2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

NAIS: A Modular ROS 2 Framework for Real-Time Scene Graph Construction and Language-Guided Navigation

Jose M. Flores Gonzalez[a] , Enrique Coronado[a] and Natsuki Yamanobe[a]

***Abstract*— Indoor robot navigation in unfamiliar environments requires accurate mapping, contextual understanding, and semantic reasoning to interpret user intent. While ROS 2 navigation frameworks provide strong 3D SLAM capabilities, they lack an integrated, modular approach that unifies perception, semantic representation, and high-level reasoning in a single, real-time system. We present Navigation AI Impulse by Scene Graph (NAIS), the first ROS 2 framework to continuously fuse SLAM, dynamic open-vocabulary scene graph construction, and foundation-model reasoning into a deployable and modular pipeline for mobile robots. NAIS operates online, using VLMs to describe objects and LLMs to infer spatial relations and resolve ambiguous natural language requests. This unified design links environment understanding directly to navigation planning, enabling context-aware goal selection without manual configuration or static class lists. Demonstrations on a TurtleBot 4 in office environments show NAIS handling high-level, context-dependent commands such as "I'm thirsty" or "I'm tired", illustrating its potential for robust, adaptable indoor navigation.**

***Index Terms*— Robot Navigation, Large Language Models, Scene Graphs, Vision-Language Models, ROS 2, Semantic Mapping, System Integration**

I. Introduction

Autonomous navigation remains a key challenge in mobile robotics, where robots must map their environment and plan collision-free routes [1], [2]. Frameworks such as ROS 2 Nav2 excel at geometric mapping and path planning, but fall short when robots must interpret ambiguous, high-level user instructions and translate them into navigation goals [3]. For instance, a person might say "I'm thirsty," expecting the robot to locate a suitable object like a cup or water dispenser tasks that require contextual reasoning beyond geometry.

Scene graphs combined with large language models (LLMs) provide a promising solution. LLMs enable the interpretation of natural language commands [4], while scene graphs represent environments as nodes and edges capturing objects, places, and their relations [5]. Recent vision-language models (VLMs) enhance scene graphs with open-vocabulary descriptions connecting perception to reasoning. However, integrating SLAM, detection, VLMs, and LLMs in one real-time, extensible system remains challenging.

To bridge this gap, we introduce NAIS (Navigation AI Impulse by Scene Graph), a modular ROS 2 pipeline that unifies semantic mapping, foundation models, and navigation. NAIS operates in three stages: (1) constructing semantic scene graphs during exploration, (2) parsing user commands with LLMs for intent understanding, and (3) generating navigation plans that combine high-level reasoning with classical planning. We validate NAIS on a TurtleBot 4 in office environments, demonstrating its ability to execute context-dependent navigation tasks. Figure 1 shows an example of the scene graph representation, as well as the 3D test environment where the system was evaluated. The environment consists of two rooms with distinct contextual settings: a kitchen and an office. Within these rooms, various objects were detected and visualized as colored markers (cubes) in the 3D map. Specifically, green markers represent chairs, blue markers correspond to umbrellas, purple to bottles, pink to cups, and lime green to sofas. These visual cues illustrate how the proposed system integrates semantic information into the geometric map, allowing readers to easily correlate detected objects withnone deliverand contextual meaning across different environments.

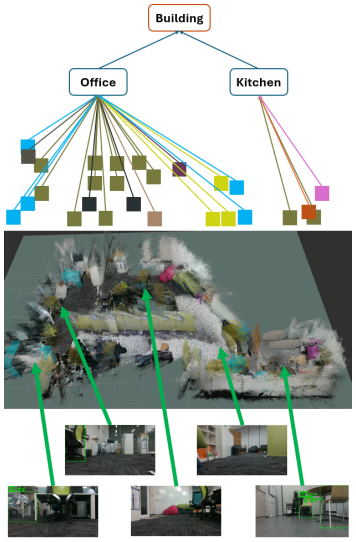

Fig. 1: Scene graph data structure showing objects within rooms and rooms within the building, along with the 3D map and real-world scene of the experiment

II. Related work and contributions

The scene graph literature offers a rich variety of methods, yet all fall short of delivering the full set of capabilities required for truly intelligent, adaptive robots. Early pipelines, such as those by Rosinol et al. [3], Chang et al. [6] and Gu et al. [7], separate mapping and graph construction into two offline stages. This design can work in static environments

[a] AIST Tokyo Waterfront, 2 Chome-3-26 Aomi, Koto City, Tokyo 135-0064, Japón.

Authors are members of the [a] AIST Tokyo Waterfront, 2 Chome-3-26 Aomi, Koto City, Tokyo 135-0064, Japón.

but inevitably incurs latency and prevents the robot from reacting to on the fly changes. NAIS overcomes this by fusing SLAM and semantic graph generation in a single, real-time ROS 2 pipeline, ensuring that both the metric map and the Scene Graph remain up to date as the robot explores.

ConceptGraphs [7] and HOV-SG [8] push semantic richness further ConceptGraphs with open vocabulary models in post processing, and HOV-SG with a hierarchical, multi stage batch pipeline that compacts map size. However, both require static setups and manual tuning of hyperparameters, making them unsuitable for continuous operation. Hydra [6], in contrast, achieves real-time graph updates onboard, yet confines nodes to basic geometric metadata without open vocabulary or complex relationships. Graph2Nav [9] similarly updates its graph online for navigation but limits semantics to a fixed, pre-trained set of classes. The 3D Dynamic Scene Graphs of Rosinol et al. [3] bring agents and objects together in a unified dynamic graph but only in photo realistic simulation, never on real hardware.

NAIS combines the best of these projects. It delivers continuous, low latency mapping and graph updates in real environments; it embeds powerful VLMs (LLaVA) and GPT 4 to achieve open vocabulary, context aware descriptions; and it links semantic understanding directly to navigation via a native GPT 4 based NLP component. All modules from sensor acquisition and YOLO based detection through semantic enrichment and graph construction to Nav2 execution run as independent ROS 2 nodes, making NAIS easy to extend or replace. This unified, real-time, semantically rich and fully modular design sets NAIS apart from every prior approach. The main differences between NAIS and related work are summarized in Table I.

Our main contributions are as follows.

- First ROS 2 framework to combine real-time SLAM and dynamic scene graphs with open-vocabulary VLM semantic enrichment and LLM-based intent parsing for ambiguous natural-language navigation on a real robot, delivered as a plug-and-play modular pipeline for seamless integration into existing ROS 2 navigation stacks.
- Demonstrated real-world operation on a TurtleBot4 in office environments, showing robust performance in handling high-level, ambiguous navigation commands.

III. Methods

A. Hardware Architecture

The hardware architecture for NAIS experimentation consists of two interconnected platforms running ROS 2 Humble with CycloneDDS. The mobile platform is a TurtleBot 4, responsible for locomotion and publishing odometry (`/odom`) using CycloneDDS's default QoS settings. The processing station handles most intensive computation on a laptop equipped with a 14th-generation Intel Core i9 CPU and an NVIDIA RTX 4070 GPU (8 GB VRAM) dedicated to vision processing. An Intel RealSense D435i camera is attached to this laptop via USB 3.0, providing synchronized RGB

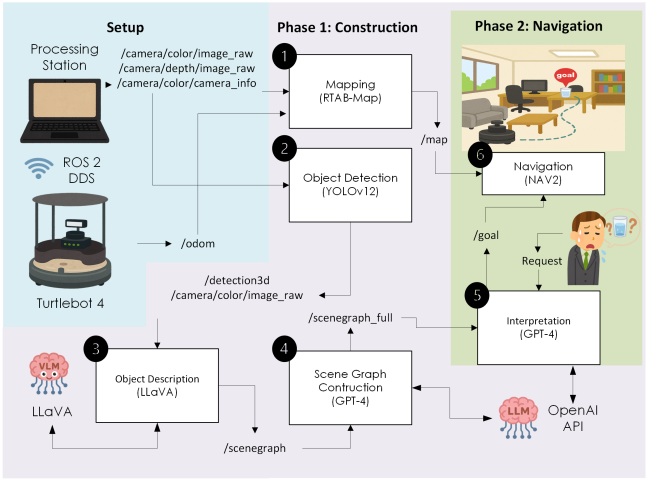

Fig. 2: High-level block diagram of the NAIS pipeline. The system operates in two phases: construction, where a semantic map is built using RTAB-Map, YOLOv8x, LLaVA, and GPT-4; and navigation, where GPT-4 interprets user requests and NAV2 executes goals on the TurtleBot 4 via ROS 2 topics.

images (`/camera/color/image_raw`), depth maps (`/camera/depth/image_raw`), and camera calibration parameters (`/camera/color/camera_info`). In addition, a desktop workstation (Intel Core i9 CPU, NVIDIA RTX 4090 GPU) runs the LLaVA model as an external service, which communicates with the laptop through an API interface. Wireless communication between the TurtleBot 4 and the processing station occurs over a WiFi 6 router operating on the 2.4 GHz band.

B. Software Architecture

In Figure 2 we illustrate the information flow that comprises the entire NAIS project, as well as its division into two processing phases. Phase 1, *Scene Graph Construction*, gathers all environmental data, processes it, and generates the scene graph data structure. Phase 2, *Navigation*, interprets user requests, determines how to satisfy them, and directs the robot to navigate autonomously.

Phase 1 consists of the nodes numbered in execution order in Figure 2: (1) Mapping, (2) Object Detection, (3) Object Description, and (4) Scene Graph Construction. Phase 2 comprises nodes (5) Interpretation and (6) Navigation. In this pipeline, the Mapping and Object Detection nodes run in parallel, while the remaining nodes execute sequentially. The following sections provide a detailed description of each node.

1) Node 1. Mapping: The Mapping node leverages RTAB-Map in its default configuration to construct all necessary environmental representations. It subscribes to raw RGB frames on the `/camera/color/image_raw` topic and depth frames on `/camera/depth/image_raw`, both of type `sensor_msgs/Image`, as well as camera calibration data on `/camera/color/camera_info`

TABLE I: Comparative analysis of recent Scene Graph–based frameworks in robotics, in the order mentioned in Section II

Aspect	3D Dynamic [3]	Hydra [6]	ConceptGraphs [7]	HOV-SG [8]	Graph2Nav [9]	NAIS
Year	2020	2022	2023	2024	2025	2025
Platform / Architecture	Python	ROS	Python	Python	ROS 2	ROS 2
Real-time Scene Graph Generation	✓	✓	✗	✗	✓	✓
Simultaneous Mapping + Scene Graph	✗	✓	✓	✓	✓	✓
Semantic Richness	✗	✗	VLMs	✓	✗	VLMs
Type of Relations	Spatial	Spatial	Spatial & Semantic	Spatial & Semantic	Spatial	Spatial & Semantic
Natural language interpretation	No	No	Adv. (GPT 4)	Adv. (Open Vocab)	Adv. (SayNav)	Adv. (GPT 4)
Application Domains	Semantic perception of dynamic elements	Real-time perception	Task planning	NL navigation indoor	Navigation	Natural language navigation indoor
Experimental Evaluation	Simulated	Real indoor	Real	Real & Simulated	Real in/outdoor	Real indoor
Modularity & Extensibility	Medium	Low	Medium	Medium	Medium	High

Notes: 3D Dynamic Scene Graphs and ConceptGraphs operate in offline mode, limiting adaptability to dynamic environments. HOV-SG uses a hierarchical, multi-stage batch pipeline that compacts map size but is not real-time. Hydra and Graph2Nav allow real-time updates but with limited semantic richness. NAIS uniquely combines real-time SLAM, open-vocabulary semantic enrichment, and LLM-based intent parsing on a real robot.

(`sensor_msgs/CameraInfo`) and odometry on `/odom` (`nav_msgs/Odometry`). From these inputs, it publishes a three dimensional point cloud at 10 Hz on `/rtabmap/cloud_map` (`sensor_msgs/PointCloud2`) for visualization in RViz. Additionally, RTAB-Map generates a geometric occupancy grid used by Nav2 for global planning and publishes the estimated robot pose on `/rtabmap/odom` (`nav_msgs/Odometry`), updating the system's TF tree accordingly.

To ensure consistency across reference frames, a custom URDF defines the static transform between `base_link` and the `camera_color_optical_frame` and `camera_depth_optical_frame` required by RTAB-Map. This setup guarantees that incoming messages are correctly labeled in TF (`odom` to `map`, `map` to `camera_link`, etc.). All RTAB-Map parameters including frame identifiers and internal memory settings remain at their defaults, allowing any compatible 3D SLAM system to replace this node without modifying the rest of the pipeline.

2) Node 2. Object Detection: The Object Detection node subscribes synchronously to the RGB image topic `/camera/color/image_raw` (`sensor_msgs/Image`) and the aligned depth image topic `/camera/aligned_depth_to_color/image_raw` (`sensor_msgs/Image`), as well as to camera calibration data on `/camera/color/camera_info` (`sensor_msgs/CameraInfo`) and to room context updates on `/scene_graph_json` (`std_msgs/String`). Upon receiving each image pair, the node applies YOLOv8x to produce two-dimensional bounding boxes with associated confidence scores [10]. Any detection with confidence below 0.7 or whose class belongs to a predefined set of dynamic objects (for example, people, vehicles, or animals) is discarded immediately. This ensures that only static elements of the environment are considered for the scene graph.

For each remaining bounding box, the node computes a robust depth estimate by taking the median value over a small neighborhood around the box center. That three-dimensional point is then transformed into the global `map` frame via TF2, and its Euclidean distance from the camera is calculated. To eliminate unreliable far-range detections, any object whose distance exceeds 4.5 meters is rejected, reflecting the effective depth range of the RealSense sensor.

To further improve detection stability, the node implements three additional filters. First, a temporal voting filter requires each candidate to appear in at least ten frames before it is accepted [11]. This heuristic prevents transient false positives, although it may occasionally overlook objects that the robot traverses too quickly. Second, a spatial de-duplication filter maintains a short history buffer of confirmed object positions (up to 100 entries) and rejects any new detection within a 0.5 m radius of an existing entry. This avoids duplicate entries caused by minor fluctuations in depth estimation or rapid robot motion. Third, when the room context changes—as indicated by an update on `/scene_graph_json`—the node allows "legitimate" duplicates: if an object previously confirmed in another room is now detected closer, the old record is removed and replaced by the new one, ensuring that the scene graph reflects the current environment.

After filtering, all validated objects are assembled into a `Detection3DArray`, where each `Detection3D` contains a three-dimensional bounding box and a full pose covariance. For convenience in downstream processing, a parallel `Detection2DArray` is also published, with each `Detection2D` storing the two-dimensional box center and size. Both arrays are published on `/objects/array` and `/objects/array_2d` at each frame. An annotated RGB image, displaying retained detections with green boxes and labels, is published on `/camera/detections/image_raw` for visualization. Throughout its operation, the node logs high-level statistics such as the current room and the number of published objects to facilitate debugging and performance monitoring.

3) Node 3. Object Description: Object description proceeds in three stages. In the filtering stage, any detection beyond 5 m is discarded to avoid excessive depth error, and a spatial buffer of ± 30 cm in x, y, z merges successive detections with the same label into a single instance; detections of mobile entities such as people or animals are also ignored to focus exclusively on static, inanimate objects. Next, semantic enrichment is performed by sending full frame images and 20% expanded object crops to a locally hosted LLaVA 1.5–7b model. Two specialized prompts one for classifying the room type and another for generating detailed object descriptions produce JSON formatted annotations that are parsed and attached to each node. The specialized prompt used for generating detailed object descriptions is responsible for extracting attributes such as color, relative size, and shape of the objects. Finally, the construction stage bundles filtered detections and enriched descriptions into a single JSON payload, which is sent to GPT-4 via the OpenAI API; the returned JSON encodes a complete SceneGraph of rooms, objects, and their spatial and semantic relationships. During runtime, the node publishes the current SceneGraph as a `std_msgs/String` on `/scene_graph` every 30 s and writes a final `.json` file upon shutdown.

4) Node 4. Scene Graph Construction: This node subscribes to the `/scene_graph_json` topic, which publishes a JSON-encoded string listing all objects detected in their respective rooms. Whenever the node detects that this list has been updated, it sends the data—together with a carefully crafted prompt—to GPT-4. The prompt (shown in Listing 1) instructs the model to organize the information into a hierarchical scene graph in strict JSON format. It begins by placing all entities under a root "Building" node (ID 0) and then establishes "contains" relationships based on object type and context—for example, rooms belong to the building and small items (such as a laptop) belong to larger surfaces (such as a desk). The prompt was refined through a heuristic process to ensure consistent, accurate graph structure. Finally, the node overwrites the file `Scene_Graph.json` with GPT-4's output each time a new update arrives, keeping the scene graph synchronized with the latest detections.

5) Node 5. Interpretation: This node begins by offering the user two options for sourcing the scene graph: loading it from a JSON file or subscribing to the `/scene_graph_json` topic. This flexibility enables manual modifications or enrichment of the graph prior to further processing. Once the source is selected, the node enters a loop in which it waits for the user to press Enter, captures and transcribes their speech, and submits the resulting text to GPT-4 along with the current scene graph and a carefully designed prompt. GPT-4 responds with a list of candidate objects that could satisfy the user's request. The node then evaluates which of these objects is closest to the robot's current position and publishes that object's pose as a `PoseStamped` message on `/goal_pose`. In this way, Nav2 can plan and navigate the robot autonomously to the selected object.

Fig. 3: Logic of the interpretation node: it interprets user requests, searches the scene graph, and sends a goal to the nearest matching object. Cyan markers in the 3D map indicate detected umbrellas.

Listing 1: Prompt for Scene Graph Generation

```
You are an expert scene graph architect with extensive
experience building JSON-based graph structures.
Receive a flat list of rooms and objects (each object
has id, label, coordinates, description; each room has
id, label, description). The list is ordered: a room
entry, followed by its objects, then the next room, and
so on. Build a directed graph representation with two
top-level keys: nodes and edges. nodes is an array of
all entities, each with keys: id, type (``building'',
``room'', or ``object''), label, description, and
coordinates (for objects) or omitted for non-objects.
edges is an array of relationships, each with keys:
source_id, source_label, target_id, target_label,
and relation. Create a node for the building with
id 0, type ``building'', label ``Building'', and
empty description. For each room, create a node (type
``room'') and an edge from building 0 to that room
with relation ``contains'', using the room's label
as source_label. For each object, create a node (type
``object'') and an edge from its containing room to
the object with relation ``contains'', including both
labels. If an object's description mentions spatial
relations (``in front of'', ``behind'', ``under'', ``on
top of'', ``next to'', etc.), create edges between
that object and the referenced object only if the
referenced object exists in the list, with keys:
source_id, source_label, target_id, target_label, and
the appropriate relation label. Do NOT create edges to
missing entities. Return ONLY the JSON with nodes and
edges; no prose or comments.
```

6) Node 6. Navigation: The Navigation node uses the Nav2 package with its default settings and no additional modifications. It subscribes to the geometric map produced by RTAB Map as a `nav_msgs/OccupancyGrid` message and to the robot pose information provided via TF or odometry. When the Interpretation node publishes a navigation goal as a `geometry_msgs/PoseStamped` message, Nav2 performs global path planning to compute a feasible route and then runs its local planning module to generate continuous velocity commands. These commands are published on the `/cmd_vel` topic, enabling the robot to drive autonomously toward the specified goal. An example of the navigation from the start point to the object "umbrella", answer in the experiments from the request "I think is going to rain in Tokyo" can be seen in the Figure 3

C. Assumptions and Limitations

NAIS faces four important limitations that merit consideration. First, relying on YOLOv8x (trained on the COCO dataset) for object detection introduces a constraint because the model cannot recognize categories beyond its training set. In addition, our three-stage filtering approach, while effective at removing false positives, also discards some valid detections.

For scene graph construction, our experiments assume a static environment in which objects remain in fixed positions. Under this setup, the graph does not update if an object moves or is removed. This assumption was made to simplify validation and isolate other performance variables. Nevertheless, the NAIS architecture is inherently capable of handling dynamic updates: because all perception modules run continuously and independently, enabling real-time scene graph refresh in response to environmental changes would require only minor modifications to the update triggers in the construction node.

NAIS uses the OpenAI API to build the scene graph and interpret user requests, which means that network latency and rate limiting can significantly affect response times and may vary depending on the volume of data transmitted. Another limitation arises from the physical construction of the robot: the TurtleBot 4's design places the RGB-D camera at approximately 40 cm above the ground, restricting its field of view to low-lying areas. As a result, the robot cannot perceive objects placed on top of desks or tables. This constraint dictated the experimental setup, where all target objects were intentionally positioned on the floor or slightly above the 40 cm threshold to ensure visibility.

Finally, the locally hosted LLaVA model demands substantial GPU memory to operate efficiently. During our experiments, running with only 7 GB of VRAM required periodic memory clearing and process restarts to avoid out-of-memory errors.

IV. Experimental Validation

We designed two navigation experiments to evaluate NAIS under different interaction modalities. Prior to each trial, both the occupancy map and the scene graph were generated and remained static throughout the run. Figure 4 illustrates the step-by-step process the robot used to build the experimental environment map. The map is visualized in Gazebo, a widely used simulator for autonomous navigation [12].

1) Experiment 1: Intuitive Language Requests: In the first experiment, participants issued intuitive natural-language commands such as "I am thirsty," "I am tired," or "I am bored." For each command, the robot was placed at a different starting location. We measured the success rate, defined as the proportion of trials in which the robot reached an object that satisfied the user's intent. We recorded node publication rates and monitored CPU, RAM, GPU, and VRAM usage during each run.

2) Experiment 2: Referential Language Requests: The second experiment focused on referential commands that leveraged the enriched object descriptions, for example

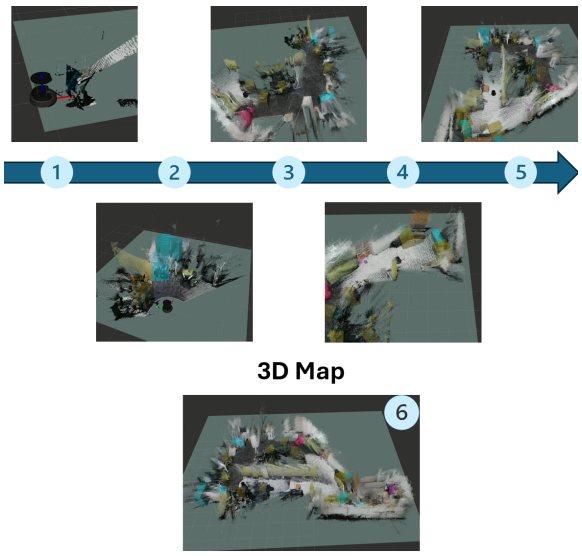

Fig. 4: Step-by-step construction of the 3D environment map, from initial scans (1–5) to the completed map (6).

TABLE II: Success rates for intuitive and referential commands

Command	% Success (JSON)	% Success (Topic)
I am thirsty	100	100
I am tired	100	100
I am bored	100	100
I want the grey backpack	60	60
I'm searching the blue chair	80	80

"Bring me the red backpack," "I am looking for the pink bottle," or "Find the laptop in front of the sofa." Again, the robot began each trial at a different location, and we computed the success rate in reaching the specified object. We also logged per-node latencies and overall system resource consumption as in Experiment 1.

3) Trial Protocol: For each experiment, we conducted ten trials using five distinct commands, randomized across starting positions. Table II will summarize the success rates for each command in both JSON-loaded and topic-subscribed modes. Table III will report the average publish rate of each node and the mean CPU, RAM, GPU, and VRAM usage of the complete system.

As shown in II, the NAIS system successfully interprets intuitive commands with near-perfect accuracy. This result is expected, as in these cases the Large Language Model (LLM) primarily processes the command and identifies potential

TABLE III: Per-node publish rates and system resource usage

Node	Hz	CPU	RAM	GPU	VRAM
Mapping	0.999	2.8	1.14	0.0	0.0
Object Detection	0.163	11.4	2.4	46.0	.892
Object Description	1.007	43.8	1.4	30.0	6.478
Interpretation	1.495	1.4	.341	0.0	0.0

Note: Hz indicates node publish rate; CPU and GPU usage are given in percentages; RAM and VRAM usage are given in gigabytes.

objects within the scene graph that can satisfy the user's intent. The reasoning pipeline in such scenarios is therefore more straightforward. In contrast, for requests requiring more contextual or referential information about the environment, a decrease in success rate was observed. This reduction is mainly attributed to the Vision Language Model (VLM), which did not always generate consistent or complete object descriptions. This limitation could be mitigated through improved prompt engineering, by defining the expected output more precisely and providing additional in-context examples to guide the model's responses. Regarding system performance (III), the Object Description node consumed the largest portion of GPU resources (30%) and VRAM (6.5 GB), reflecting the computational cost of VLM-based semantic enrichment. Nonetheless, overall latency remained acceptable for real-time operation (1 Hz for most nodes), confirming the feasibility of deploying NAIS in real-world environments.

These results highlight an inherent trade off between semantic richness and computational efficiency. While the integration of large scale vision language models enhances contextual understanding and enables flexible interpretation of user intent, it also introduces variability in output consistency and increased computational load. Future improvements should therefore focus on optimizing semantic generation either through lightweight multimodal models or adaptive prompting strategies to balance accuracy, robustness, and real time performance within constrained onboard resources.

V. Conclusions

Our experiments demonstrate that NAIS is a practical framework for autonomous indoor navigation driven by natural language commands that leverage contextual information from the scene. In intuitive tasks, the system consistently achieved success rates of up to 100% in office environments. However, in referential tasks, we identified an issue with object description generation that will require further refinement through prompt engineering, as the LLaVA model occasionally produces the requested descriptions accurately but at other times generates them incompletely. These results validate the advantages of integrating vision–language models with large language models for enriched map construction and precise command interpretation. Compared to prior scene-graph-based approaches, NAIS uniquely combines continuous SLAM, semantic enrichment, and natural language reasoning within a modular ROS 2 pipeline, enabling low-latency updates and straightforward extensibility.

Although NAIS performed well under test conditions, several improvements remain. Updating the scene graph to reflect dynamic environments will require more advanced object detection. Since the architecture already supports continuous perception and modular execution, enabling real-time updates would only need minor adjustments to the construction node's triggers. Using detectors with broader coverage and higher accuracy would further enhance versatility. A stronger recognition model could also simplify room transition logic and improve navigation across larger spaces. Finally, optimizing JSON generation for the scene graph will reduce overhead and increase responsiveness. Together, these enhancements will strengthen NAIS and support deployment in more complex real-world settings.

VI. Acknowledge

This paper includes the results of Cross-ministerial Strategic Innovation Promotion Program (SIP) 3rd Phase, "Expansion of fundamental technologies and development of rules promoting social implementation to expand HCPS Human-Collaborative Robotics" promoted by Council for Science, Technology and Innovation(CSTI), Cabinet Office, Government of Japan. (Project Management Agency: New Energy and Industrial Technology Development Organization (NEDO), Project Code: JPJ012494, HCPS: Human-Cyber-Physical Space)

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of SLAM: Towards the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] I. Kostavelis and A. Gasteratos, "Semantic mapping for mobile robotics tasks: A survey," *Robotics and Autonomous Systems*, vol. 66, pp. 86–103, 2015.

[3] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, "3d dynamic scene graphs: Actionable spatial perception with places, objects, and humans," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, p. 3133–3140. [Online]. Available: https://doi.org/10.1109/ICRA40945.2020.9197078

[4] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on robot learning*. PMLR, 2023, pp. 287–318.

[5] I. Armeni, Z.-Y. He, J. Gwak, A. R. Zamir, M. Fischer, J. Malik, and S. Savarese, "3D Scene Graph: A structure for unified semantics, 3D space, and camera," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 5664–5673.

[6] M.-F. Chang, C. Agia, and A. Gupta, "Verigraph: Scene graphs for execution verifiable robot planning," 2024. [Online]. Available: https://arxiv.org/abs/2411.10446

[7] Q. Gu, A. Kuwajerwala, S. Morin, K. M. Jatavallabhula, B. Sen, A. Agarwal, C. Rivera, W. Paul, K. Ellis, R. Chellappa, C. Gan, C. M. de Melo, J. B. Tenenbaum, A. Torralba, F. Shkurti, and L. Paull, "Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning," *arXiv preprint arXiv:2309.16650*, 2023. [Online]. Available: https://arxiv.org/abs/2309.16650

[8] N. Werby, S. Liu, and S. Song, "Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation," 2024. [Online]. Available: https://arxiv.org/abs/2403.17846

[9] T. Shan, A. Rajvanshi, N. Mithun, and H.-P. Chiu, "Graph2nav: 3d object-relation graph generation to robot navigation," *arXiv preprint arXiv:2504.16782*, 2025. [Online]. Available: https://arxiv.org/abs/2504.16782

[10] M. Safaldin, N. Zaghden, and M. Mejdoub, "An improved yolov8 to detect moving objects," *IEEE Access*, vol. 12, pp. 59 782–59 806, 2024.

[11] W. Liu *et al.*, "Temporal voting and spatial de-duplication for robust 3d object detection," in *Proceedings of ICRA*, 2023.

[12] J. M. Flores Gonzalez, E. Coronado, and N. Yamanobe, "Ros-compatible robotics simulators for industry 4.0 and industry 5.0: A systematic review of trends and technologies," *Applied Sciences*, vol. 15, no. 15, p. 8637, 2025.