CENTRALIZED INTELLIGENCE FOR DYNAMIC SWARM NAVIGATION

Team 53

End Term Submission Report

BHARAT FORGE | KALYANI

Inter IIT Tech Meet 13.0

Content

1 Approach Outline

1.1 Problem Understanding

Indoor autonomous navigation is crucial for industries like **warehousing**, manufacturing, and logistics, where GPS is ineffective and manually placed markers are impractical. Dynamic environments with unpredictable obstacles demand **intelligent systems** for safe, efficient navigation. This project develops an AI-driven solution enabling **Autonomous Mobile Robots (AMRs)** to navigate independently using a robust control algorithm for **real-time path planning**, **dynamic obstacle avoidance**, and collaborative task execution. A continuously updating **environmental map** stores data about obstacles and changes, ensuring scalability for larger environments and robot swarms. Additionally, a **chatbot interface**, powered by a large language model, streamlines task assignments and provides real-time updates, enhancing usability.

1.2 Final Approach and Workflow

The system utilizes a **Central Nervous System (LLM)** to process textual inputs and issue commands to a **scheduler** which further passes it to unit robots. The scheduler also monitors **robot states** and optimizes task allocation. Each unit robot performs functions like **SLAM** for mapping and localization, and **stereo depth** sensing and **Instance Segmentation** to generate a **semantic map**. This map updates the map interpreter which informs the Scheduler about the changes. The robots autonomously handle **task-specific navigation**, **manipulation**, and **exploration**, ensuring dynamic and coordinated swarm navigation.

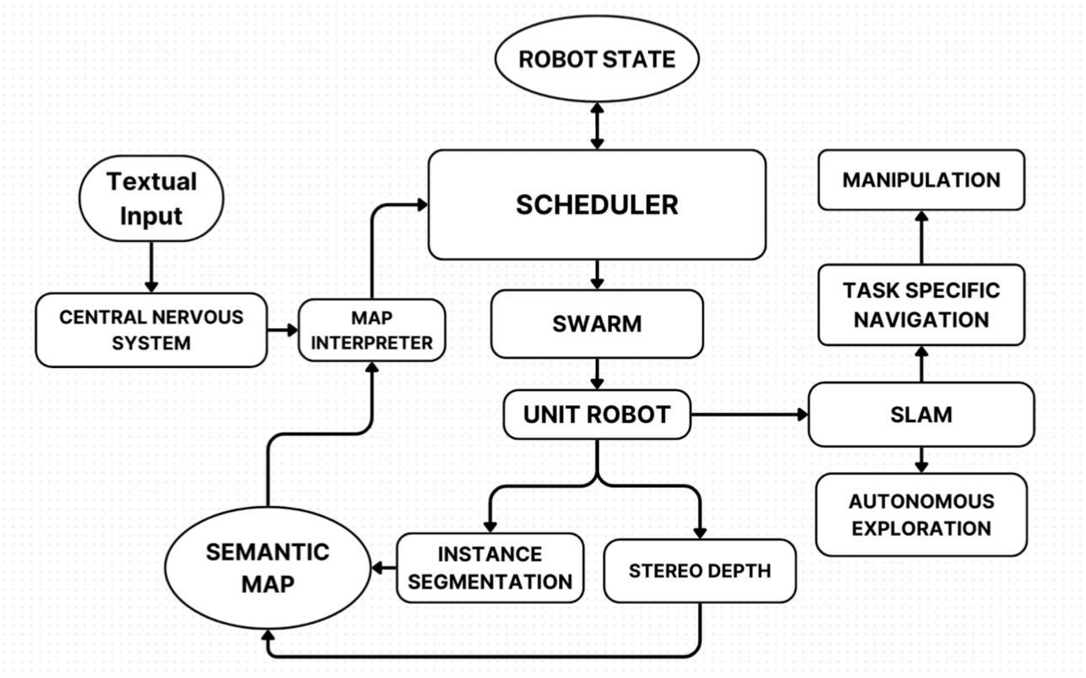
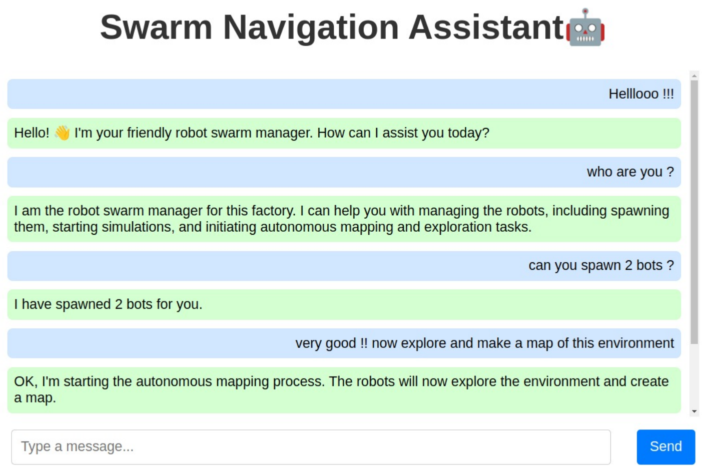


Figure 1: Workflow Diagram

Central Nervous Functioning

The **Gemini 1.5 Flash** chatbot uses advanced **transformer architecture** and **zero-shot learning** to handle tasks like bot spawning and exploring without task-specific examples. Leveraging **attention mechanisms**, it processes complex inputs efficiently, focusing on relevant details to enhance accuracy. Currently our chatbot model just analyzes the prompt and returns an intent in special format with which it was instructed in zero shot prompt along with the chatbot response which is further transferred to another function which uses regular expression library of python and processes intent and runs specific commands for our operational uses along with arguments given by user needed for that script.


Swarm Navigation Assistant🤖

Helllooo !!!

Hello! 👋 I'm your friendly robot swarm manager. How can I assist you today?

who are you ?

I am the robot swarm manager for this factory. I can help you with managing the robots, including spawning them, starting simulations, and initiating autonomous mapping and exploration tasks.

can you spawn 2 bots ?

I have spawned 2 bots for you.

very good !! now explore and make a map of this environment

OK, I'm starting the autonomous mapping process. The robots will now explore the environment and create a map.

Figure 2: Chatbot Interface

Scheduler

The **scheduler** is the core decision-making unit that manages bot activities by defining their states—**idle**, **exploration**, or **occupied**—and coordinating tasks using inputs from the central nervous system. All bots start in the idle state by default and remains stationary until instructed otherwise. We can change their state to exploration mode to navigate and map their surroundings. These maps are then shared to central nervous system. When a new task arises, the scheduler assigns it to the **nearest available bot (matric)** in either the idle or exploration state, calculating distances and prioritizing the closest bot. If the nearest bot is occupied, the scheduler checks the next closest bot, ensuring ongoing tasks are not disrupted.

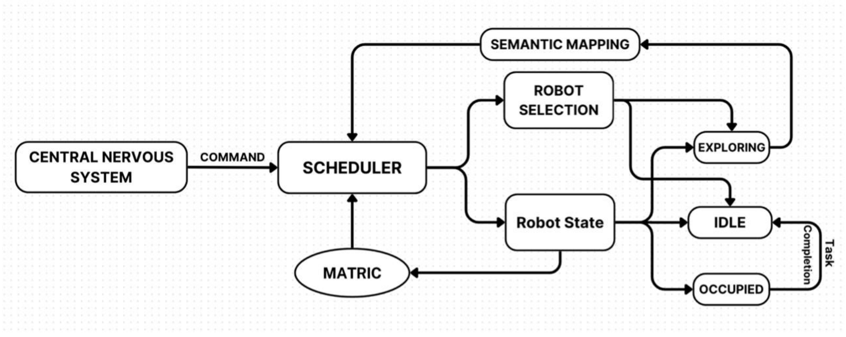


Figure 3: Scheduler Flowchart

Interpreter

The Interpreter acts as a critical node in the system, bridging the chatbot interface and the swarm control mechanism. It processes user input from the chatbot, identifies the most relevant mathematical representation from the semantic map, and formulates a goal to be passed to the Scheduler for execution. This ensures seamless communication between the user and the swarm management system.

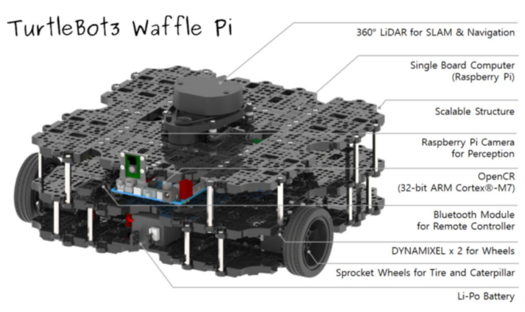
Swarm

The **swarm control system** is designed to efficiently manage and coordinate multiple robots by enabling them to share real-time data, maps, and positional information. When the central control system, acting as the ”nervous system,” issues a command to deploy one or more robots, it activates the swarm. In **exploration mode**, each robot independently navigates the environment, continuously gathering data to build and update a **global semantic map**. This map includes detailed information about the physical layout, key objects, and landmarks. By sharing their findings with the central system and other bots, each robot contributes to a collective understanding of the environment.

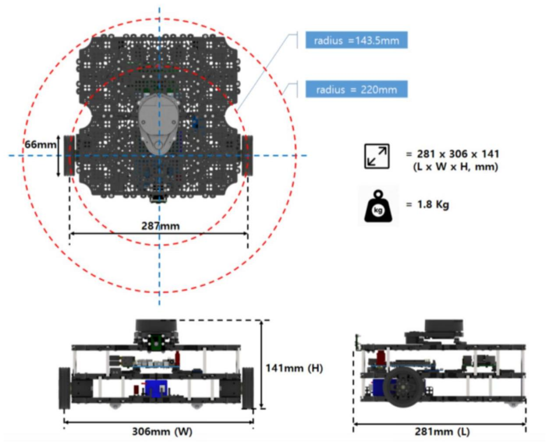
In **occupied mode**, when a robot is executing a specific task, it utilizes the existing semantic map built by other robots to navigate efficiently to its destination and this is possible due to swarm acts as a central system . This collaborative approach ensures that each robot benefits from the collective knowledge, saving time and resources by avoiding repetitive mapping efforts. As the robots continue to perform tasks, they also update the central map with new information, maintaining its accuracy and relevance.

Unit Robot

We are using the TurtleBot3 Waffle Pi is an open-source, ROS standard platform widely used for robotics research and product prototyping. It features essential sensors such as **LIDAR**, which uses laser beams for distance measurement and environment mapping (SLAM), an **IMU** (Inertial Measurement Unit) for tracking orientation and motion stability, a **Depth camera** for capturing 3D spatial information, and an **RGB camera** for visual recognition tasks. Operating on a differential drive mechanism with two independently controlled wheels, it enables precise movement and maneuverability by adjusting the wheels' speed or direction.


TurtleBot3 Waffle Pi
360° LiDAR for SLAM & Navigation
Single Board Computer (Raspberry Pi)
Scalable Structure
Raspberry Pi Camera for Perception
OpenCR (32-bit ARM Cortex®-M7)
Bluetooth Module for Remote Controller
DYNAMIXEL x 2 for Wheels
Sprocket Wheels for Tire and Caterpillar
Li-Po Battery

(a) Sensors



(b) Dimensions

SLAM

SLAM (Simultaneous Localization and Mapping) forms the backbone of swarm navigation by building the map of its environment while simultaneously determining its position within that map, without any prior knowledge of the surroundings. Unlike traditional methods where mapping and localization are done separately, **SLAM** updates both the map and the robot's position in real-time. It uses a **graph-based optimization** approach, where the robot's path is represented as a graph of poses (positions) connected by edges, with the graph being optimized to determine the most accurate real-time position. While the robot moves, it uses sensors like **LIDAR**, **cameras**, and **IMUs** to gather data and create an accurate map of the environment. The process relies on **sensor fusion**, combining data from multiple sensors to improve accuracy and minimize errors in both the map and localization.

Navigation

The **Nav2 stack** in ROS2 is a powerful framework for autonomous navigation, integrating key components to enable efficient movement and exploration. It starts with **costmaps**, which represent the environment using grids to indicate free, occupied, and unknown spaces. The **global costmap** covers large areas for long-range planning, while the **local costmap** focuses on the robot's immediate surroundings. The **Planner Server** uses the **A*** algorithm to generate optimal paths, and the **Controller Server** ensures the robot follows the path by adjusting movement commands in real-time. The **Behavior Tree** manages the overall decision-making, coordinating the components and triggering actions like path re-planning or obstacle avoidance as needed. This seamless integration allows the robot to autonomously navigate complex environments, continuously adapting to dynamic changes and ensuring robust, efficient exploration and navigation.

Autonomous Exploration

Frontier detection is essential for autonomous exploration, identifying unexplored areas at the boundary of known and unknown spaces. Greedy frontier-based exploration in ROS2 autonomously creates environment maps by exploring until no frontiers remain. This minimizes human intervention, automating the environment mapping and easing robot swarm navigation in previously unknown areas. The algorithm selects the next frontier based on immediate gain, such as proximity. **Cost mapping** assigns traversal costs based on obstacles or terrain, with robots using global cost-maps for planning and local ones for real-time navigation. Pathfinding algorithms like A* use cost-maps to find efficient, safe routes.

1. Defining the Frontier

A frontier cell is defined as a free cell adjacent to at least one unknown cell:

$$M(x,y) = \begin{cases} 0 & \text{if the cell is free} \\ 1 & \text{if the cell is occupied} \\ -1 & \text{if the cell is unknown} \end{cases}$$

A frontier $F(x,y)$ satisfies:

$$F(x,y) = \text{True} \quad \text{if} \quad \exists\,(x',y') \quad \text{where} \quad M(x',y') = -1 \quad \text{and} \quad M(x,y) = 0$$

2. Frontier Detection Algorithm

The algorithm identifies frontiers by finding free cells next to unknown cells:

$$\text{Frontiers} = \{(x,y) \mid M(x,y) = 0 \text{ and } \exists\,(x',y') \text{ where } M(x',y') = -1\}$$

3. Cost Function for Frontier Selection

A cost function is used to select the frontier based on distance and size:

$$\text{Cost}(F) = \alpha \cdot d(\text{robot}, F) - \beta \cdot A(F)$$

Where:

- $d(\text{robot}, F)$: Distance from robot to frontier.
- $A(F)$: Area of the frontier.
- α, β: Weights for distance and area.

4. Path Planning

The robot plans a path to the selected frontier by minimizing total cost:

$$\pi = \text{argmin}_\pi \sum_{i=1}^{n} \text{Cost}(p_i)$$

Where:

- p_i: Waypoints along the path.
- $\text{Cost}(p_i)$: Cost of moving through each waypoint.

5. Multi-Frontier Exploration

In multi-frontier exploration, the robot selects the frontier with the lowest cost:

$$\text{Cost}(F) = \alpha \cdot d(\text{robot}, F) - \beta \cdot A(F)$$

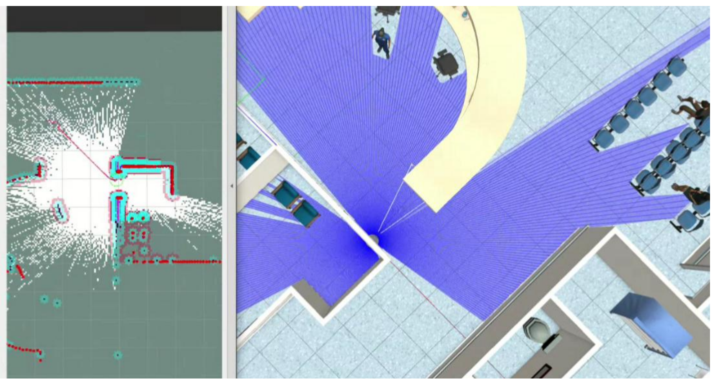


Figure 5: Mapping with Frontier Search

Instance Segmentation

We first collected data from various photos and videos captured from various environments, then applied several **preprocessing** techniques to enhance the data's quality and consistency. These techniques included auto-orienting the images to correct their orientation, performing static cropping to focus on relevant areas, resizing the images to a consistent size, and **auto-adjusting the contrast** to improve visibility. After preprocessing, we **augmented** the images by flipping them, converting them to grayscale, adjusting their hue, saturation, exposure, and adding noise to simulate real-world variations. These steps were designed to increase the diversity of the dataset and help the model generalize better. Finally, we trained the **YOLO v8n-seg** model for **instance segmentation**, using various **hyperparameters** to optimize its performance for **object detection** and **segmentation** in our environment.

YOLOv8 Segmentation Loss Components:

1. Localization Loss The localization loss optimizes how well predicted bounding boxes align with the ground truth, using Intersection over Union (IoU) and Complete IoU (CIoU):

$$\mathcal{L}_{\text{loc}} = \lambda_{\text{box}} \left(\text{IoU}(\hat{b}, b) + \text{CIoU}(\hat{b}, b) \right)$$

where $\hat{b}$ is the predicted box and b is the ground truth.

2. Classification Loss Classification loss measures how well the predicted class matches the ground truth class, formulated as:

$$\mathcal{L}_{\text{cls}} = -\sum_{i} 1_{i=\hat{y}} \log(p_i)$$

where $\hat{y}$ is the predicted class, and p_i is the predicted probability for class i.

3. Segmentation Loss Segmentation loss compares the predicted mask with the ground truth using Binary Cross-Entropy (BCE):

$$\mathcal{L}_{\text{seg}} = E\left[-\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \right]$$

where y_i and $\hat{y}_i$ are the ground truth and predicted masks, respectively.

4. Total Loss The total loss is a sum of the above losses:

$$\mathcal{L} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{seg}}$$

This combined loss optimizes the model for object detection, classification, and segmentation.

We trained our model on a custom dataset with the following hyperparameters: epochs set to 50, an initial learning rate $(lr0)$ of 1×10^{-6}, a cosine learning rate scheduler enabled, a dropout rate of 0.2, and 6 layers frozen. The training graphs are given below:

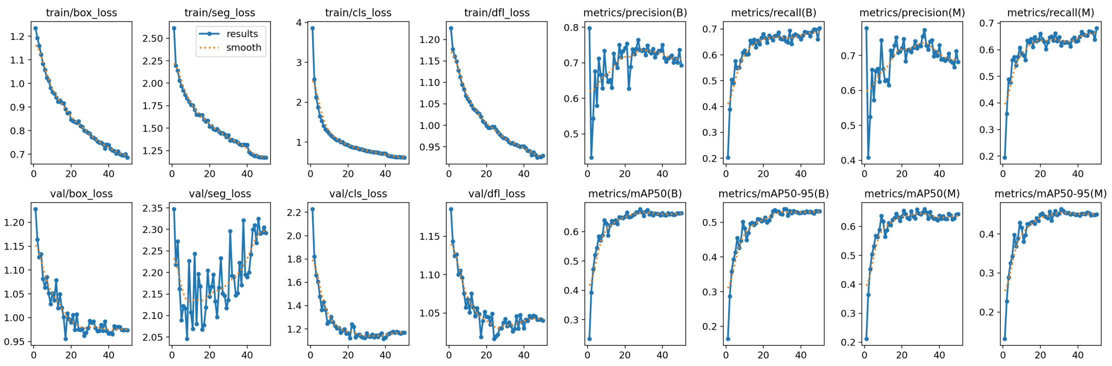
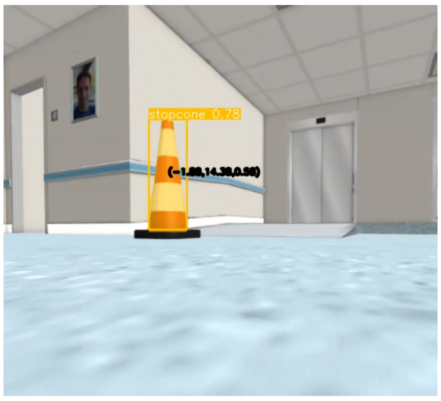


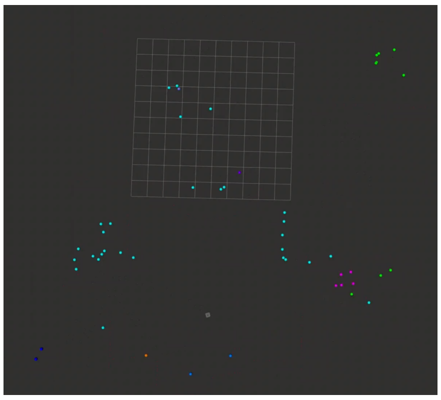
Figure 6: Training Graphs

Semantic Mapping

The process begins with the robot using its **camera** to capture real-time visual data of the environment, which is then fed into the **YOLOv8 segmentation model**, a state-of-the-art object detection model designed for fast, accurate detection in dynamic environments. YOLOv8 segments and identifies objects, providing initial coordinates in the camera's frame.
Once objects are detected, the next step is to **calculate the coordinates** relative to the bot's depth camera view. These coordinates are initially in the camera's local frame of reference, and they are **transformed into a global map frame**, aligning them with the broader environmental context for navigation.
The transformed coordinates are stored in a **database** for further processing. The system calculates **pairwise distance** between detected objects to filter out **noise**, removing objects whose distances exceed a predefined threshold to retain only relevant data.



(a) Instance Segmentation



(b) 3D Semantic Point Cloud

1.3 Background Study

To address the problem statement, a strong understanding of several core technologies is essential. **ROS2 odometry** estimates a robot's position from sensor data, but **ROS2 localization** (e.g., 'amcl') corrects drift using LIDAR or camera inputs. **Sensor fusion** combines data from IMU, GPS, and LIDAR with Kalman filters for improved accuracy. **SLAM (Simultaneous Localization and Mapping)** creates maps and tracks position, essential for navigating unexplored areas using tools like 'slam-toolbox' in ROS2. **Semantic mapping** adds context by labeling maps with object detection models, such as YOLO, enabling context-aware navigation.

Nav2 in ROS2 handles path planning and obstacle avoidance, using algorithms A* and costmaps for safe navigation. Effective **data engineering** manages large datasets from mapping and sensor fusion, including data pipelines and storage systems. **Instance segmentation** allows robots to recognize and map objects accurately. Finally, **chatbot LLMs** (e.g., Gemini) provide a natural interface for diagnostics, task assignments, and user interaction, improving system control and communication.

2 Analysis and Experiments

2.1 Simulation Setup

The initial simulations were conducted with the Gazebo classic simulator. Here are some key configurations:

- The robot spawning process was optimized by **parameterizing** each **XACRO file**, assigning unique namespaces to each robot for isolated operation and resource management.
- Custom Gazebo model plugins were integrated to introduce **dynamic obstacles**, allowing robots to navigate in a more complex, evolving environment.
- A **depth camera sensor** was added to the Turtlebot3 Waffle Pie, enhancing its ability to perceive and map objects in 3D space.

2.2 Performance Analysis

After rigorous testing and debugging, our instance segmentation model achieves an **mAP50 of 76%**, indicating a high level of accuracy in detecting and segmenting individual objects within a given scene. This performance is particularly strong in enclosed environments, where the robot can more easily differentiate between obstacles and free space.

The model performs even better in settings with a large number of obstacles, as the dense arrangement of objects helps the system leverage its segmentation capabilities effectively.

3 Challenges and Lessons Learned

3.1 Technical Challenges

Several technical challenges have been encountered so far:

- **Resource Issues:** As the swarm size increases, the computational speed decreases, making it challenging to launch more than five bots in a single video demonstration.
- **Time Constraints:** Due to time constraints, we were unable to optimize our packages further and could not port our Python packages to C++.

3.2 Future Scope and Recommendations

The following implementations can elevate the project and enable the robots to reach an entirely new level of capability:

- **Dynamic Map Merging:** A system where multiple robots contribute individual maps to a central system, which merges them into a unified map. This distributed approach speeds up mapping by dividing the workload, making the process faster and more efficient. The main challenge lies in dynamically updating the map as the environment changes and enabling effective exploration using the partially built map.
- **Robotic Arm Integration:** A robotic arm has been developed to autonomously reach designated points and manipulate objects. The next challenge is integrating it with the central control system to autonomously retrieve target coordinates from a semantic map and perform tasks without human intervention, making the arm a fully independent and efficient addition to its environment.
- **Swarm Map Exploration:** Integration of swarm exploration, which will enable the robots to collaboratively navigate and investigate unknown environments. By working together, the swarm can efficiently cover larger areas and adapt to changing conditions.
- **3D Semantic Mapping:** 3D semantic representation allows the swarm to build a detailed, three-dimensional understanding of its environment, incorporating both spatial and contextual information.

References

[1] S. Macenski and I. Jambrecic, “Slam toolbox: Slam for the dynamic world.” [Online]. Available: https://doi.org/10.21105/joss.02783

[2] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, “The marathon 2: A navigation system,” 2020. [Online]. Available: https://github.com/ros-planning/navigation2

[3] “Gemini 1.5 flash.” [Online]. Available: https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/gemini-1.5-flash-002?pli=1

[4] Y. Pyo and H. L. Allen, “Turtlebot3.” [Online]. Available: https://github.com/ROBOTIS-GIT/turtlebot3

[5] “yolov8.” [Online]. Available: https://yolov8.org

[6] “m-explore.” [Online]. Available: https://github.com/hrnr/m-explore

[7] J. G. Carlos Andrés Álvarez Restrepo and T. Clephas, “m-explore-ros2.” [Online]. Available: https://github.com/robo-friends/m-explore-ros2

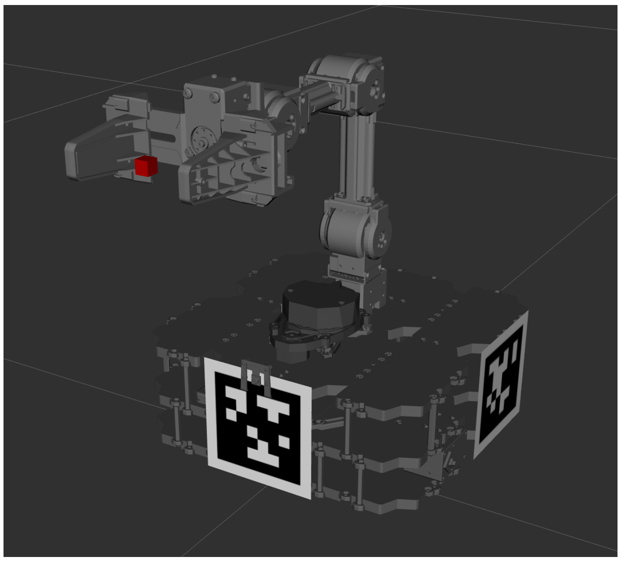
[8] “mapmergeformultirobotmapping-ros2.” [Online]. Available: https://github.com/abdulkadrtr/mapMergeForMultiRobotMapping-ROS2

[9] AndrejOrsula, “Deep reinforcement learning for robotic grasping from octrees.” [Online]. Available: https://github.com/AndrejOrsula/drl_grasping

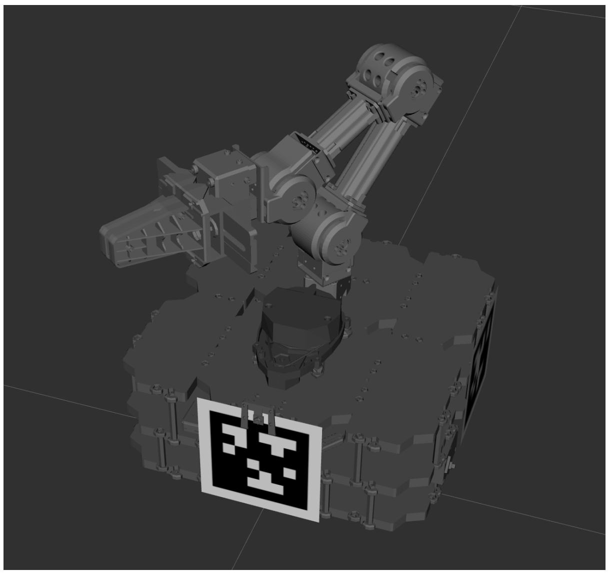
[10] J. Wallace and A. Wong, “aws-robomaker-hospital-world.” [Online]. Available: https://github.com/aws-robotics/aws-robomaker-hospital-world

[11] K. Deshpande and A. Wong, “aws-robomaker-small-warehouse-world.” [Online]. Available: https://github.com/aws-robotics/aws-robomaker-small-warehouse-world

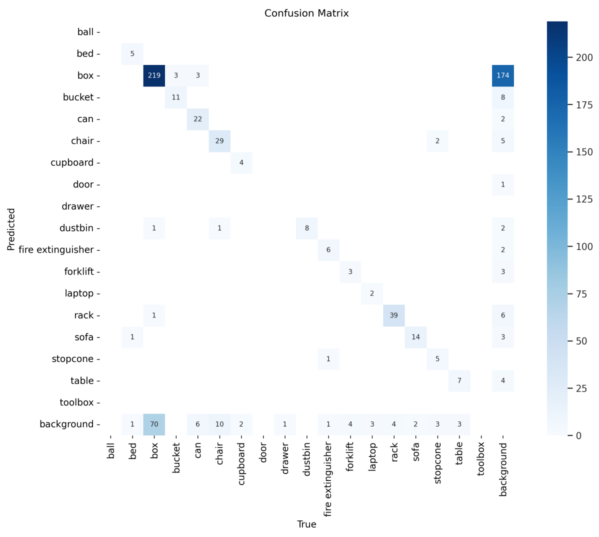
APPENDIX



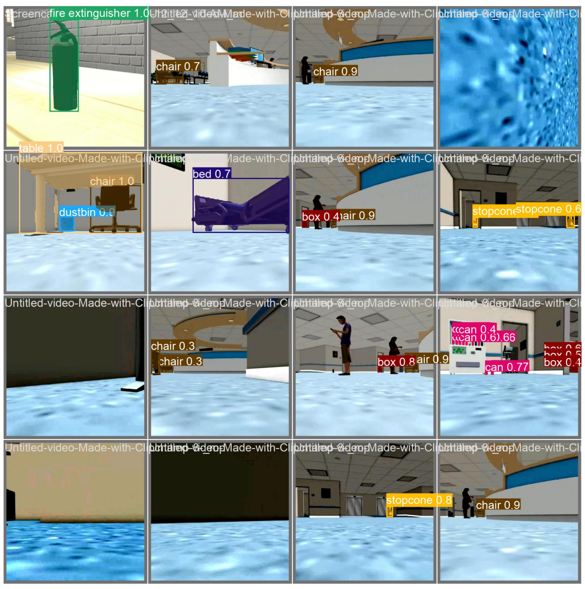
(a) FUTURE SCOPE : Arm Integration (GRIPPER OPENED)



(b) FUTURE SCOPE : Arm Integration (GRIPPER CLOSED)



(a) Validation Batch from yolov8 Segmentation



(b) Confusion Matrix from yolov8 Segmentation

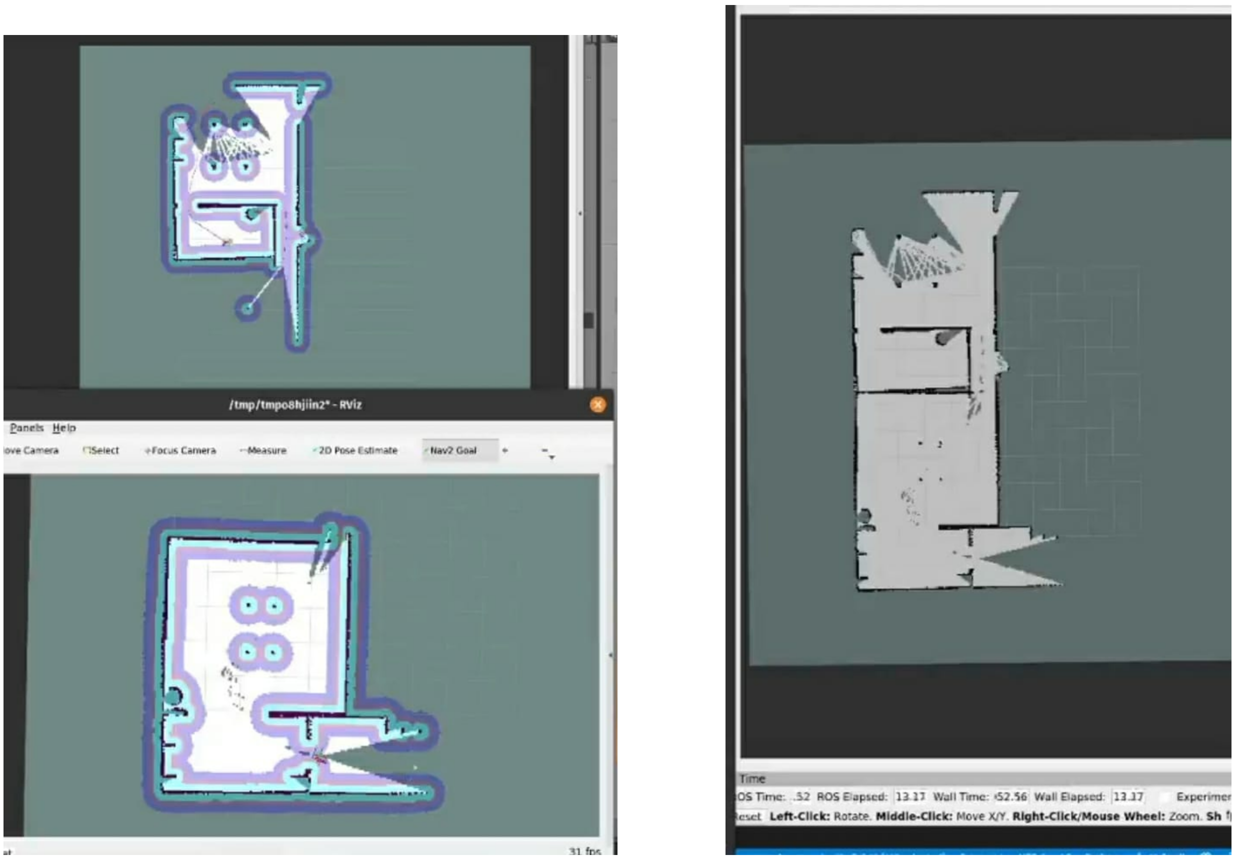

/tmp/tmpo8hjiin2* - RViz
Panels Help
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
Nav2 Goal
31 fps
Time
ROS Time: .52 ROS Elapsed: 13.17 Wall Time: 52.56 Wall Elapsed: 13.17
Experimen
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Sh

Figure 10: Future Scope (Map Merge): Left Side-Individual Maps ; Right Side-Merged Map