Vision-Language-Guided Motion Planning for Instruction-Based Robot Navigation

Priyanka Rose Varghese (prv2108), Tanya Mehta (tm3517)

***Abstract*—Autonomous robots navigating unfamiliar indoor environments must locate specific objects based on natural language instructions without prior map knowledge, requiring intelligent exploration and semantic reasoning under uncertainty. Existing approaches either rely on exhaustive room-by-room searching or fail to integrate detection confidence into their exploration strategies. We present a progressive confidence navigation system that combines zero-shot object detection with semantic scene understanding to make adaptive navigation decisions. Our approach uses Grounding DINO for open-vocabulary object detection across panoramic views, extracting 3D object locations through depth-based projection. A three-tier confidence framework determines whether to navigate directly to high-confidence detections (≥ 0.80), approach and re-assess medium-confidence matches (0.60-0.80), or trigger frontier-based exploration for low-confidence scenarios (< 0.60). When exploring, vision-language models verify room suitability before committing to detailed searches, reducing exploration time by approximately 40% for objects with strong room associations. The system integrates continuous SLAM mapping with dynamic A* replanning to handle newly discovered obstacles. Implemented in PyBullet with a three-room apartment environment, our system successfully navigates to semantically diverse objects across different rooms with a 99% success rate on easy scenes and 53% success rates on relatively harder scenes.**

I. Introduction

Classical planning algorithms, such as A*, Dijkstra, and RRT*, form the foundation of robotic navigation because they provide structured, reliable, and mathematically grounded methods for pathfinding. Operating on known map representations, these algorithms compute optimal or sub-optimal paths from a start position to a goal while avoiding obstacles. Their key strength is that, given a defined environment and cost function, the resulting path is reproducible and explainable.

Recent advances in Vision-Language Models (VLMs) have begun to bridge the gap between natural language understanding and semantic visual reasoning. Rather than treating vision and natural language as two separate modules, VLMs can connect words to visual concepts [3, 8], enabling robots to interpret instructions like "go to the fridge" by grounding language in visual context. Modern open-vocabulary object detectors like GroundingDINO extend this capability by detecting arbitrary objects specified through text prompts, without requiring class-specific training.

Robots operating in human environments should be able to understand and act upon instructions expressed in natural language. To achieve this, our project integrates VLMs for interpreting natural language instructions and performing semantic reasoning with classical planning algorithms for accurate and efficient navigation. By combining these two components—what we call a Progressive Confidence Navigation System—the system connects human intent with robotic execution in previously unexplored environments. Our final system includes the following components:

- **Language-to-goal grounding:** interprets natural language instructions and proposes candidate targets from visual observations.
- **Confidence-aware navigation:** uses model confidence to decide whether to navigate directly, move closer to verify, or explore for more evidence.
- **Mapping and planning:** builds an incremental map, plans collision-free paths, and replans as the environment becomes better understood.

II. Related Work

Recent advances in vision-language models have enabled robots to ground natural language instructions in visual perception.

1) Vision-Language Models for Navigation:: CLIP (Contrastive Language-Image Pre-Training)[7] demonstrated that joint training on image-text pairs enables zero-shot transfer to various visual tasks, including object recognition from natural language descriptions. Building on this foundation, GroundingDINO[4] introduced open-set object detection by combining a Transformer-based detector with language grounding, allowing detection of arbitrary objects specified through text prompts without class-specific training.

HuLE-Nav[2] proposed a human-like exploration strategy for zero-shot object navigation using VLM-driven semantic value maps and exploration memory. Their approach demonstrated that incorporating semantic priors from vision-language models improves navigation efficiency compared to purely geometric methods. Similarly, Shah et al.[10] integrated LLM-derived semantic priors into classical planning frameworks, showing that "semantic guesswork" as a heuristic significantly reduces exploration time compared to uninformed search methods.

2) Frontier-based exploration: Frontier-based exploration, introduced by Yamauchi [11], identifies the boundary between known free space and unknown regions as exploration targets. This seminal work established that frontier cells—where known space meets unexplored space—provide natural goals for autonomous exploration. The original algorithm used a simple distance-based frontier selection strategy. Information-gain-based exploration extends frontier methods by quantifying expected information from visiting each frontier. Holz et al. [1] proposed evaluating frontiers based on both distance and

potential information gain, balancing exploration efficiency with map coverage. Their approach demonstrated significant improvements in exploration time for structured indoor environments. Wavefront frontier detection using breadth-first search (BFS) provides computational efficiency for real-time exploration. Keidar and Kaminka [9] showed that BFS-based wavefront expansion efficiently identifies reachable frontiers while naturally incorporating connectivity constraints from the occupancy grid.

III. Methodology

Our Progressive Confidence Navigation System integrates vision-language models with classical planning algorithms to enable autonomous object-goal navigation in unexplored indoor environments.

The system architecture consists of seven primary modules: (1) Natural Language Query Processing, (2) Multimodal Perception, (3) Confidence Assessment, (4) 3D Goal Localization, (5) SLAM and Mapping, (6) Path Planning, and (7) Frontier-Based Exploration. Figure 1 illustrates the complete system pipeline.

The core innovation lies in our progressive confidence framework, which mimics human verification behavior by reassessing object identity at closer distances when initial confidence is uncertain. This approach addresses the fundamental challenge that vision models may confidently misclassify objects or correctly identify distant objects with moderate confidence.

A. Simulation Environment and Hardware Setup

1) PyBullet Physics Simulator: We implemented our system using the PyBullet physics engine, which provides realistic rigid body dynamics and collision detection. PyBullet was chosen over alternatives like HabitatSim [6] due to its flexible simulation parameters, direct low-level control interface, and compatibility with custom SLAM implementations without navigation mesh conflicts. The simulation environment consists of a multi-room apartment layout with realistic furniture and objects. The environment dimensions span approximately 20m × 20m with multiple rooms including a kitchen, bedroom, and living area.

a) Robot Platform: We simulate a Clearpath Husky A200 unmanned ground vehicle (specifications mentioned in Appendix A)

b) Apartment Layouts: We generate 3 layouts/scenes to test our system.(specifications mentioned in Appendix B)

B. Natural Language Query Processing

The system accepts natural language instructions in the form: "Go to [object]" or "Find the [object]". We use simple keyword extraction to identify the target object:

The extracted object name is then used as the text prompt for GroundingDINO and as part of the semantic reasoning query for GPT-4V.

C. Multimodal Perception

Our perception system combines two complementary vision-language models: GroundingDINO for fine-grained object detection and GPT-4V for high-level semantic reasoning.

1) Panoramic View Capture: To achieve comprehensive 360° coverage, the robot captures 8 RGB-D views at 45° intervals around its current position.

2) GroundingDINO Object Detection: GroundingDINO [4] is an open-vocabulary object detector that combines self-distillation (DINO) with language grounding, enabling detection of arbitrary objects specified through natural language prompts.

Architecture : The model consists of three main components:

1) Feature Enhancer: fuses image and text features using Transformer encoders.
2) Language-Guided Query Selection: generates object queries conditioned on the text prompt.
3) Cross-Modality Decoder: performs iterative cross-attention between visual and linguistic features to localize objects.

Detection Process : For each panoramic view, the input image and target object text (e.g., "fridge") are processed. Images are preprocessed and normalized, features are extracted from both modalities, and bidirectional cross-attention fuses them. The decoder iteratively refines object queries through six layers, producing bounding box predictions with associated confidence scores. Detections below threshold $\tau_{\text{box}} = 0.35$ are filtered.

Output: Each detection includes a bounding box (x_1, y_1, x_2, y_2) in pixel coordinates, confidence score $c \in [0, 1]$, semantic label ℓ, and the associated depth and RGB images. Aggregating detections across all 8 views produces the complete detection set $\mathcal{D}$.

3) GPT-4V Semantic Reasoning: While GroundingDINO provides precise spatial localization, GPT-4V [5] contributes high-level semantic understanding and contextual reasoning. For each detection, we query GPT-4V with the cropped RGB image and structured prompt requesting:

1) Room type classification
2) Likelihood assessment that the detected object matches the target category
3) Numerical confidence with reasoning

GPT-4V returns a structured response containing room type r, likelihood label $\lambda \in \{\text{yes, uncertain, no}\}$, base confidence $c_{\text{VLM}} \in [0, 1]$, and natural language reasoning.

4) Combined Confidence Scoring: We combine GroundingDINO and GPT-4V confidences through weighted averaging:

$$C_{\text{combined}} = \alpha \cdot C_{\text{GroundingDINO}} + \beta \cdot C_{\text{GPT-4V}}, \quad \alpha + \beta = 1$$

Empirically, $\alpha = 0.7$ and $\beta = 0.3$, prioritizing precise spatial localization while incorporating semantic verification. For the detection set $\mathcal{D}$, each detection d receives a combined score $C_{\text{combined}}(d)$. The best detection is:

$$d^* = \arg\max_{d \in \mathcal{D}} C_{\text{combined}}(d)$$

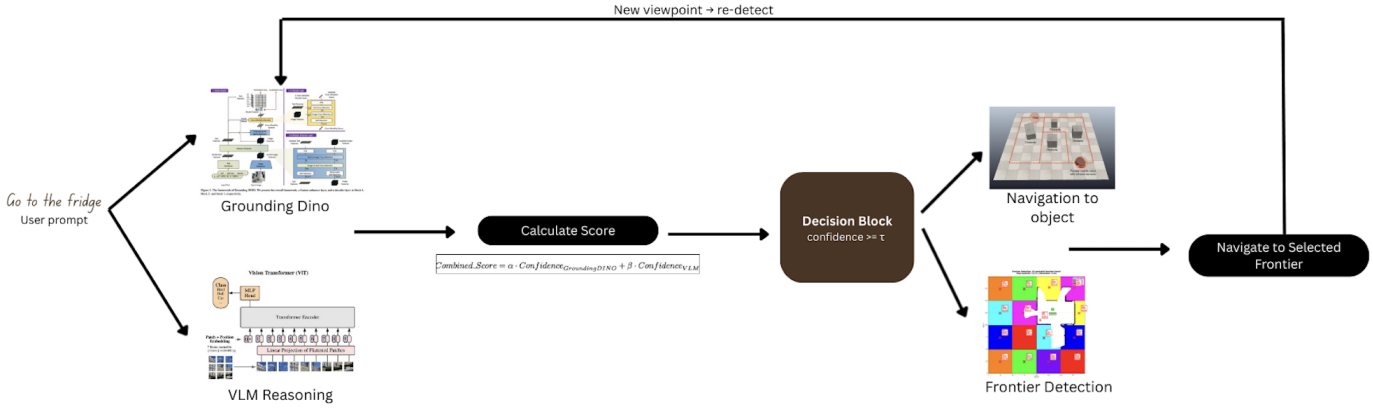


Fig. 1: A flowchart of our proposed pipeline

D. Progressive Confidence Assessment and Decision Making

We implement a two-stage verification framework that mimics human behavior: when uncertain about an object's identity, humans move closer to verify. Our system replicates this strategy through progressive confidence assessment.

1) Confidence Thresholds: Robot behavior is guided by three determined confidence levels:

- **High confidence** ($C_{\text{combined}} \geq 0.80$)**:** Object identity is reliable; navigate directly.
- **Medium confidence** ($0.60 \leq C_{\text{combined}} < 0.80$)**:** Object is plausible but requires closer verification.
- **Low confidence** ($C_{\text{combined}} < 0.60$)**:** Object is unlikely to be the target; explore alternative locations.

E. 3D Goal Localization from Detected Objects

After confirming a high-confidence detection, we compute a collision-free navigation goal through coordinate transformations and geometric reasoning.

1) Depth and Camera Coordinates: From the detected bounding box and depth image, we extract the median depth z_{object} (filtering invalid readings) and project the box center to 3D camera coordinates using the pinhole camera model with intrinsics (f_x, f_y, c_x, c_y).

2) World Transformation: We transform camera coordinates to the world frame using the robot's pose $\mathbf{p}_r = (x_r, y_r, \theta_r)$ and camera offset $\mathbf{t}_{\text{cam}}$, yielding the object's world position $\mathbf{p}_{\text{obj}}$.

3) Goal Selection: Since $\mathbf{p}_{\text{obj}}$ may be unreachable due to the robot's footprint radius $r = 0.5$ m, we sample 16 candidate goals uniformly around the object at distance $d = 0.8$ m (footprint plus 0.3 m safety margin). The collision-free candidate closest to the robot is selected:

$$\mathbf{g}^* = \arg\min_{\mathbf{g}_i \in \text{feasible}} \|\mathbf{g}_i - \mathbf{p}_r\|_2$$

If no feasible goal exists, frontier-based exploration is triggered.

F. SLAM and Occupancy Grid Mapping

1) Occupancy Grid Representation: We maintain a 2D occupancy grid map M that discretizes the environment into cells. Each cell $m_{i,j}$ stores the log-odds probability of being occupied:

$$m_{i,j} = \log \frac{P(\text{occupied})}{P(\text{free})}.$$

The grid has a resolution of 0.1 m/cell (10 cells per meter) over a 200×200 grid (20 m $\times$ 20 m). Cells are initialized to $m_{i,j} = 0$ (unknown, corresponding to 0.5 probability). Cells are classified as occupied if $m_{i,j} > 0.7$ and free if $m_{i,j} < -0.7$, with intermediate values representing uncertainty.

2) LiDAR Raycasting and Map Update: We update the occupancy grid using 2D LiDAR range measurements with 270° coverage ($-135°$ to $+135°$) and a maximum range of 6 m. For each valid measurement (0.1–6.0 m), the ray angle in the world frame is computed, and the obstacle endpoint is determined. All grid cells intersected by the ray from the robot to the obstacle are identified using Bresenham's line algorithm.

3) Occupancy Query and Collision Checking: To determine occupancy, a world coordinate (x, y) is converted to grid indices and its log-odds value $m_{i,j}$ is compared against the occupied threshold (0.7), with positions outside the map bounds conservatively treated as occupied. For robot footprint collision checking, all grid cells within a circular region of radius r must be free. Cells are first selected within a square bounding box and then filtered using the criterion $dx^2 + dy^2 \leq r_{\text{cells}}^2$, where r_{cells} is the radius expressed in grid units.

G. Classical Path Planning

Once a navigation goal and occupancy map are available, collision-free paths are computed using graph search algorithms.

1) A Search Algorithm:* A* finds the shortest path by exploring nodes in order of increasing estimated total cost:

$$f(n) = g(n) + h(n),$$

where $g(n)$ is the actual cost from the start to node n, and $h(n)$ is the heuristic estimate to the goal. We use Euclidean distance as an admissible heuristic:

$$h(n) = \sqrt{(x_n - x_{\text{goal}})^2 + (y_n - y_{\text{goal}})^2}.$$

The algorithm maintains a priority queue sorted by f-score, explores 8-connected neighbors, and checks collision-free paths using the robot's footprint radius. Diagonal movements are weighted by $2 \times \sqrt{2} \times$ resolution, while axis-aligned movements use a single resolution unit.

2) Dijkstra's Algorithm: Dijkstra's algorithm is equivalent to A* with $h(n) = 0$. It guarantees shortest paths but explores more nodes due to the absence of heuristic guidance.

3) Dynamic Replanning: As the occupancy map updates with new LiDAR measurements, blocked waypoints trigger replanning from the current position to the original goal. If replanning fails three times consecutively, navigation is aborted and exploration is initiated.

H. Frontier Detection

When no high-confidence detection is available, the robot switches from goal-driven navigation to exploration. We use **frontiers**—the boundary between known free space and unknown space—as exploration targets.

1) Frontier Detection via Wavefront BFS: We run a wavefront BFS flood-fill from the robot's current cell over *reachable free* cells only [9]. A free cell is marked as a frontier if it is adjacent to at least one unknown cell. This ensures the frontiers we detect are both informative (they touch unknown space) and feasible to reach (they lie in the connected free component).

2) Frontier Clustering with Connected Components: The detected frontier cells are grouped into connected components using 8-connectivity, so each physical boundary (e.g., a doorway edge) becomes a single frontier region. We filter very small clusters as noise and represent each remaining frontier by its centroid (world coordinates) and its size.

3) Frontier Selection Strategy: We rank candidate frontiers using a simple tradeoff between distance (prefer closer goals) and information gain (approx. by frontier size) [1]. In practice, we also use a distance-first variant that selects the nearest frontier above a minimum size threshold, which reduces failures from chasing far, hard-to-reach boundaries.

IV. Experiments & Results

A. Simulation Environment

We conducted all experiments in the PyBullet physics simulator using three indoor layouts (in Appendix B) of increasing complexity.

Layout 1 (Single Room) is a simple open environment with minimal obstacles, used to evaluate baseline perception and navigation performance.

Layout 2 (Multi-Room Easy) consists of multiple connected rooms with moderate structural complexity, enabling evaluation of room transitions and goal localization across spaces.

Layout 3 (Cluttered) is a densely furnished environment with narrow passages and significant occlusions, designed to stress-test perception, path planning, and collision avoidance.

TABLE I: Environment Layout Statistics

Layout	# Rooms	# Objects
Single Room	1	5
Multi-Room (Easy)	3	12
Cluttered	4	20

B. Multimodal Perception Evaluation

Our methodology adopts a dual-perception framework that integrates visual grounding and semantic reasoning. To isolate and quantify the contribution of each component, we evaluate three perception configurations:

- **GroundingDINO Only**: Open-vocabulary object detection using raw visual confidence scores.
- **Semantic only**: Semantic reasoning based on image crops and textual prompts, without explicit object localization (we used CLIP initially).
- **Hybrid (GroundingDINO + GPT-4V)**: Our full system combining spatial localization and semantic verification.

1) Evaluation Metrics: Perception performance is measured using the following metrics:

- **True Positive Rate (TPR)**: $\frac{\text{Correct identifications}}{\text{Total target objects}}$
- **False Positive Rate (FPR)**: $\frac{\text{Incorrect identifications}}{\text{Total detections}}$

TABLE II: Multimodal Perception Performance Comparison

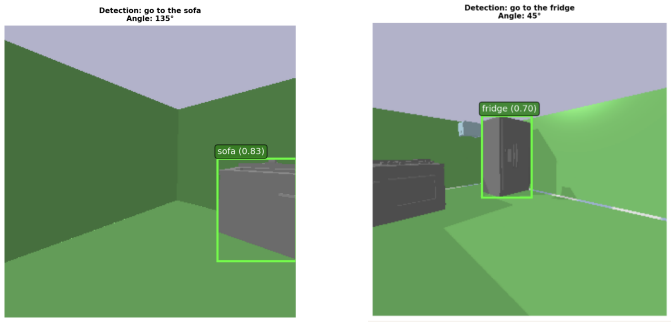
Method	TPR (%)	FPR (%)
GroundingDINO Only	88	18
Semantic Only	61	9
Hybrid (Final)	**93**	**6**

2) Analysis: GroundingDINO achieves high true positives but suffers from false positives in cluttered scenes, while GPT-4V reduces false positives using semantic context but misses some detections due to limited spatial grounding. The hybrid approach balances both, increasing true positives and suppressing false positives by combining precise localization with contextual reasoning, yielding the most reliable performance for navigation.

Fig. 2: Output of GPT-4V

C. Evaluation of Progressive Confidence

The progressive confidence mechanism improves object verification by increasing true positives and true negatives, particularly for objects that are distant or partially occluded. The



(a) False Positive (b) True Positive

Fig. 3: Output examples for GroundingDINO

TABLE III: Impact of Progressive Confidence

Method	Correct Detections (%)
W/o Progressive Confidence	40
With Progressive Confidence	$\geq$70

progressive confidence mechanism improves object verification by increasing true positives and true negatives, particularly for distant or partially occluded objects. Using empirically tuned hyperparameters mentioned in the Section D.1 yielded the results in Table III.

While this method significantly improves detection in simpler environments, performance degrades in highly cluttered or complex layouts. More sophisticated strategies or adaptive hyperparameter tuning may be required to handle these challenging scenarios effectively.

D. Frontier Detection

We evaluate three strategies for selecting frontiers during exploration:

1) **Information Gain (IG):** Select frontiers that maximize the expected reduction in map uncertainty, prioritizing areas likely to reveal the most unknown space.
2) **Distance-Based (D):** Select the closest frontier to the robot, minimizing travel cost but potentially ignoring high-value unexplored regions.
3) **Combined (IG + D):** Score frontiers based on a weighted combination of information gain and distance:

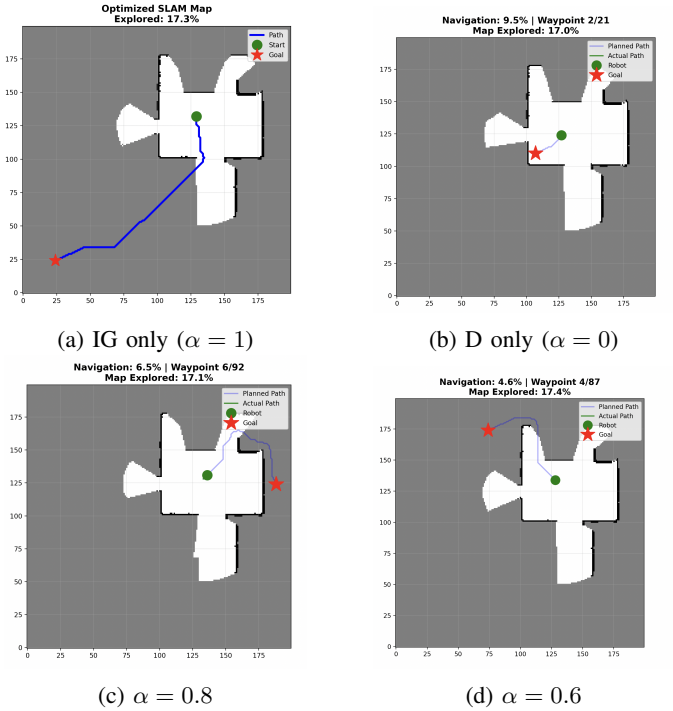
$$\text{Score} = \alpha \cdot \text{IG} - \beta \cdot \text{D},$$

where α =0.6 and β=0.4 (after tuning).

Fig 5 shows the difference in the outputs with the different α and β choices.

1) Analysis: : Using only Information Gain (IG) drives the robot toward frontiers with maximal expected information, often leading to distant goals that may be unreachable in practice. Using only Distance (D) prioritizes the nearest frontiers, enabling quick exploration but potentially ignoring high-value regions with more informative content.

Combining IG and D with a weighted approach (e.g., $\alpha = 0.6$ for IG, $\alpha = 0.4$ for D) strikes a balance: the robot slightly prioritizes information gain while still considering distance, resulting in efficient exploration that reaches informative frontiers without unnecessary travel.



(a) IG only ($\alpha = 1$) (b) D only ($\alpha = 0$)

(c) $\alpha = 0.8$ (d) $\alpha = 0.6$

Fig. 4: Comparison of frontier detection outputs for different strategies.

The frontier selection strategy is currently blind to semantic information and chooses the next frontier purely based on the information-distance tradeoff equation. Incorporating a dynamic semantic check before committing to a frontier could reduce unnecessary travel and save time by avoiding frontiers unlikely to contain relevant objects.

E. Evaluation of Entire Pipeline

1. Task Completion Metrics: We evaluate the overall performance of the robot using task completion metrics that capture both effectiveness and efficiency.

- **Success Rate (%):** Fraction of trials in which the robot successfully reaches the target object.
- **Failure Rate (%):** Fraction of trials where navigation fails due to mislocalization, collisions, unreachable paths, or incorrect object identification.
- **Number of Replanning Events:** Total number of times the robot had to recompute its path due to dynamic obstacles or blocked paths.

TABLE IV: Task Completion Metrics for A*

Metric	Layout 1	Layout 2	Layout 3
Success Rate (%)	99	85	58
Failure Rate (%)	1	15	42
Number of Replanning Events (avg)	2	5	7

TABLE V: Task Completion Metrics for Dijkstra

Metric	Layout 1	Layout 2	Layout 3
Success Rate (%)	100	90	65
Failure Rate (%)	0	10	35
Number of Replanning Events(avg)	1	3	6

1) Analysis: Navigation in Layout 3 is slower and less reliable due to increased obstacles, triggering frequent replanning. In some cases, the robot exceeds the maximum allowed replanning attempts, resulting in navigation failures. Layout 1 rarely fails, with issues only arising when the target is initially out of view and exploration is required. Layout 2 exhibits intermediate behavior, with occasional replanning but generally successful navigation.

2. Navigation Efficiency

We evaluate navigation efficiency using the following metrics on a single task *"go to the fridge"* from the living room (Layout 2 and 3):

- **Path Length (m):** Total distance traveled to reach the target. Shorter paths indicate more efficient planning.
- **Total Time (s):** Time taken from start to planning to goal. Lower times suggest faster navigation and lesses replanning attempts.

TABLE VI: Navigation Efficiency: Single Room Layout

Algorithm	Path Length(m)	Total Time(s)
A*	3.2	75.3
Dijkstra	3.1	84.8

TABLE VII: Navigation Efficiency: Multi-Room Easy Layout

Algorithm	Path Length(m)	Total Time(s)
A*	10.1	223.7
Dijkstra	10.0	224.5

TABLE VIII: Navigation Efficiency: Cluttered Layout

Algorithm	Path Length(m)	Total Time(s)
A*	26.4	1231.4
Dijkstra	26.6	1332.1

2) Analysis: The navigation efficiency results reflect the influence of environmental complexity on performance. Multi-Room and Cluttered layouts exhibit increased total navigation time due to a higher frequency of obstacles, narrow passages, and occlusions, which trigger multiple replanning events. Consequently, the robot spends more time recomputing paths, leading to longer navigation durations. A* generally outperforms Dijkstra in planning time due to heuristic guidance, especially in layouts with more rooms and obstacles.

Currently, replanning is capped at three attempts, which may occasionally lead to missed goals. An effective improvement would be to retain semantic information after initial exploration, reducing the need for repeated full perception passes. This approach would save planning time and computational resources while maintaining navigation reliability.

V. Conclusion

We developed a robot system capable of understanding natural language queries (e.g., "go to the fridge") and autonomously navigating to target objects in unexplored environments. Key contributions include: a two-stage Progressive Confidence Navigation strategy that reduces false positives while maintaining high success rates; Multimodal Perception Fusion combining GroundingDINO and GPT-4V for accurate and context-aware object detection; intelligent exploration balancing information gain and distance to efficiently discover new spaces; and a robust implementation validated across layouts of increasing complexity, where A* outperformed Dijkstra in path length and planning time, and both achieved 100% success in single-room scenarios.

Future Enhancements: These are the promising directions for future enhancement: Incorporating semantic priors into frontier selection, e.g., refrigerators are more likely in kitchens, to reduce exploration time in large, multi-room environments, extending VLM integration to maintain persistent room-type and object distributions, enabling smarter search strategies based on context.

Perform evaluation on diverse and dynamic environments (e.g., Habitat HSSD-200, Matterport3D) and explore advanced planners. Work on compounded instructions (e.g., "find the cup on the kitchen table") and sequential tasks (e.g., "go to fridge, then microwave").

VI. Team Member Contributions

Priyanka Rose Varghese: Worked on development of the frontier-based exploration module, where frontiers were prioritized using distance, information gain, and their weighted combination, with hyperparameters tuned for all layouts. Implemented and integrated the multimodal perception module (GroundingDINO object detection with GPT-4V semantic verification), designing experiments to evaluate the contributions of visual and semantic reasoning individually and in combination. Tuned hyperparameters for progressive confidence mechanism. Contributed equally on system architecture design and final report.

Tanya Mehta: Setup simulation + navigation infrastructure across HabitatSim and PyBullet. Implemented the SLAM-style occupancy mapping pipeline from depth/range observations and designed the map-to-planning-grid conversion used by the planners. Built and evaluated the classical planning module (A*, Dijkstra), along with collision checking and visualization of paths and exploration traces. Implemented 3D goal localization and developed the local execution layer (waypoint following, stuck handling, and dynamic replanning when obstacles invalidate the current plan). Ran and summarized the HabitatSim prototype experiments. Contributed equally on system architecture design and final report.

References

[1] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, P. Abbeel, Nathan Michael, and Vijay R. Kumar. Information-theoretic planning with

trajectory optimization for dense 3d mapping. *Robotics: Science and Systems XI*, 2015. URL https://api.semanticscholar.org/CorpusID:3014987.

[2] Peilong Han, Min Zhang, Jianye HAO, Hongyao Tang, and YAN ZHENG. HuLE-nav: Human-like exploration for zero-shot object navigation via vision-language models. In *NeurIPS 2024 Workshop on Behavioral Machine Learning*, 2024. URL https://openreview.net/forum?id=akVkINWMxg.

[3] Zhiyuan Li, Yanfeng Lu, Yao Mu, and Hong Qiao. Cogga: A large language models-based generative agent for vision-language navigation in continuous environments, 2024. URL https://arxiv.org/abs/2409.02522.

[4] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2024. URL https://arxiv.org/abs/2303.05499.

[5] OpenAI. GPT-4V(ision) System Card. https://cdn.openai.com/papers/GPTV_System_Card.pdf, September 2023. URL https://cdn.openai.com/papers/GPTV_System_Card.pdf. System Card.

[6] Xavi Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Ruslan Partsey, Jimmy Yang, Ruta Desai, Alexander William Clegg, Michal Hlavac, Tiffany Min, Theo Gervet, Vladimir Vondrus, Vincent-Pierre Berges, John Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A co-habitat for humans, avatars and robots, 2023.

[7] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

[8] Abhinav Rajvanshi, Karan Sikka, Xiao Lin, Bhoram Lee, Han-Pang Chiu, and Alvaro Velasquez. Saynav: Grounding large language models for dynamic planning to navigation in new environments, 2024. URL https://arxiv.org/abs/2309.04077.

[9] P.G.C.N. Senarathne, Danwei Wang, Zhuping Wang, and Qijun Chen. Efficient frontier detection and management for robot exploration. In *2013 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, pages 114–119, 2013. doi: 10.1109/CYBER.2013.6705430.

[10] Dhruv Shah, Michael Equi, Blazej Osinski, Fei Xia, Brian Ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning, 2023. URL https://arxiv.org/abs/2310.10103.

[11] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997. doi: 10.1109/CIRA.1997.613851.

APPENDIX

A. Robot Platform and Sensor Configuration

1) Robot Platform: Clearpath Husky A200: We simulate a Clearpath Husky unmanned ground vehicle with the following specifications:

- **Dimensions:** $0.99\ \mathrm{m}$ (L) × $0.67\ \mathrm{m}$ (W) × $0.39\ \mathrm{m}$ (H)
- **Footprint radius:** $0.5\ \mathrm{m}$ (for collision checking)
- **Maximum velocity:** $1.0\ \mathrm{m/s}$ (linear), $2.0\ \mathrm{rad/s}$ (angular)
- **Kinematics:** Differential drive, four-wheeled skid-steering configuration

Fig. 5: Pybullet's Husky

2) Sensor Configuration: The robot is equipped with two primary sensors:

RGB-D Camera:

- **Resolution:** 640×480 pixels
- **Field of View:** $90°$ horizontal
- **Depth range:** $0.1\ \mathrm{m}$ to $10\ \mathrm{m}$
- **Mounting:** Forward-facing at $0.39\ \mathrm{m}$ height
- **Frame rate:** 30 Hz
- **Camera intrinsics:**

$$f_x = 320.0, \quad f_y = 320.0, \quad c_x = 320.0, \quad c_y = 240.0$$

2D LiDAR:

- **Range:** 6 meters
- **Angular coverage:** $270°$ ($-135°$ to $+135°$)
- **Angular resolution:** $1°$ (271 rays)
- **Mounting:** At robot center, $0.2\ \mathrm{m}$ height
- **Scan rate:** 10 Hz

B. Environment Layouts

We have 3 main layouts genertaed for our evaluation and all three are created using 3D furniture models from Kenny's freely available asset library[1]. Kenny provides high-quality, low-poly 3D models under the CC0 1.0 Universal license, including furniture, household objects, and architectural elements. We selected models that closely represent real-world household items to ensure realistic perception challenges for our vision-language detection pipeline.

[1]https://kenney.nl/assets

Layout 1: Single-Room Kitchen : **Dimensions:** 8m × 8m single open room
Structural Complexity: One rectangular space with no internal walls or doorways
Object Count: 5 target objects
Objects: Refrigerator, stove, sink, toaster, Island
Purpose: Baseline evaluation for perception and navigation without multi-room exploration challenges.

This layout isolates object detection performance from exploration complexity. All target objects are potentially visible from multiple locations within the room, allowing assessment of GroundingDINO and GPT-4V performance under optimal geometric conditions.

Layout 2: Three-Room Apartment: **Dimensions:** 12m × 16m multi-room layout
Structural Complexity: Moderate—three interconnected rooms with doorways
Object Count: 9 objects across 3 rooms
Room Configuration:

- **Living Room** (12m × 8m, center): Coffee table
- **Bedroom** (8m × 8m, top-left): Bed, nightstand (×2), coat rack
- **Kitchen** (8m × 8m, top-right): Refrigerator, stove, sink, toaster

Purpose: Evaluate room-to-room navigation, doorway traversal, and frontier-based exploration when target objects are not immediately visible.

This layout introduces spatial reasoning challenges: the robot must explore multiple rooms through doorways connecting to a central living room hub. Object distribution across rooms tests the system's ability to handle negative search results (object not in current room) and trigger exploration behavior.

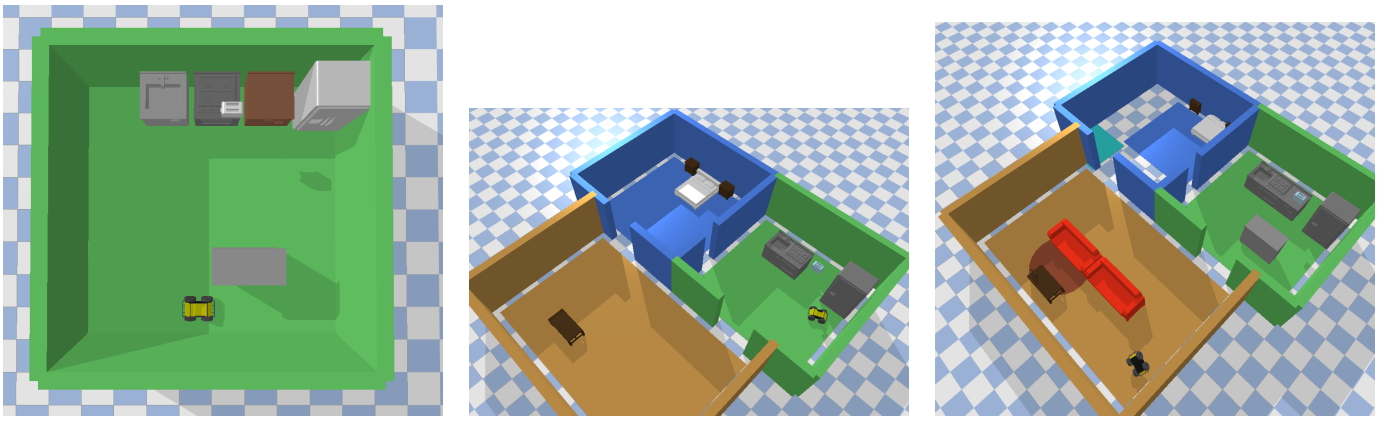
Layout 3: 3-Room Apartment: **Dimensions:** 16m × 12m multi-room layout
Structural Complexity: High—four rooms with more objects, narrow passages and occlusions
Object Count: 14 objects across 3 rooms
Room Configuration:

- **Living Room** (12m × 6m): Coffee table, Couch(x2)
- **Bedroom** (6m × 6m): Bed, nightstand (×2), coat rack
- **Bathroom** (4m × 6m): Toilet, bathtub, sink, mirror
- **Kitchen** (6m × 6m): Refrigerator, stove, sink, toaster, Island

Purpose: Maximum complexity, longer exploration paths due to more obstacles, semantically similar objects, and increased occlusion.

These layouts represents realistic apartment complexity with multiple challenges:

1) **Longer Exploration Chains**: Target objects in the bedroom require navigating through multiple doorways from typical starting positions.
2) **Narrow Passages**: 1.5m doorways stress-test collision avoidance with the Husky's 0.5m footprint radius.



(a) Single Room(Kitchen (b) Multi-Room Easy (c) Cluttered

Fig. 6: Visualization of the three simulation environment layouts

3) **Partial Observability**: Furniture occlusions are more frequent, requiring the "approach and re-identify" confidence threshold strategy.

C. HabitatSim Prototype (Pre-PyBullet)

Before moving our full stack to PyBullet, we prototyped the mapping + planning pipeline in HabitatSim [6] to leverage its photorealistic indoor scenes and built-in sensors. In this prototype, we were able to (i) generate a local SLAM-style occupancy map from range/depth observations, (ii) compute collision-free global paths using classical planners (A*, Dijkstra, and RRT), and (iii) visualize planned trajectories and exploration behavior.

In practice, execution was the main bottleneck. HabitatSim's navigation mesh and our occupancy-grid planner could disagree, producing paths that looked valid on the grid but were difficult to execute reliably. We also observed sensitivity to mapping parameters (e.g., obstacle inflation and depth/range thresholds) and brittle local control: the waypoint follower could oscillate near walls or get stuck in narrow passages. These issues motivated our switch to PyBullet for more direct low-level control and to avoid navigation-mesh conflicts in our custom SLAM loop.

In HabitatSim, we reached the point where our pipeline could build a local occupancy map from sensor observations and run classical planners on that learned map, which let us evaluate mapping quality separately from execution. Figure 7 shows two example planning grids produced by automatic SLAM runs, Table IX reports map coverage and the size/connectivity of the reachable free-space component across runs, Table X summarizes how often A* plans could be executed to completion (including waypoint counts and replans), and Figure 8 compares the planned trajectories produced by Dijkstra, A*, and RRT on the same learned occupancy grids.

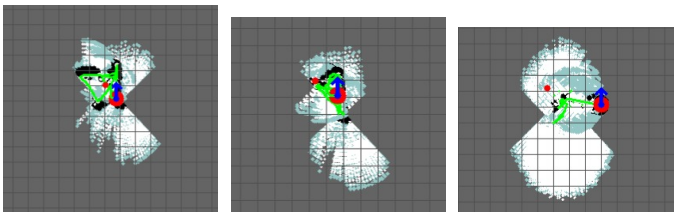
Fig. 7: Planning grids from two automatic SLAM runs.

TABLE IX: Coverage and connectivity over multiple automatic SLAM runs.

Free cells	Reachable cells	Max dist (cells)	Explored (%)
2662	233	22.8	0.3
16842	1	0.0	1.7
14518	1174	36.2	1.5
18713	1	0.0	1.9
12737	16010	164.9	1.3
3043	9	3.0	0.3
20531	28129	172.2	2.1
15700	1	0.0	1.6
13293	14	3.0	1.3
9282	13229	109.0	0.9
12914	26794	168.9	1.3
12636	15	3.0	1.3

A* path (cells)	Waypoints	Replans	Reached goal?
6	2	0	Yes
4	2	0	Yes
4	2	0	Yes
150–170	20–30	3 (max)	No (stuck early)
240	29	3 (max)	No (stuck early)
184	35	3 (max)	No (stuck at wp 4)
221	28	3 (max)	No (stuck at wp 2)

TABLE X: A* navigation performance across automatic SLAM runs



(a) Dijkstra path (b) A* path (c) RRT path

Fig. 8: Comparison of Dijkstra, A*, and RRT paths on the learned occupancy grid.