only 1 or more optimal ways, hence, deterministic output remains preferable. Therefore, this suggests that maintaining both parameters at low values to make the responses nearly completely deterministic. However, a problem with this is that an incorrect response would also be repeated every time in this scenario. Therefore, in this implementation, both temperature and top-p are set to 0.1, which increases determinism while retaining a small degree of variability to mitigate error repetition. For example, when both parameters are set to low values, the output is typically concise and adheres strictly to the required format, such as "Forward 3" or "Turn 90". In contrast, with higher values for temperature and top-p, the model may produce more verbose and descriptive outputs, such as "Move forward 3 meters" or "Turn 90 degrees to the left", which may violate format constraints and reduce consistency in command interpretation. Additionally, with lower parameter values, the model is more likely to consistently follow the same path for identical inputs, which is advantageous in scenarios where predictable and repeatable behavior is required, such as robotic navigation.

5.2 Communication with LLMs

In this section the technical details of the communication with the LLMs is explained. This includes the communication with ChatGPT, Gemini and other models. The APIs or services to connect to the models will be mentioned, along with the structure of the code that is used and where the models are running.

5.2.1 ChatGPT

As mentioned in Chapter 4, APIs are used to communicate with LLMs. These APIs enable fully automated communication, thereby maximizing efficiency in reducing the robot's time to reach its destination. Access to the OpenAI API [1] is available for interaction with ChatGPT. Additionally, this gives access to the OpenAI Developer Platform, which also includes access to the Assistants API. This API allows users to create a personalized "Assistant" via the web interface and configure various parameters. These include system instructions, the model, files, functions, response format and model parameters, such as temperature and top-p. This approach facilitates the creation of a task-specific assistant, offering a more streamlined process than the standard API. The task prompt is specified in the system instructions field, with the temperature and top-p parameters set to 0.1 and the selected model being GPT-4o. The API is accessed using Python and the OpenAI library, which offers code to connect to the API with the API key, retrieve the assistant created in the web interface, create a conversation thread, add messages to the thread and retrieve the responses. The procedure begins by establishing a connection to the API with the API key. Subsequently, the assistant is retrieved via its ID and a conversation thread is created. To transmit a message, one is added to the thread with the "user" role and the thread is run with the correct assistant. Upon completion of the run, the thread's message list is queried, and the most recent response is retrieved. The code for this can be found in the documentation of the OpenAI API.

5.2.2 Gemini

For Gemini, the API is freely available from Google [2]. However, no web interface is available for customization, necessitating configuration via code. As with ChatGPT, interaction was implemented using Python code, specifically using the "genai" library developed by Google. With this library, the API can be accessed using an API key, a chat can be created with a specific model and a configuration that includes system instructions and model parameters can be passed. Messages can also be added to the chat, and responses retrieved accordingly. The free API has a rate limit of 1500 requests per day and 15 requests per minute. To begin,

[1] https://openai.com/api/
[2] https://ai.google.dev/

a connection to the large language model (LLM) is established using an API key. A chat session is then created using a specific model and configuration; in this case, the model used is Gemini-2.0-Flash. The task prompt is stored in a .txt file, read into the program, and passed in the configuration alongside the model parameters. The temperature and top-p are set to 0.1. Initiating a conversation is comparatively simpler than the OpenAI API, as the `send_message()` function automatically returns the LLM's response. However, the response remains an object, from which the textual content must be extracted. The Gemini documentation provides code snippets for all these tasks.

5.2.3 Other Models

Two other models are used: DeepSeek-R1 and Llama3.1. These do not have free APIs available from the developers and subsequently need other methods to access them, such as using OpenRouter or running them locally.

OpenRouter. The first method is via OpenRouter [3], an online platform providing both paid and free API access to a variety of AI models. The service has providers that either resell API access they bought from official providers or provide access to models they run locally. The platform supports a wide range of models, including free versions, which may include distilled models or model ensembles. For example, they have a distilled version of DeepSeek-R1 which converts the original model into a more compact and computationally efficient architecture. Model distillation involves creating a smaller, more efficient version of a large language model and transferring the knowledge of the main model to the smaller one. A potential limitation of using OpenRouter is increased latency from certain providers, particularly those offering free access. In this implementation, the DeepSeek-R1 model was utilized. To access OpenRouter models in code, the OpenAI library in Python is used. OpenRouter provides code examples for this on the web page. While the API connection follows a similar process, message handling differs in this context. For this, a list of messages and corresponding sender roles must be maintained, typically designated as "user" for human inputs and "assistant" for model outputs. This list is passed to the LLM each time and a response is returned, after which the response is appended to the list. The response is an object and the message is accessed from the object. This approach enables the simulation of a complete conversational exchange. The task prompt is now read from a .txt file, as with the Gemini implementation, and is included as an initial message at the start of the conversation. The temperature and top-p can be added to each request.

Locally. The second method is running the models locally. This is achieved using Ollama [4], a free software tool that provides access to a broad range of LLMs to download and run locally, including custom models, such as distilled variants. Most models are available in multiple versions resulting in variations in model size and performance. This makes it feasible to run models on less powerful hardware, though potentially with reduced performance. On Windows systems, the user must first download and install the Ollama software, which is available via the official website and integrates Ollama into the command-line interface. Through command-line instructions, users can subsequently download specific models, run them, define custom models and upload them to a registry. To add customizations to the models, custom models can be constructed from existing ones using a Modelfile that contains necessary information such as the base model, system instructions and model parameters. In this implementation, custom models were created of both DeepSeek-R1 and Llama3.1, with the task prompt as system instructions and the temperature set to 0.1. The DeepSeek model has 7 billion parameters and is 4.7GB in size. The Llama one has 8 billion parameters and is 4.9GB in size. They have a note on their Github page saying that at least 8GB of ram is needed for 7 billion parameter models, at least 16GB for 13 billion and at least 32GB for 33 billion. The development system used

[3] https://openrouter.ai/
[4] https://ollama.com/

contains 16GB of RAM, though it concurrently runs other software. Accordingly, models in the 7–8 billion parameter range were selected. The models can be run in the command line itself or may alternatively be accessed through an API. This can be done in Python with either the OpenAI library or Ollama's own library. In this implementation the Ollama library is used, which allows messages to be sent to a specific model and the response to be retrieved. Conversational interaction follows the same procedure as in the OpenRouter implementation, which means the user needs to keep a list of all messages sent along with the sender. Unlike other APIs, no explicit connection initialization is required. The `chat()` function is called with the model and message list as parameters. This function returns the response as an object from which the message can be extracted.

5.3 Simulation

To simulate and evaluate robot navigation with LLM control, a virtual environment, a movable robot model, and a communication interface with the LLMs are required. This section outlines the design of the simulation in Unity, including the environment setup, robot movement mechanics, and obstacle handling.

5.3.1 Environment and Scripts

The simulation environment was developed using Unity [5], specifically version 6000.0.26f1. The environment is a model of a hotel room from a publicly available package in the Unity Asset Store [6]. A preconfigured room model from the package was selected and modified by removing furniture to increase the available movement area. This configuration is illustrated in Figure 5.1 (a). The model was selected due to its relatively large navigable area and its enclosed structure, which realistically simulates an indoor room environment. Its rectangular geometry allows for uniform movement possibilities along multiple directions. Two main C# modules are used in the implementation: one for the movement of the robot, and one for the socket communication. The robot movement module handles the movement of the robot and has functions to move the robot forward, backward, make it turn smoothly and request the coordinates of the robot. The socket module handles the connection with the Python code that connects to the LLM. The module receives commands and invokes the corresponding functions in the robot movement module. Upon completion of the executed movement, the socket module creates a response and sends the response back to the Python code. The response includes either the robot's updated coordinates or a confirmation of the executed command.

5.3.2 Robot Model and Moves

For the robot, a 3D representation of the TurtleBot3 was implemented, which is shown in Figure 5.1 (c). Moving the robot forward or backward is achieved by computing a three dimensional vector that represents the destination with the following formula: "$destination = currentPosition + currentOrientation * distance$", and moving the robot to that destination. The current position is queried from the "RigidBody" of the model with `rigidBody.position`, which returns a three dimensional vector. The current orientation is queried from the "Transform" of the model with `transform.forward`, which similarly returns a three dimensional vector. Lastly, the distance is provided in the parameters and is a float value. This approach ensures linear movement without deviation. Turning is done by calculating the target rotation with "$targetRotation = currentRotation * Quaternion.Euler(0, degrees, 0)$", and turning to that rotation. In this case, all values are quaternions. The degrees in the second factor of the multiplication are the relative degrees the robot needs to turn, represented as a float value. For relative turning the input rotation values are applied directly, while for absolute turning the shortest turn to the destination is calculated first. The calculation is done with the

[5] https://unity.com/

[6] https://assetstore.unity.com/packages/3d/props/interior/hotel-room-collection-214335

`DeltaAngle()` function from the Mathf library. This means the robot model turns around its center, allowing it to turn in place perfectly. The TurtleBot3 in real life is not able to perfectly do this, as the wheels are positioned more forward on the robot. To make sure the movement looks smooth and the robot does not spontaneously transition to the target position, linear interpolation is used. This entails moving the robot in small increments at a constant rate until the destination is reached. Linear interpolation is used for both moving in a straight line and turning. Unity has built in functions `Lerp()` (straight movement) and `Slerp()` (turning) for interpolation. In the grid-based movement scenario, a formula is used to test if the current orientation is diagonal in order to know when to move forward $\sqrt{2}$ meters instead of 1 meter. The formula is: "$(currentRotation.eulerAngles.y/45)\%2 != 0$", which checks if the current orientation is an odd multiple of 45 degrees. In contrast, for the free movement, the length of the movement is decided by the input. When using the Unity orientation convention, turning is simply done with the input degrees. However, for the mathematical orientation convention, the input degrees first have to be translated to Unity's orientation convention. This consists of multiplying the degrees by -1 for relative turning and subtracting them from 90 degrees for absolute turning. The coordinates are rounded to a precision of 2 decimals for the free movement, resulting in a positional precision of approximately one centimeter.

To simulate the real world situation, the errors of the TurtleBot3 where simulated. This was done by adding a random error to the movement of the robot in Unity. The maximum size of the error was calculated based on the size of the movement. For moving straight, the error has a maximum size of 5% of the distance. For turning, the maximum size is the minimum of 10% of the turning distance and 10 degrees. Based on this maximum size, a random value is chosen between the negative and positive of the maximum size, meaning the robot can move too far or not far enough. There was no error added to the path of the robot, ensuring that the trajectory remained linear. This makes it not an exact replica of the real world situation, but sufficiently accurate to make the LLMs run into the same issues.

5.3.3 Obstacles

For the obstacles, cylinders were used with a height of 1 meter and a diameter of 1 meter. Cylindrical obstacles were chosen instead of rectangular ones to enable diagonal traversal past them. For example, if the robot is at coordinate (0,0) and there is an obstacle at (0,1), the robot should still be able to move diagonally to (1,1). With rectangular obstacles this could result in collisions or graphical clipping through the edge of the obstacle, due to the width of the robot's model. Representative examples are shown in Figure 5.1 (b). This makes it so obstacles on coordinates next to each other do not leave any space in between, to make it similar to an actual wall. In the grid based movement scenario, a `Physics.Raycast()` was used to check if the coordinate the robot was trying to move to contained an obstacle or not, which decided whether the move would be executed or not. In contrast, in the free movement scenario, the robot would move until there was a collision with an object, which was similarly checked with a ray cast, but after each increment of the interpolation.

5.3.4 Python Simulation

A simplified simulation for the distance-based movement scenario was also implemented in Python. This secondary implementation was developed to enable rapid prototyping and testing of movement logic, particularly in contexts where full physics or visual feedback was unnecessary. The Python simulation mirrors the functional behavior of the Unity environment but omits graphical components, focusing instead on state updates and command processing. It maintains the robot's position and orientation and displays this information in the terminal after each action. A starting position and a target destination can be predefined within the code. Based on input commands (e.g., forward, backward, turn), the robot's state is updated accordingly. For turn commands, the specified angle is added to the current orientation. For forward and backward movement, geometric calculations are required: the current orientation (in degrees) is first converted to radians, and movement along the x- and y-axes is computed using the cosine

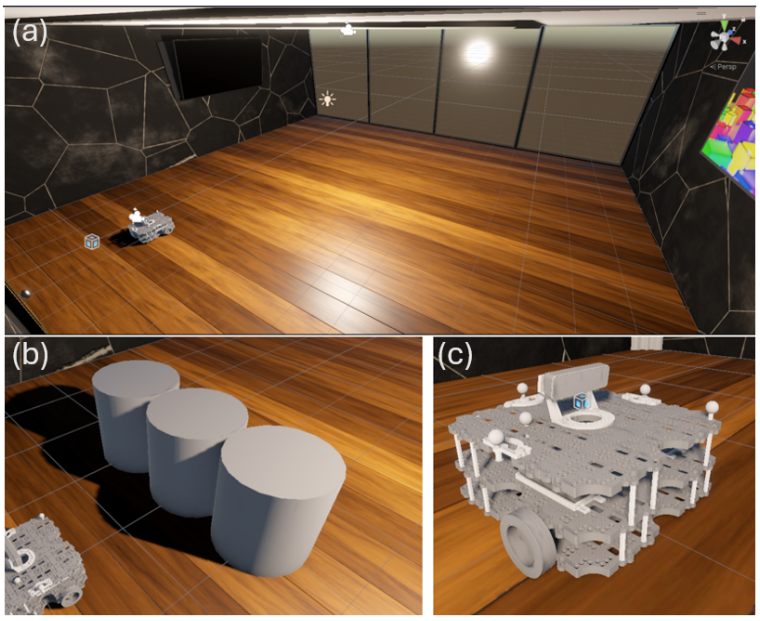

Figure 5.1: Assets used in the Unity simulation: (a) the hotel room environment where the robot operates, (b) three obstacles arranged in a row, and (c) the TurtleBot robot model.

and sine functions, respectively. These values are then added to or subtracted from the robot's current position, depending on the direction of motion.

5.4 Real-World Setup

This section outlines the setup used for conducting the real life experiments. It describes the TurtleBot3 platform and its software configuration, including how movement is controlled using ROS. It also details the motion capture system used for tracking the robot's position, and explains how various components of the system communicate with each other.

5.4.1 TurtleBot3 Configuration and Control

The TurtleBot3 is equipped with a Raspberry Pi 3 running an Ubuntu Linux distro. It also has Robot Operating System (ROS) [7] installed, which is a set of frameworks for robot software development. ROS provides prebuilt packages for functionalities such as sensor integration, robot control, navigation, and perception.Additionally, ROS decouples software from hardware, allowing code developed for one robot to be reused on another with minimal modification. This simplifies the process of writing code for the TurtleBot3 and also makes it reusable.

The TurtleBot is controlled through Python code by publishing Twist messages to a designated topic within the Robot Operating System (ROS) framework. These Twist messages define the robot's motion by specifying linear and angular velocity vectors in three dimensions. The on-board motion controller continuously listens to this topic and interprets the incoming messages to execute movement commands. Listing 5.1 presents the `Move()` and `Turn()` functions used to control the TurtleBot3. Linear motion, both forward and backward, is achieved by assigning positive or negative values to the linear velocity component along the x-axis. Rotational movement is executed by adjusting the angular velocity around the z-axis: a positive value induces a left turn, while a negative value causes a right turn. This turning behavior is realized by driving the wheels in opposite directions. To initiate movement, the appropriate velocity values

[7] https://www.ros.org/

are set and the message is published to the topic. To stop the robot, all velocity components are reset to zero and the message is republished. The speed for both moving in a straight line and turning was set to half of the maximum value. This value was initially selected for speed testing, but was retained for all experiments, as it was well-suited to the dimensions of the test environment. Moving forward a specific distance was done by calculating the time to move by dividing the distance by the speed. After starting the movement, the `sleep()` function was executed for the calculated time, after which the movement was stopped. The TurtleBot3 does not consistently achieve the intended displacement due to wheel acceleration, slippage, and motor inaccuracies. A scaling factor was added to the distance based on measurements of the error to make the moves as accurate as possible. However, this error varies between executions. Additionally, the error in the motors can cause speed differences in the wheels, causing the TurtleBot3 to not move perfectly in a straight line. As a result, perfectly accurate motion cannot be guaranteed.

```
compensation_factor_move = 1.05
def move(node, distance, forward):
    # set speed and calculate duration
    if forward:
        speed = WAFFLE_MAX_LIN_VEL / 2
    else:
        speed = -WAFFLE_MAX_LIN_VEL / 2
    duration = (distance / speed) * compensation_factor_move
    # Create and publish a Twist message (start moving)
    twist = Twist()
    twist.linear.x = speed
    twist.angular.z = 0.0  # No rotation
    node.pub.publish(twist)
    time.sleep(duration)  # Move for the calculated time
    # Stop the robot
    twist.linear.x = 0.0
    node.pub.publish(twist)

compensation_factor_turn = 1.07
def turn(node, angle):
    # set speed and calculate duration
    speed = WAFFLE_MAX_ANG_VEL / 2
    angle_radians = math.radians(angle)
    time_required = (abs(angle_radians / speed))
    time required *= compensation_factor_turn
    # Create and publish a Twist message (start turning)
    twist = Twist()
    twist.angular.z = speed if angle > 0 else -speed  # left or right
    node.pub.publish(twist)
    time.sleep(time_required)  # Wait for turn completion
    # Stop turning
    twist.angular.z = 0.0
    node.pub.publish(twist)
```

Listing 5.1: Python functions for controlling robot movement. The move() function handles forward and backward motion, while the turn() function controls left and right rotation.

The TurtleBot3 has to be initialized first before it can move, which consists of running ROS and running the Python code. This is achieved by establishing SSH connections with the TurtleBot3. To run ROS, the following command is executed in the terminal: “ros2 launch turtlebot3_bringup robot.launch.py”. To run the Python code, the code first has to be downloaded to the TurtleBot3. This is done by setting up an SSH agent on the TurtleBot3 and connecting a Github account to the agent. From there, the code can be pulled and can be run from the terminal.

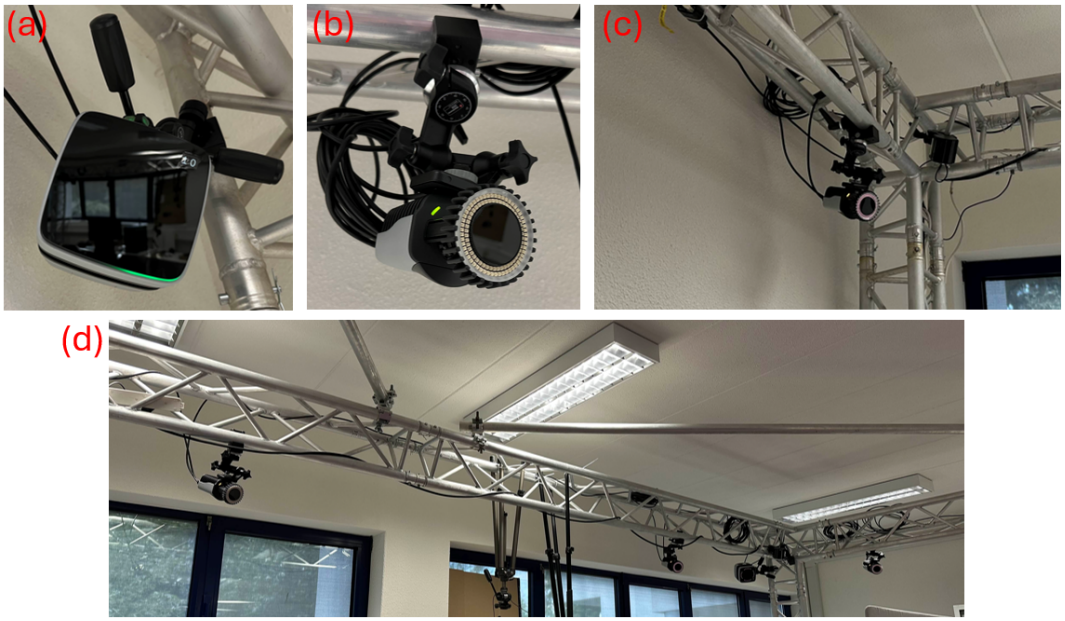

Figure 5.2: Setup of the Qualisys motion capture system in the room: (a) Arqus A5 camera, (b) Miqus M3 camera, and (c–d) cameras mounted on scaffolding within the environment.

5.4.2 Qualisys Motion Capture

The Qualisys motion capture system [8] was used to track the position of the TurtleBot. The setup includes a total of 11 cameras: eight Miqus M3 cameras, which are Qualisys' standard model (Figure 5.2 (b)), and three Arqus A5 cameras, which are a more advanced model (Figure 5.2 (a)). These cameras are suspended from scaffolding surrounding the tracking area, as shown in Figure 5.2 (c) and 5.2 (d). The system is configured and operated using the Qualisys Track Manager (QTM) software, which enables calibration, recording, and real-time tracking. Initially, the cameras are connected via Ethernet by joining the same local network. Calibration is performed using a calibration wand that is moved throughout the environment to align the camera system. Once calibrated, the TurtleBot3, equipped with reflective tracking markers, is placed in the area. Within the QTM software, these markers are grouped to define a "rigid body" allowing the system to recognize and continuously track the TurtleBot as a unified object. A comparison between the virtual scene in QTM and the physical setup is shown in Figure 5.3. Position tracking requires only a single marker, as it provides sufficient spatial data to determine coordinates. However, orientation tracking is more complex due to the symmetric, non-directional nature of individual markers. A single marker provides only a point in space, and two markers form a line, which offers limited orientation information that depends on consistent marker identification. Reliable 3D orientation requires at least three non-collinear markers arranged in a uniquely identifiable shape, such as an "L" or an irregular triangle, ensuring that the configuration appears distinct from all possible viewing angles and enabling re-identification after temporary occlusion. Although the TurtleBot is outfitted with multiple markers for broader use cases, only position data is utilized in this implementation. The orientation must be inferred by the language models themselves. The marker located at the center of the TurtleBot is used for tracking, ensuring consistency with the virtual simulation in Unity. Accessing the positional data in code is achieved using the Qualisys Track Manager SDK for Python. This library provides functionality to connect to the tracking system, control recordings, and continuously stream data packets that include marker coordinates and other tracking information.

[8] https://www.qualisys.com/

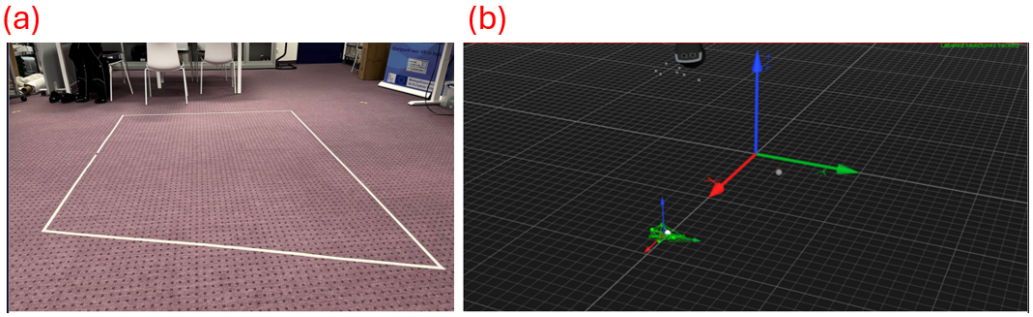

Figure 5.3: The TurtleBot3's operating area in the real world and its representation in Qualisys Track Manager: (a) the physical environment, and (b) the QTM interface showing the TurtleBot3 at the bottom of the tracked space.

5.5 Communication Between LLMs and Robot

To enable communication between the LLMs and the robot, a connection must be established between the Python code handling the LLM interaction and either the simulation or the physical TurtleBot3. While Unity (used for the simulation) supports C# and can directly access LLM APIs, it lacks native terminal input functionality. Implementing user input in Unity would require the creation of a canvas-based input interface. To avoid this added complexity and improve code reusability, a separate Python script running on the same laptop is used to handle communication with the LLMs. Communication between the Python code and the Unity simulation is achieved using TCP socket scripts. In this setup, the Unity C# code initializes a TCP server that listens for incoming connections, while the Python script acts as the TCP client. Commands generated by the LLM are sent from Python to Unity, where the robot executes the corresponding action. After completion, Unity responds with either a position update or a confirmation of a turn. A visual overview of the data flow in the simulation is illustrated in Figure 5.4 (a).

For the real-world setup, the same Python script is used to interface with the LLMs. It also runs on the laptop, as the TurtleBot3's limited computational capabilities necessitate a lightweight codebase on the robot itself. Moreover, the connection to the Qualisys motion capture system requires an Ethernet interface, which the TurtleBot3 does not support, further justifying that all LLM and tracking-related code should run on the laptop. In this configuration, the TurtleBot3 runs only a basic TCP socket server and movement control code. The laptop runs a corresponding socket client to send movement commands. Once the TurtleBot3 completes an action, it responds with a "done" message. The laptop then queries the Qualisys system for the updated position and forwards this information back to the LLM via the API. Figure 5.4 (b) gives a visual overview of the data flow in the real-world setup.

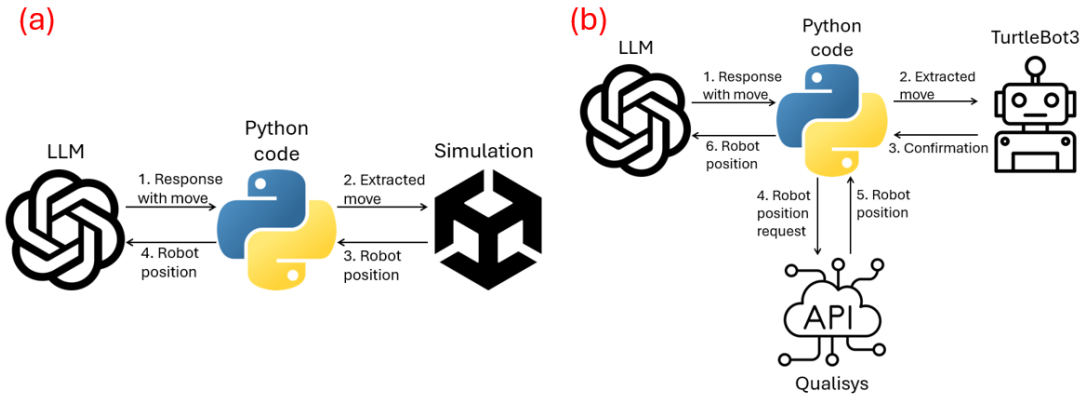

Figure 5.4: Overview of the data flow architecture in both simulation and real-world implementations: (a) data flow within the simulation environment, and (b) data flow in the physical system setup.

Chapter 6

Evaluation

This chapter provides an overview of all the evaluations conducted during this study, beginning with insights from iterative prototyping. These initial exploratory tests helped identify key challenges and informed the design of the final quantitative experiments. The chapter then outlines the methodology of the quantitative tests, covering test design, evaluation metrics, and measures taken to ensure consistency and fairness. Finally, the various test scenarios are described alongside their corresponding results, offering a data-driven perspective on LLM performance across diverse navigation tasks and conditions.

6.1 Exploratory Results

Before designing the quantitative tests, a series of exploratory experiments were conducted to better understand the capabilities and limitations of LLMs in navigation tasks. Prompting strategies, output formats, and scenario configurations were iteratively refined based on these findings to ensure the evaluation would meaningfully capture model behavior under realistic constraints.

6.1.1 Overview of Evaluated Models

In addition to ChatGPT and Gemini, experiments were conducted using DeepSeek and LLaMA models. DeepSeek-R1 was evaluated both via OpenRouter and through local deployment, while LLaMA3.1 was tested only locally due to the lack of freely available full-scale versions with sufficient request limits on OpenRouter. Results from locally running both models were generally poor, as they consistently failed to adhere to the required output format and exhibited highly unpredictable behavior. This was likely caused by the use of smaller model variants, constrained by the limited computational resources of the local testing setup. In contrast, DeepSeek-R1 accessed through OpenRouter showed more promising results; however, it exhibited significant latency, requiring over ten minutes to complete simple tasks. In one more complex scenario, the test was terminated after 45 minutes due to excessive runtime. Given this performance, the current version of DeepSeek-R1 is not suitable for real-time robotic navigation applications, primarily due to its slow response time. Therefore, further experimentation focused solely on ChatGPT and Gemini.

6.1.2 Output Format

The output format was tested under two conditions: without reasoning, where LLMs provided only movement commands, and with reasoning, where they could freely think in their responses. Without reasoning, performance was generally poor, meaning most tests failed. The LLMs struggled particularly with moving downward in the coordinate system and when initially

misaligned with the destination. Success was limited to very simple grid-based tasks. Additionally, they had difficulty with turning directions, often confusing left and right, and failed to avoid obstacles entirely. Attempts using "clockwise" and "counter-clockwise" terminology did not improve results, while signed degree values yielded slightly better outcomes. The best performance without reasoning was observed using absolute turning, especially when directions were named based on coordinate axes (e.g., "positive-x"), which aligned well with spatial reasoning about coordinate changes. In the distance-based movement scenario, the LLMs were unable to reach the destination and repeatedly got stuck in loops trying the same directions. However, when reasoning was permitted, the LLMs' performance improved dramatically. These results are discussed in the following sections.

6.1.3 Orientation

Initially, experiments used Unity's orientation convention since the simulation ran in Unity, eliminating the need for conversion. However, LLMs frequently made errors with turn directions and degree values despite detailed explanations of the Unity system. Switching to the standard mathematical orientation convention significantly reduced these errors, likely due to its more widespread use and representation in the LLMs' training data. LLMs demonstrated strong proficiency in calculating the orientation of the robot, meaning and this approach was used throughout most experiments.

A common error observed was the calculated orientation often being incorrect by exactly 180 degrees, due to how LLMs applied the arctangent calculation. The typical formula computes the displacement vector $(x, y) = (x2 - x1, y2 - y1)$, then calculates the angle as $\theta = arctan(y/x)$. However, the basic arctangent function only returns angles correctly in the first and fourth quadrants, requiring quadrant correction for vectors in the second or third quadrants. For example, a move from $(1, 0)$ to $(0, 1)$ yields a displacement vector of $(0-1, 1-0) = (-1, 1)$ and and angle of $arctan(1/-1) = -45 = 315$ degrees, but must be corrected by adding 180 degrees to get the true orientation of 135 degrees. The LLMs often omitted this correction, resulting in systematic 180 degree errors. Adding an explicit reminder in the task prompt to consider the quadrant when interpreting arctangent results significantly reduced this issue.

6.1.4 Turning Methods

With reasoning allowed, both relative and absolute turning methods yielded strong results. Absolute turning was slightly more reliable, as LLMs occasionally lost track of their orientation during extended movement sequences when using relative turning. Among the relative methods, specifying signed degree values performed marginally better than using directional labels. For absolute turning, performance was consistent across all naming conventions.

In the physical tests with TurtleBot3, movement inaccuracies accumulated over time, leading to increasing deviation between expected and actual orientation. This affected both relative and absolute turning methods, which eventually failed under these conditions. Importantly, such errors were often a secondary effect of output hallucinations, which caused excessive and unnecessary movement steps. In ideal conditions, only one or two correct steps were required to reach the target.

6.1.5 Obstacles

Obstacles posed a major challenge in both grid-based and free-movement scenarios. LLMs were only able to avoid a single obstacle directly in front of the robot; when multiple obstacles (e.g., walls) were introduced, they frequently got stuck in loops, retrying the same blocked paths and often hallucinating. In some cases where execution was not forcibly terminated, Gemini eventually stopped attempting the task and explicitly stated that it was unable to complete it, often accompanied by an apology. Whether obstacles were known or unknown made little difference, highlighting a lack of memory and backtracking capability. To address this, the

full path history was included in the input, along with explicit instructions to avoid revisiting previous locations. However, this strategy had no noticeable effect, as LLMs largely ignored both the path and the instruction. As a result, the path history method was excluded from the quantitative tests.

6.1.6 Distance-Based movement

While earlier sections addressed general challenges across scenarios, the distance-based movement scenario exposed distinct error patterns that required targeted prompt modifications. A recurring issue was inconsistent handling of direction changes. For instance, after turning right and observing an increased distance, the LLM would often reverse the turn instead of exploring further in the same direction. Afterwards, they would reverse the direction again, resulting in loops. However, switching direction when the distance increased is not necessarily a problem and can even be a good strategy, but staying consistent after a switch is crucial. Another frequent error involved accepting equal distance readings as progress, which led to circling around the goal without approaching it. To address this, instructions were added to the prompt directing the model to backtrack and try a new direction when the distance remained unchanged. While these changes reduced some errors, they did not fully resolve the underlying issues.

6.2 Methodology for Quantitative Tests

The primary LLMs used in this study are GPT-4o (ChatGPT) and Gemini-2.0-Flash. These models were chosen for the majority of the experiments to ensure comparability across test conditions. DeepSeek-R1 and LLaMA3.1 were excluded from the quantitative evaluation due to poor performance of the locally run LLaMA and the excessively slow response times of DeepSeek-R1 when accessed via OpenRouter. For all experiments, the temperature and top-p parameters were set to 0.1 to reduce variability and promote consistent, deterministic outputs. Additionally, both models were explicitly instructed, and encouraged, to use reasoning in their responses, as previous findings showed this significantly improved performance. Each test was conducted using both models in both simulated and real-world environments. To eliminate the effect of memory or context accumulation, each individual test was run in a new conversation. This ensured that both LLMs started from the same knowledge baseline for every task. Given the known inconsistency of LLM behavior, all tests were repeated three times per model, with the exception of the orientation-specific experiments. Performance was evaluated using two main metrics: pass percentage and number of moves. Both the overall success rate and the per-task pass rate were recorded to assess the models' general capabilities as well as task-specific consistency. In the free movement scenarios, the final distance to the destination was also measured as an auxiliary performance metric. However, in real-world settings this value is partly affected by hardware inaccuracies, meaning the LLMs do not have full control over this. All experiments within a single scenario were conducted on the same day to reduce the influence of time-based changes, such as LLM updates. Tests across different scenarios were conducted on separate days, which may have introduced minor variability due to potential model updates.

6.3 Scenarios and Results

Tests were conducted in a range of scenarios designed to evaluate different aspects of LLM-guided robotic navigation. These scenarios mirrored those used in the exploratory phase of the study and include: orientation estimation, grid-based movement, free movement, and distance-based movement. For each scenario, multiple test cases were developed to evaluate model performance across varied conditions. At the end of the chapter, there is a brief discussion on the response times of both LLMs.

6.3.1 Figuring Out the Orientation

The first series of tests evaluates the LLMs' ability to compute the robot's orientation based on a given start and endpoint. This was tested using both grid-based and free movement setups. Two variations of tests were conducted: one where the LLMs were instructed to write out the full orientation formula and one where only the final orientation value was required. All orientation tests were conducted in simulation, as the TurtleBot3 in the real world does not move in perfectly straight lines, leading to unavoidable deviations in final orientation. Nonetheless, free-movement tests in the real world still served to verify orientation accuracy by checking whether the robot correctly reached the intended destination. In the grid-based scenario, tests included both orthogonal and diagonal orientations, with and without obstacles. If an obstacle was present, the LLM was expected to adjust its movement through relative turning, as absolute turning was not feasible without knowledge of the robot's orientation. If the robot had to turn, the LLM was expected to report the new facing direction, not the original one. In the free-movement scenario, both orthogonal and diagonal orientations were tested as well, though in this case the diagonal values could be arbitrary rather than limited to multiples of 45 degrees. Performance was measured as the percentage of correctly passed tests.

Grid Movement. Ten configurations were tested under grid-based movement where the full orientation formula was included in the response. Six of these had no obstacles, and four included an obstacle. Of the ten, seven were orthogonal cases and three were diagonal. Each test was executed once. Both ChatGPT and Gemini achieved a 100% success rate in this setting. A notable difference was observed in obstacle handling: ChatGPT consistently turned 45 degrees to the right, whereas Gemini opted for a 90-degree right turn.

In the version of the test where the LLMs only provided the orientation result (without showing the formula), three configurations were used: two orthogonal and one diagonal. Each configuration was repeated three times to evaluate consistency. Again, both models achieved a 100% success rate across all tests. The only distinction was in how orientation was expressed: in one case, ChatGPT returned 315 degrees while Gemini returned –45 degrees, which represent the same orientation. Overall, omitting the formula did not negatively impact performance in grid-based movement.

Free Movement. For free movement, ten unique setups were tested with the formula included in the response. Two of these featured orthogonal orientations, while the remaining eight had arbitrary angles. One setup included an obstacle. Each test was performed once. ChatGPT achieved a 70% success rate. However, all failures stemmed from incorrect task execution rather than computational errors. Specifically, ChatGPT would sometimes predict its location without actually issuing a movement command, e.g., "If I move forward, I assume my location will be (x,y) and therefore my orientation is x degrees." This indicates a failure to follow instructions, not a failure in orientation calculation. Additionally, one test failed due to improper output formatting, although the orientation itself was correct and thus was still counted as a success. Gemini achieved a 90% success rate, with fewer errors related to instruction-following. The single failure involved reporting the robot's initial orientation rather than the updated one after obstacle avoidance. Both models showed minor discrepancies (1–2 degrees off) in a few tests, likely due to rounding errors in trigonometric calculations.

When the formula was omitted, three diagonal setups were tested and repeated three times each. ChatGPT correctly passed all iterations, with two of the cases showing a deviation of about 2 degrees. Gemini correctly answered two out of the three test setups. In the third case, where the correct orientation was 246 degrees, Gemini consistently answered 225 degrees, likely rounding to the nearest 45-degree multiple. When this same test was immediately repeated with the formula shown, Gemini provided the correct response, indicating the error was due to estimation rather than reasoning failure.

Conclusion. The results demonstrate that both ChatGPT and Gemini are perfectly capable of accurately calculating robot orientation in both simple and complex conditions. Additionally, both models were able to adjust orientation in response to obstacles. While presenting the full formula slightly improves consistency, accurate results can still be obtained without it, especially with grid-based movement. The only errors with the formula written out were failures to follow the instructions. These failures may stem from factors such as instruction complexity, model configuration, or recent changes in model behavior due to updates.

6.3.2 Grid-Based Movement

The grid-based movement scenario assesses the LLMs' ability to navigate a simplified, structured environment. Tests were conducted both with and without obstacles, and each case was repeated three times for consistency. Navigation was evaluated using both relative turning (positive/negative angle conventions) and absolute turning (using degree values), with the same test cases applied to each method. In some scenarios, the robot was initially oriented toward the destination, while in others, it began facing a different direction. Tests included both orthogonal and diagonal initial orientations. The complexity of the cases varied: some involved a direct path to the goal, while others required multiple turns. Notably, in every test, the LLMs had to determine their orientation independently. As a result, the first move was always executed in the direction the robot initially faced. Obstacle tests were limited to three configurations, as the specific direction of obstacle approach was deemed irrelevant. These cases tested different numbers of obstacles and included both known and unknown obstacle setups. The performance metrics were the percentage of successfully passed test cases (i.e., reaching the correct destination), consistency across repeated trials, and path length, as efficiency is also critical in navigation. All tests were carried out in simulation.

Without Obstacles. Ten distinct test cases without obstacles were evaluated. Of these, seven involved orthogonal starting orientations, and three involved diagonal ones. Three cases had the robot already facing the destination, while the remaining seven required at least one turn. In six cases, the destination could be reached in a single straight line; the other four required directional changes.

Using relative turning, both ChatGPT and Gemini reached the destination in at least one of the three attempts for each test case. Failures were rare: ChatGPT failed one attempt in three separate test cases, while Gemini failed one attempt in two cases. All other attempts across test cases were successful. These results can be seen in Table 6.1. ChatGPT's failures were exclusively due to output formatting errors. For example, some moves were not enclosed within the required "$" delimiters, causing execution failures in the code. In contrast, Gemini's failures were due to logical errors. In one instance, Gemini hallucinated success by responding with "Done" even though the robot had not yet reached the destination. Both LLMs followed the most optimal path in the same four cases, those where the destination could be reached via a direct line. In the remaining six cases, there was a mix of optimal and suboptimal paths. ChatGPT often avoided diagonal movement in these situations, instead moving the x and y distances separately. For instance, to go from (0,0) to (1,1), it would first move to (1,0) and then to (1,1). Gemini occasionally made similar non-diagonal moves, but also exhibited more severe inefficiencies, taking unnecessarily long detours, albeit still reaching the goal. Illustrative examples of these behaviors are shown in Figure 6.1. Subfigures (a), (b), and (c) correspond to one scenario, while (d), (e), and (f) belong to another. In the first scenario, the optimal path is shown in (a). ChatGPT overshot the destination, then moved back down as shown in (b), while Gemini traveled the x and y distances separately, shown in (c). In the second scenario, the optimal path is shown in (d). ChatGPT again avoided diagonal movement as shown in (e), and Gemini took a peculiar detour, first moving up-right and then down, illustrated in (f). Additionally, Gemini occasionally began by turning in the wrong direction but corrected itself in subsequent steps.

The results for absolute turning closely mirrored those of relative turning and are displayed in

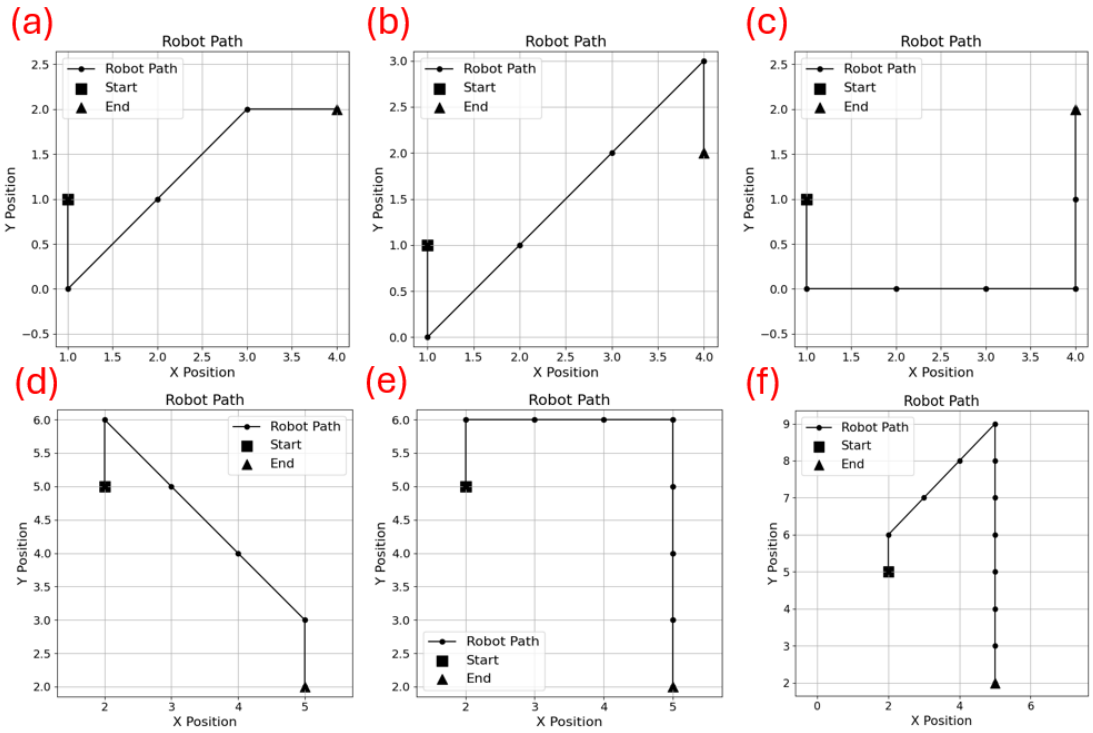

Figure 6.1: Traveled paths of the robot in two grid-based navigation test cases. The x- and y-axes represent position coordinates, and each line shows a path taken by the robot. The starting point is marked with a square, and the endpoint with a triangle. (a) Optimal path in the first test case, (b) path with an overshot diagonal movement in the first case, (c) non-diagonal path in the first case, (d) optimal path in the second test case, (e) non-diagonal path in the second case, and (f) an irregular path in the second case.

	Attempts passed (out of 3)			
	Relative turning		Absolute turning	
Test	ChatGPT	Gemini	ChatGPT	Gemini
1	3/3	3/3	3/3	3/3
2	3/3	3/3	3/3	3/3
3	3/3	3/3	3/3	3/3
4	3/3	2/3	3/3	2/3
5	3/3	3/3	3/3	3/3
6	2/3	3/3	3/3	3/3
7	3/3	2/3	3/3	3/3
8	3/3	3/3	2/3	2/3
9	2/3	3/3	3/3	3/3
10	2/3	3/3	3/3	3/3

Table 6.1: Number of tests passed (out of 3) by ChatGPT and Gemini across multiple scenarios without obstacles in the grid-based movement task with both relative and absolute turning.

Table 6.1. Both LLMs reached the destination at least once in every test case and successfully completed all three attempts in nearly all scenarios. As with relative turning, ChatGPT's errors were mostly due to output formatting mistakes and its occasional failure to take diagonal paths that would yield a more optimal route. Gemini exhibited similar behavior but often compounded mistakes with incorrect initial turns, requiring it to later correct its path. A particularly notable issue in the absolute turning tests was a repeated failure by both LLMs to follow a critical output formatting instruction. They were explicitly instructed to first calculate and output the robot's orientation, wait for a confirmation, and only then proceed with the movement. Despite these clear instructions, and even with an additional reminder at the end of the prompt, both models frequently returned the orientation and the movement in a single response. This oversight was not problematic in relative turning because the LLMs only needed to internally track orientation. However, in absolute turning, this instruction is crucial: the underlying code can use the separately returned orientation to compute the shortest rotation in later moves. In simulation, this did not cause errors since Unity provides the orientation directly. In a real-world deployment, however, this behavior could lead to execution failures, as the current code setup depends on receiving orientation data in a distinct message. While this limitation could be addressed by adapting the code to handle additional output formats, maintaining consistency in the LLMs' responses is essential, as it is not feasible to anticipate and accommodate every possible variation. Interestingly, the prompt instructions were nearly identical for both relative and absolute turning, only the part about turning was different, making the inconsistency even more puzzling. This suggests that minor differences in model interpretation or task framing can significantly affect instruction adherence.

With Obstacles. The obstacle navigation tests used three different configurations, visualized in Figure 6.2. In configuration (a), a single obstacle was placed directly in front of the robot, with the goal located behind it. Configuration (b) involved a row of three obstacles forming a wall between the robot and its destination. Configuration (c) built on (b) by adding two more obstacles above the existing row, creating a U-shaped arrangement that required the robot to navigate around and enter the "U" to reach the target. Each of these tests was executed with both known and unknown obstacle configurations, using only relative turning in all cases. In the first obstacle test, both ChatGPT and Gemini successfully navigated around the obstacle in both known and unknown scenarios. ChatGPT demonstrated higher consistency, completing all three attempts without failure. Gemini failed two attempts by hallucinating, issuing arbitrary, directionless movements. ChatGPT followed clean, predictable paths each time, although it only used diagonal movement for the optimal route in one of the attempts. Gemini, meanwhile,

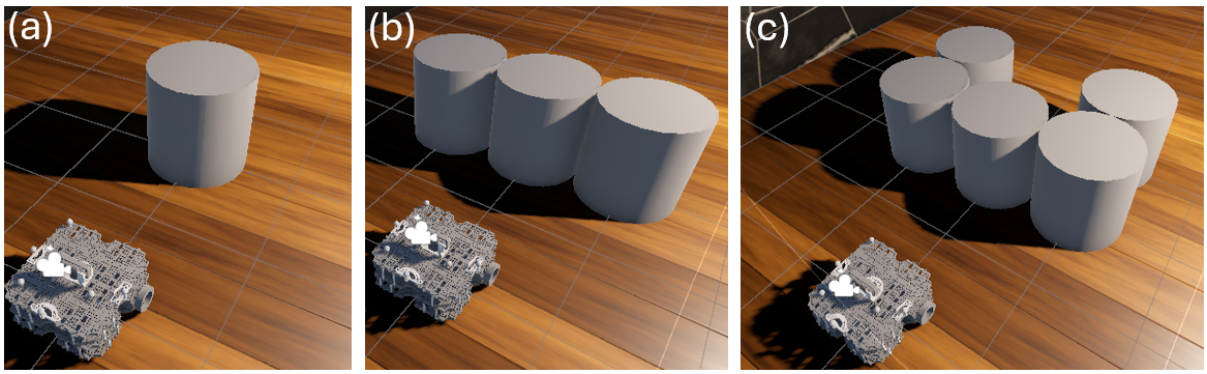

Figure 6.2: Various obstacle configurations used in the simulation environment: (a) a single obstacle, (b) three obstacles placed side by side, and (c) five obstacles arranged in a "U" shape.

never used the optimal diagonal path and frequently introduced inefficient detours. In some attempts, it even began by moving in the wrong direction and had to correct its course later. The only functional difference between known and unknown obstacles was a single extra move required to identify the obstacle in the unknown configuration. For the second and third obstacle setups, neither LLM passed a single attempt. Both models exhibited the same flawed behavior: after detecting an obstacle, they would turn (usually right), encounter a second obstacle, then revert to the original path and repeat the process. This would either result in getting stuck in a loop or hallucinations. This pattern persisted even in the known-obstacle configuration, indicating that awareness of obstacle positions alone is insufficient for more complex reasoning in constrained environments.

Conclusion. The results show that both LLMs can effectively navigate simplified, obstacle-free environments. However, their path choices can vary between attempts, and they often fail to consistently select the most efficient routes. ChatGPT generally outperformed Gemini, with most of its failures attributed to instruction-following issues rather than logic errors. Gemini, on the other hand, was more prone to hallucinations and frequently had to correct itself. The inconsistent instruction adherence in the absolute turning tests, despite nearly identical prompts to the relative turning tests, highlights how small prompt variations or context differences can significantly impact LLM behavior. When it comes to obstacle handling, both LLMs struggled beyond the most basic configuration. They reliably navigated around a single obstacle, but any increased complexity, such as a wall or U-shaped barrier, led to repetitive, looping behavior, even when obstacle positions were pre-defined. The only difference between known and unknown obstacles was an extra move to locate the obstacle. This suggests that reasoning about obstacles in a constrained grid and applying backtracking remains a challenging task for current LLMs.

6.3.3 Free Movement

This scenario evaluates the LLMs in a more realistic setting, where movement is not restricted to a discrete grid and small execution errors are possible. Tests were conducted both in simulation and on the TurtleBot3 in the real world. In the simulation, movements were assumed to be perfectly accurate, while in the real-world tests, physical inaccuracies from the TurtleBot3 were taken into account. Unlike the grid-based scenario, fewer test cases were needed here, as the optimal navigation strategy is straightforward: after determining its orientation, the robot must rotate toward the target and move forward until it arrives. Both relative and absolute turning were tested, with and without obstacles. Each test was repeated three times. In simulation, a tolerance of a few centimeters was accepted due to rounding errors, while in real-world tests, the goal was to get within 0.5 meters of the target due to higher execution noise. The performance

	Attempts passed (out of 3)			
	Relative turning		Absolute turning	
Test	ChatGPT	Gemini	ChatGPT	Gemini
1	3/3	3/3	3/3	3/3
2	2/3	1/3	2/3	3/3
3	3/3	3/3	3/3	3/3
4	2/3	3/3	3/3	2/3
5	3/3	3/3	3/3	3/3

Table 6.2: Number of tests passed (out of 3) by ChatGPT and Gemini across multiple scenarios without obstacles in the free movement task with both relative and absolute turning.

metrics included pass rate, consistency, and final distance to the destination.

Simulation. Five test cases were performed in the simulation. In two of them, the robot started roughly aligned with the destination; in the remaining three, it was oriented differently, including arbitrary angles. Table 6.2 shows the results. ChatGPT and Gemini both achieved 100% pass rates for both relative and absolute turning across these scenarios. However, not each attempt was passed. ChatGPT had three failed attempts: one due to a hallucinated forward-backward loop, and two where it stopped too far from the destination. Gemini also failed three attempts, all due to orientation miscalculations. In these, the LLM misinterpreted the orientation quadrant and was 180 degrees off, causing incorrect initial moves and follow-up hallucinations. It was only able to recover from this error in one case. Most successful attempts reached within 1–5 cm of the target. These minor deviations were accepted, as they result from unavoidable rounding errors in trigonometric calculations. Notably, even in cases where the robot was nearly aligned with the goal, both LLMs sometimes performed a minor corrective turn (e.g., 3°) to improve alignment. ChatGPT occasionally exhibited odd behavior: in one attempt, it approached the target in small incremental steps rather than a single straight move, increasing execution time unnecessarily. In two other cases, it decomposed the navigation into separate x and y movements, echoing its earlier behavior from the grid-based tests, despite being in a free-movement context.

The three obstacle configurations from the grid-based tests were reused here (see Figure 6.2), with separate evaluations for known and unknown obstacles. Only relative turning was used. Performance dropped slightly compared to the grid-based scenario, primarily due to imprecision in obstacle avoidance. Since movement was not confined to grid steps, the LLMs often moved only half a meter sideways when encountering an obstacle. This was typically not enough to avoid another collision with the same obstacle, requiring another corrective move. In some cases, this was handled successfully; in others, the LLMs became confused and began issuing hallucinated or aimless commands. ChatGPT passed five out of six attempts. The single failure involved misidentifying a second obstacle after re-colliding with the first one, followed by hallucinated movement. Gemini failed half of the attempts. Most failures were due to hallucinations after a second obstacle collision, though in one case, the robot simply stopped at the wrong destination. Even in successful cases, Gemini's behavior was less efficient, often turning in the wrong direction after passing an obstacle and then correcting itself. This was due to an internal orientation error during relative turning. Since Gemini's reasoning was correct but it turned in the wrong direction, this issue would likely be resolved by using absolute turning instead. As with grid-based movement, the two more complex obstacle scenarios (the wall and U-shape) failed in every attempt. Both LLMs displayed the same behavior as before, looping through the same paths, failing to try new routes, or hallucinating random navigation steps. This indicates that the challenge posed by multi-step detours in constrained environments remains unresolved in both models, regardless of movement freedom.

Real World. Two test cases were performed in the real-world scenario using the TurtleBot3. In one case, the robot was already oriented toward the destination, and in the other, it was not. The results were similar for both relative and absolute turning. Both LLMs were able to pass the tests, but ChatGPT showed greater consistency, with zero failed attempts. Gemini, on the other hand, failed in a few ways. In one absolute turning attempt, it failed to follow the instructions and instead applied relative turning. Furthermore, when using relative turning, Gemini made the same mistake twice by failing to take the correct quadrant into account when calculating the orientation, resulting in a 180-degree error. This led to the robot initially turning in the wrong direction and moving away from the target. A notable difference in behavior between the models appeared in the test where the TurtleBot3 started already aligned with the destination. ChatGPT recognized that no rotation was necessary and moved straight forward, whereas Gemini still performed a slight turn in each attempt to better align itself, though this did not translate into better accuracy or a closer final position. Another interesting observation was that, in the first test case, both LLMs often made small corrective movements even when already within the required half-meter threshold from the destination. While this behavior was not explicitly forbidden by the instructions, it introduced unnecessary movements and reduced efficiency, especially when the robot was already close enough.

Conclusion. Free movement does not pose a significant challenge to either LLM. Both were able to consistently reach the destination in simulation and real-world tests, even with small inaccuracies in movement. However, as with previous scenarios, obstacles continued to be a major challenge. Both LLMs were able to avoid a single obstacle in some cases, but they consistently failed when faced with more complex obstacle arrangements. Whether the obstacles were known or unknown had minimal effect on performance, aside from an additional move to identify unknown ones. The most persistent issue across all scenarios was the LLMs' failure to fully adhere to instructions, which led to execution errors. Although relative and absolute turning performed similarly overall, some specific failures, such as Gemini's missteps with turning direction, could have been avoided with absolute turning.

6.3.4 Distance-Based Movement

The distance-based scenario tests whether the LLMs can navigate through an unknown environment using only the distance to the destination as feedback. The goal was not just to verify if navigation is possible, but also to determine whether the LLMs could demonstrate spatial reasoning that might allow them to outperform basic search or exploration algorithms. These tests were conducted in a Python-based simulation that tracks the robot's position and calculates the Euclidean distance to the destination. Only relative turning was used in this scenario, as the robot's absolute orientation was not available. Two test cases were used, and each was executed three times for both ChatGPT and Gemini. In the first test case, the robot started directly next to the destination but was not oriented towards it. The starting position was (1,1), the destination was at (2,1), and the robot's initial orientation was 90 degrees, facing upward. The destination was intentionally placed to the right because the LLMs tend to begin turning to the left by default. This setup was meant to assess whether the models could quickly discover the correct direction or if they would inefficiently rotate through all possibilities.

In this first test, both LLMs reached the destination in all attempts. ChatGPT failed to show any efficient strategy in two of the attempts, trying nearly all directions in succession, suggesting a lack of reasoning. The movement paths for these cases are illustrated in Figure 6.3 (a). However, in the third attempt, ChatGPT performed slightly better and reached the destination in fewer moves, as seen in Figure 6.3 (b). Despite the improvement, it still repeated one previously attempted direction unnecessarily. After detecting that turning left from the original orientation increased the distance, it correctly turned right but then re-tested the original direction again, showing it did not fully rule out ineffective options. Gemini, on the other hand, consistently reached the destination in fewer moves than ChatGPT and took the same path across all three attempts. This is shown in Figure 6.3 (c). One slightly odd behavior was

that Gemini never tested the initial orientation of the robot; it immediately rotated to a new direction instead of verifying the one it started in. Nonetheless, its efficiency in discovering the destination stood out.

The second test was a bit more complicated. The robot started at (1,1) with an initial orientation of 135 degrees, facing diagonally to the top left, and the destination was located at (3,4). Because the destination could not be reached in a straight line from the start, this case tested whether the LLMs could form a general understanding of the target's direction and plan a more complex route accordingly.

ChatGPT struggled in this scenario, reaching the destination in only one out of three attempts. However, the single successful attempt was the most efficient among all the attempts from both models. As depicted in Figure 6.3 (d), ChatGPT began by turning right after recognizing that the original direction did not decrease the distance. It then continued forward until the distance stopped decreasing. At that point, it turned left, tested that direction, realized it was ineffective, and then turned right instead. Notably, once at position (1,4), it tested upward and downward movements and recognized that it was moving parallel to the destination's direction. It then made a 90-degree turn to the right, correctly orienting itself directly toward the destination. This path possibly suggests some form of spatial awareness and logical deduction. The other two attempts by ChatGPT failed due to repetitive loops, either retrying the same directions without learning or failing to backtrack when the distance remained constant. In contrast, Gemini successfully reached the destination in all three attempts, albeit with a slightly longer path each time, as shown in Figures 6.3 (e) and (f). While its strategy was less efficient than ChatGPT's best run, it was far more consistent overall. Gemini appeared to apply a basic but effective reasoning process: it would start by turning left, notice the increase in distance, and then decide to turn right instead. This allowed it to quickly converge on the destination using a path that, while not optimal, was quicker than a naive algorithm and reliable. However, this strategy was explained in the task prompt and was therefore not devised by the LLMs themselves.

Conclusion. Both ChatGPT and Gemini demonstrated the ability to reach the destination with limited information, yet their performance did not significantly surpass that of a basic algorithm. The spatial reasoning observed in their responses largely mirrored the strategies outlined in the prompt, indicating that these were not independently developed by the models. Notably, ChatGPT exhibited some capacity for extrapolation beyond the provided instructions, particularly in the second test case, where it executed a 90-degree turn after detecting it was moving parallel to the target, suggesting a degree of situational awareness. Gemini, by contrast, delivered more consistent outcomes across all attempts, reliably reaching the destination through a repeatable and effective path.

6.3.5 Response Time

Throughout all experiments, Gemini consistently responded much faster than ChatGPT. In fact, Gemini's response time was so rapid that a deliberate delay had to be introduced in the code to avoid exceeding the request rate limits. This speed advantage can be attributed to the use of Gemini-2.0-Flash, a lightweight variant optimized for speed at the cost of some reasoning complexity. On the other hand, GPT-4o is designed for broader reasoning capabilities and more accurate outputs, which naturally results in slightly longer response times. This difference in design purpose is reflected in the outcomes as well: while Gemini responded more quickly, GPT-4o achieved better overall performance across the tasks.

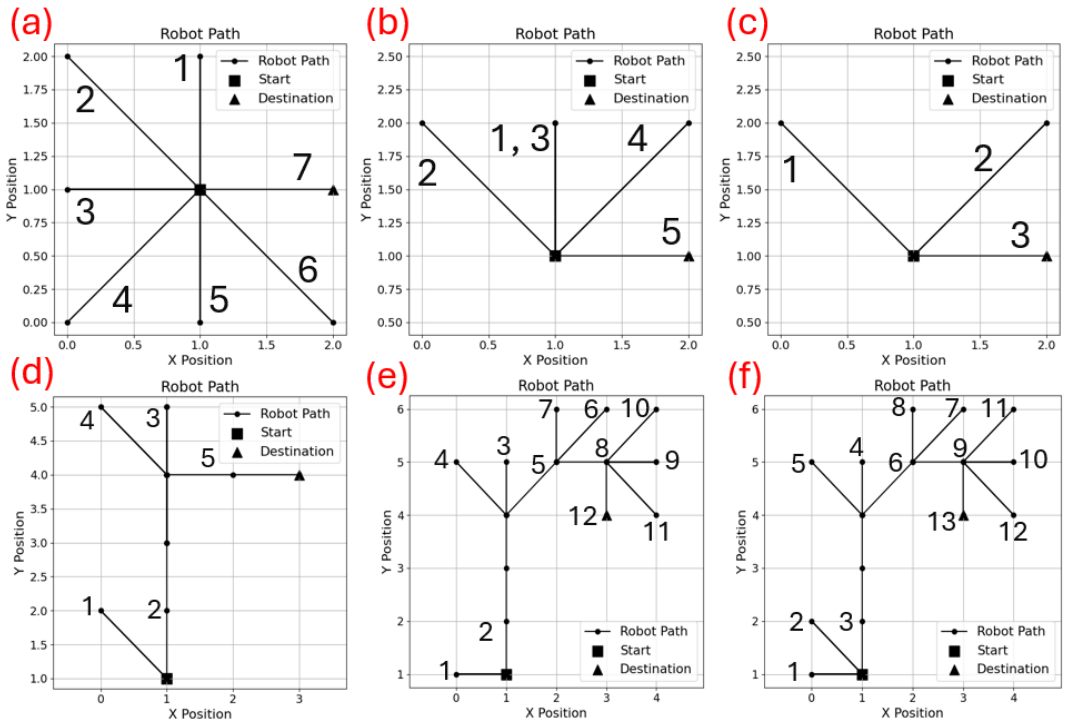

Figure 6.3: Traveled paths of the robot in the distance-based navigation scenario. The x- and y-axes represent position coordinates. Each line indicates the path taken by the robot, with the starting position marked by a square and the endpoint marked by a triangle. The numbers indicate the order of directions followed by the LLM. (a) Non-optimal path generated by ChatGPT in the first test case, (b) improved path by ChatGPT in the same case, (c) path by Gemini in the first test case, (d) path by ChatGPT in the second test case, and (e–f) two separate paths generated by Gemini in the second test case.

Chapter 7

Conclusion & Future Work

This chapter concludes the thesis by summarizing the key findings and addressing the central research question. It also discusses the limitations encountered during the study, outlines directions for future research, and offers a personal reflection on the research process.

7.1 Conclusion

This thesis set out to explore whether large language models (LLMs) can independently navigate mobile robots with minimal external support. The goal was to evaluate how well these models could interpret commands, reason through spatial challenges, and generate executable actions without relying on extensive frameworks or pre-programmed logic. The study followed an iterative approach: initial prototyping was used to refine prompting strategies and identify effective interaction patterns, followed by quantitative testing to assess consistency, accuracy, and overall performance in both simulated and real-world environments.

The results have demonstrated that LLMs are capable of navigating mobile robots using a minimal supporting framework. However, they exhibit significant limitations when dealing with complex environments, especially those involving obstacles or limited contextual information. The initial feasibility tests confirmed that LLMs such as GPT-4o, Gemini-2.0-Flash, and DeepSeek-V3 can interpret navigational commands, generate full movement plans, and operate interactively with step-by-step execution while processing real-time feedback. However, these successes depended heavily on how clearly the task was defined. Without thorough explanations and illustrative examples, the LLMs frequently misunderstood task requirements, particularly the behavior of obstacles and how to respond to collisions or blocked paths. Even with clear task descriptions, all LLMs tested struggled to handle obstacles effectively. They often failed to remember the position of obstacles or previously attempted paths and would repeat ineffective strategies, becoming trapped in navigation loops. Among the models, DeepSeek-R1 showed the most promising reasoning capabilities, displaying a better understanding of environmental constraints. However, its slow response time rendered it unsuitable for real-time applications. A major challenge for all models was the generation of accurate visual representations of the navigation space. Despite having access to the correct position of the robot, the models often failed to place it accurately within generated maps or images. This suggests a disconnect between their internal state tracking and their output formatting capabilities.

Further experimentation, done primarily with ChatGPT and Gemini, emphasized the importance of allowing LLMs to reason openly in their responses. When reasoning and calculations were restricted (e.g., by asking only for final commands), performance declined significantly. Conversely, when allowed to show their reasoning, LLMs performed more reliably across both grid-based and free movement scenarios. Still, major issues persisted. In both scenarios, LLMs could typically only avoid a single obstacle before performance degraded. When multiple ob-

stacles were present, the models often hallucinated new paths or became disoriented. Similarly, optimal pathfinding was a consistent weakness. Models frequently defaulted to inefficient, orthogonal routes in grid-based scenarios, even when shorter diagonal paths were available. From a command interpretation standpoint, LLMs sometimes failed to adhere to output format requirements, leading to integration issues with robot control code. Relative turning commands also introduced confusion, with models occasionally losing track of the robot's orientation, though absolute turning commands slightly improved reliability. In scenarios where only the distance to the goal was known, LLMs showed limited spatial reasoning. In most cases, they relied on a single, prompt-given strategy and failed to develop new heuristics. Attempts to explore the environment often led to repeated errors or direction loops around the destination, problems that a conventional algorithm could solve more efficiently. Lastly, the results consistently highlighted a broader issue: inconsistency. LLMs often produced different results for identical inputs, even with deterministic settings configured. Successful navigation in one attempt might fail in the next, indicating that current LLMs lack the predictability required for reliable autonomous control.

In summary, while LLMs like GPT-4o and Gemini-2.0-Flash can navigate a robot through simple, structured environments using minimal external support, they are not yet robust or consistent enough for reliable, general-purpose deployment. Until LLMs develop more advanced and stable spatial reasoning capabilities, they will continue to require external support, either in the form of reasoning aids, memory modules, or environmental abstraction layers, to perform dependable robotic navigation.

7.2 Limitations and Future Work

This thesis primarily focused on two leading large language models, GPT-4o and Gemini-2.0-Flash, with limited experimentation conducted using DeepSeek-R1 and LLaMA3.1. These models were selected based on API availability and their status as state-of-the-art at the time of testing. However, the field of language models is evolving rapidly, and numerous other models from various providers remain unexplored. Future work could include a broader evaluation of LLMs such as Anthropic's Claude, Meta's LLaMA (in various versions), Mistral AI's models, and xAI's Grok. These models vary in size, architecture, and reasoning capabilities, and continuous updates from developers make repeated benchmarking a worthwhile endeavor. In addition, further testing could include newer versions of models from the same providers. For instance, OpenAI released GPT-4.1 and reasoning-optimized models such as o1 and o3 during the final stages of this thesis. These models offer enhanced reasoning performance but, as observed with DeepSeek-R1, tend to have slower response times. Similarly, Google's Gemini-2.5 may offer improvements in both speed and capability, and should be investigated in future studies.

A key limitation of this research is its focus on short-horizon navigation tasks, which are scenarios in which the robot performs relatively few movements to reach a target. While valuable as an initial proof of concept, these short tasks do not fully reflect the challenges encountered in real-world applications, where robots often operate in larger spaces and over extended durations. In such long-horizon tasks, limitations related to token limits, memory retention, and state tracking become more apparent. For instance, this thesis noted that LLMs occasionally forgot the robot's orientation after multiple steps. Further research should also examine how LLMs perform in long-horizon contexts, including whether it is more effective to maintain a continuous dialogue history or to restart the conversation at logical checkpoints.

Another limitation identified was the difficulty LLMs faced in handling obstacles and using distance-based movement, largely due to the absence of a supporting framework. To improve performance in these areas, future work could explore lightweight alternatives to full multimodal frameworks. For example, using a secondary LLM for verification, integrating basic sensor input to detect obstacles, or incorporating simple user feedback loops. For feedback to be useful, it must be retained across steps without requiring repeated input, which implies a need for lightweight memory mechanisms. If these current limitations of the LLMs could be solved,

expanding the environment's complexity is another avenue of interest. This thesis mainly dealt with static and relatively simple settings. By adding environmental features such as walls, furniture, elevation changes, or even moving obstacles (e.g., humans), the adaptability and robustness of LLMs could be further challenged. These scenarios would require more frequent environmental updates, pushing the limits of the models' spatial reasoning and real-time adaptability.

A practical issue encountered during testing was the LLMs' inconsistent adherence to the required output format. Even when reasoning and decision-making were correct, small formatting errors, such as incorrect syntax and outputting multiple moves in the same message, frequently caused the control system to fail. To address this, future work could explore the use of structured output formats such as JSON, which would impose stricter formatting constraints and facilitate direct parsing by control software. However, care must be taken not to overly constrain the LLMs, as the results indicate that excessive limitations can negatively impact their performance.

Beyond the navigation-focused scope of this thesis, future work could also explore LLM applications in robotic manipulation. Previous research has demonstrated promising results in this domain, though typically with the aid of significant supporting frameworks. Future work could assess whether LLMs can perform manipulation tasks (such as grasping or repositioning objects) under a bare-bones setup similar to the one used here. For instance, by treating a robot arm's workspace as a grid and controlling it from a top-down perspective, an LLM could potentially move to target coordinates and issue basic commands such as "grasp" or "release." While this approach may succeed in simplified settings, its limitations in handling more dynamic and unstructured manipulation tasks would be a critical area to explore.

In summary, this thesis establishes a foundation for LLM-controlled robotic navigation with minimal support. However, many opportunities remain to extend and refine this work through testing additional models, increasing task complexity, exploring long-term memory strategies, and expanding to other robotic capabilities such as manipulation.

7.3 Reflection

Throughout the course of this thesis, I have gained a deeper understanding of the inner workings of LLMs, particularly their capabilities and limitations in non-conversational domains such as spatial reasoning and robotic control. Initially, I was impressed by how human-like these models appeared during everyday interactions and basic reasoning tasks. However, this perception changed significantly once I tested their performance in navigation tasks. Despite their fluency and apparent intelligence in conversation, the models often failed to follow explicit instructions or reason correctly about spatial relationships, even when provided with detailed explanations. This highlighted a critical realization: the reasoning demonstrated by LLMs is often superficial and can break down when applied to domains outside of their training focus. This observation reinforces the importance of treating LLM outputs critically and not assuming genuine understanding or reliability across all domains.

The progression of my research began with strong momentum but eventually slowed as the limitations of the models became more apparent. A major turning point came when I discovered that allowing the LLMs to include their reasoning within their responses significantly improved navigation outcomes. Without such reasoning, the models frequently made errors that rendered navigation tasks nearly impossible. Unfortunately, this insight came somewhat late in the process, resulting in time lost on earlier, less productive approaches. That said, the contrast between reasoning-enabled and reasoning-disabled responses ultimately became a valuable point of comparison in the analysis.

Another challenge during the project was the tendency to refine prompts incrementally in hopes of improving model performance. While prompt engineering is essential, I now recognize that the time spent on minor prompt tweaks might have been better used testing a broader range

of models, including newer versions or alternatives from other providers. This broader benchmarking could have offered a more comprehensive understanding of the current capabilities of different LLMs in navigation contexts.

A further important lesson was the critical value of thoroughly documenting both experimentation findings and test results. Initially, results from the experiments were recorded only in brief textual summaries, which proved insufficient when compiling and analyzing data later during the thesis writing process. This issue was intensified by the frequency of exploratory testing, where informal notes were made about observed issues, but structured data was not collected. Similarly, for the quantitative testing, several tests had to be repeated because critical details, such as the robot's path, specific errors encountered, or reasons for failure, had not been logged properly. This experience highlighted the necessity of maintaining detailed, structured records throughout the development and evaluation phases to support accurate analysis and reproducibility.

In conclusion, the results of this thesis are mixed. On the one hand, LLMs have demonstrated the potential to navigate robots in structured, obstacle-free environments with minimal framework support. On the other hand, their performance is inconsistent and breaks down in the face of complexity, particularly with obstacles and limited environmental information. This reinforces that while LLMs are powerful tools, their deployment in real-world robotics still requires caution, careful design, and, in many cases, additional support frameworks. Further work will be needed to determine how these limitations can be mitigated and what strategies or model improvements can enable more reliable robotic control in the future.

Bibliography

[1] Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., ... Zeng, A. (2022). Do As I Can, Not As I Say: Grounding Language in Robotic Affordances (No. arXiv:2204.01691; Version 2). arXiv. https://doi.org/10.48550/arXiv.2204.01691

[2] Liang, J., Xia, F., Yu, W., Zeng, A., Arenas, M. G., Attarian, M., Bauza, M., Bennice, M., Bewley, A., Dostmohamed, A., Fu, C. K., Gileadi, N., Giustina, M., Gopalakrishnan, K., Hasenclever, L., Humplik, J., Hsu, J., Joshi, N., Jyenis, B., ... Parada, C. (2024). Learning to Learn Faster from Human Feedback with Language Model Predictive Control (No. arXiv:2402.11450). arXiv. https://doi.org/10.48550/arXiv.2402.11450

[3] Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.-H., Arenas, M. G., Chiang, H.-T. L., Erez, T., Hasenclever, L., Humplik, J., Ichter, B., Xiao, T., Xu, P., Zeng, A., Zhang, T., Heess, N., Sadigh, D., Tan, J., Tassa, Y., & Xia, F. (2023). Language to Rewards for Robotic Skill Synthesis (No. arXiv:2306.08647). arXiv. https://doi.org/10.48550/arXiv.2306.08647

[4] Shanahan, M. (2024). Talking about Large Language Models. Commun. ACM, 67(2), 68–79. https://doi.org/10.1145/3624724

[5] Deshpande, A., Rajpurohit, T., Narasimhan, K., & Kalyan, A. (2023). Anthropomorphization of AI: Opportunities and Risks (No. arXiv:2305.14784). arXiv. https://doi.org/10.48550/arXiv.2305.14784

[6] Leiser, F., Eckhardt, S., Leuthe, V., Knaeble, M., Mädche, A., Schwabe, G., & Sunyaev, A. (2024). HILL: A Hallucination Identifier for Large Language Models. Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 1–13. https://doi.org/10.1145/3613904.3642428

[7] Ren, A. Z., Dixit, A., Bodrova, A., Singh, S., Tu, S., Brown, N., Xu, P., Takayama, L., Xia, F., Varley, J., Xu, Z., Sadigh, D., Zeng, A., & Majumdar, A. (2023). Robots That Ask For Help: Uncertainty Alignment for Large Language Model Planners (No. arXiv:2307.01928). arXiv. https://doi.org/10.48550/arXiv.2307.01928

[8] Duan, J., Pumacay, W., Kumar, N., Wang, Y. R., Tian, S., Yuan, W., Krishna, R., Fox, D., Mandlekar, A., & Guo, Y. (2024). AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation (No. arXiv:2410.00371). arXiv. http://arxiv.org/abs/2410.00371

[9] Knight, W. (n.d.). AI-Powered Robots Can Be Tricked Into Acts of Violence. Wired. Retrieved June 15, 2025, from https://www.wired.com/story/researchers-llm-ai-robot-violence/

[10] Robey, A., Ravichandran, Z., Kumar, V., Hassani, H., & Pappas, G. J. (2024). Jailbreaking LLM-Controlled Robots (No. arXiv:2410.13691). arXiv. https://doi.org/10.48550/arXiv.2410.13691

[11] Wu, X., Chakraborty, S., Xian, R., Liang, J., Guan, T., Liu, F., Sadler, B. M., Manocha, D., & Bedi, A. S. (2024). Highlighting the Safety Concerns of Deploying LLMs/VLMs in Robotics (No. arXiv:2402.10340). arXiv. https://doi.org/10.48550/arXiv.2402.10340

[12] Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., ... Kasneci, G. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 103, 102274. https://doi.org/10.1016/j.lindif.2023.102274

[13] Kim, C. Y., Lee, C. P., & Mutlu, B. (2024). Understanding Large-Language Model (LLM)-powered Human-Robot Interaction. Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction, 371–380. https://doi.org/10.1145/3610977.3634966

[14] Williams, T., Matuszek, C., Mead, R., & Depalma, N. (2024). Scarecrows in Oz: The Use of Large Language Models in HRI. J. Hum.-Robot Interact., 13(1), 1:1-1:11. https://doi.org/10.1145/3606261

[15] Zhang, B., & Soh, H. (2023). Large Language Models as Zero-Shot Human Models for Human-Robot Interaction. 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 7961–7968. https://doi.org/10.1109/IROS55552.2023.10341488

[16] Wang, Y., Xian, Z., Chen, F., Wang, T.-H., Wang, Y., Fragkiadaki, K., Erickson, Z., Held, D., & Gan, C. (2024). RoboGen: Towards Unleashing Infinite Data for Automated Robot Learning via Generative Simulation (No. arXiv:2311.01455; Version 3). arXiv. https://doi.org/10.48550/arXiv.2311.01455

[17] Latif, E. (2024). 3P-LLM: Probabilistic Path Planning using Large Language Model for Autonomous Robot Navigation (No. arXiv:2403.18778). arXiv. https://doi.org/10.48550/arXiv.2403.18778

[18] Dorbala, V. S., Mullen, J. F., & Manocha, D. (2024). Can an Embodied Agent Find Your "Cat-shaped Mug"? LLM-Based Zero-Shot Object Navigation. IEEE Robotics and Automation Letters, 9(5), 4083–4090. IEEE Robotics and Automation Letters. https://doi.org/10.1109/LRA.2023.3346800

[19] Gadre, S. Y., Wortsman, M., Ilharco, G., Schmidt, L., & Song, S. (2022). CoWs on Pasture: Baselines and Benchmarks for Language-Driven Zero-Shot Object Navigation (No. arXiv:2203.10421). arXiv. https://doi.org/10.48550/arXiv.2203.10421

[20] Zhang, J., Wang, K., Xu, R., Zhou, G., Hong, Y., Fang, X., Wu, Q., Zhang, Z., & Wang, H. (2024). NaVid: Video-based VLM Plans the Next Step for Vision-and-Language Navigation (No. arXiv:2402.15852; Version 7). arXiv. https://doi.org/10.48550/arXiv.2402.15852

[21] Biggie, H., Mopidevi, A. N., Woods, D., & Heckman, C. (2023). Tell Me Where to Go: A Composable Framework for Context-Aware Embodied Robot Navigation (No. arXiv:2306.09523). arXiv. https://doi.org/10.48550/arXiv.2306.09523

[22] Huang, C., Mees, O., Zeng, A., & Burgard, W. (2023). Visual Language Maps for Robot Navigation (No. arXiv:2210.05714; Version 4). arXiv. https://doi.org/10.48550/arXiv.2210.05714

[23] Macdonald, J. P., Mallick, R., Wollaber, A. B., Peña, J. D., McNeese, N., & Siu, H. C. (2024). Language, Camera, Autonomy! Prompt-engineered Robot Control for Rapidly Evolving Deployment. Companion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction, 717–721. https://doi.org/10.1145/3610978.3640671

[24] Nwankwo, L., & Rueckert, E. (2024). The Conversation is the Command: Interacting with Real-World Autonomous Robots Through Natural Language. Companion of

the 2024 ACM/IEEE International Conference on Human-Robot Interaction, 808–812. https://doi.org/10.1145/3610978.3640723

[25] Luo, S., Zhu, J., Sun, P., Deng, Y., Yu, C., Xiao, A., & Wang, X. (2024). GSON: A Group-based Social Navigation Framework with Large Multimodal Model (No. arXiv:2409.18084). arXiv. https://doi.org/10.48550/arXiv.2409.18084

[26] Zu, W., Song, W., Chen, R., Guo, Z., Sun, F., Tian, Z., Pan, W., & Wang, J. (2024). Language and Sketching: An LLM-driven Interactive Multimodal Multitask Robot Navigation Framework. 2024 IEEE International Conference on Robotics and Automation (ICRA), 1019–1025. https://doi.org/10.1109/ICRA57147.2024.10611462

[27] Wake, N., Kanehira, A., Sasabuchi, K., Takamatsu, J., & Ikeuchi, K. (2023). ChatGPT Empowered Long-Step Robot Control in Various Environments: A Case Application. IEEE Access, 11, 95060–95078. IEEE Access. https://doi.org/10.1109/ACCESS.2023.3310935

[28] Zhang, J., Tang, L., Song, Y., Meng, Q., Qian, H., Shao, J., Song, W., Zhu, S., & Gu, J. (2024). FLTRNN: Faithful Long-Horizon Task Planning for Robotics with Large Language Models. 2024 IEEE International Conference on Robotics and Automation (ICRA), 6680–6686. https://doi.org/10.1109/ICRA57147.2024.10611663

[29] Rana, K., Haviland, J., Garg, S., Abou-Chakra, J., Reid, I., & Suenderhauf, N. (2023). SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning (No. arXiv:2307.06135; Version 2). arXiv. https://doi.org/10.48550/arXiv.2307.06135

[30] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT (No. arXiv:2302.11382; Version 1). arXiv. https://doi.org/10.48550/arXiv.2302.11382

[31] Marvin, G., Hellen, N., Jjingo, D., & Nakatumba-Nabende, J. (2024). Prompt Engineering in Large Language Models. In I. J. Jacob, S. Piramuthu, & P. Falkowski-Gilski (Eds.), Data Intelligence and Cognitive Informatics (pp. 387–402). Springer Nature. https://doi.org/10.1007/978-981-99-7962-2_30