

TDK-thesis

Haichen Xu

Pedro Alejandro Cordero Jara

Wentao Zhang

LLM-based Task Planning for Mobile Robots Using a Novel SLAM Method for Materials Distribution

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPT. OF ARTIFICIAL INTELLIGENCE



Authors:

Haichen Xu

Pedro Alejandro Cordero Jara

Computer Science MSc

II. grade

Wentao Zhang

Computer Science MSc

I. grade

Supervisor:

Aphilak Lonklang

PhD Student

Budapest, 2024

Contents

1	Introduction	3
2	Literature Review	5
2.1	Mobile Robot for Material Distribution	5
2.1.1	Challenges in Material Distribution with Mobile Robots	6
2.2	SLAM Techniques for Mobile Robots	6
2.3	Path Planning	7
2.4	NLP and LLM	8
2.4.1	Role of NLP in Material Distribution by Mobile Robots	8
2.4.2	Challenges and Gaps	9
3	Methodology	10
3.1	Proposed SLAM	10
3.1.1	Sensor and Image Calibration	11
3.1.2	Image Correction	12
3.1.3	Object Detection	13
3.1.4	Fixed-points Generation	15
3.1.5	Occupancy Map Generation By Using Novel SLAM Method	17
3.2	NLP and LLM Orchestration	20
3.3	Communications Architecture	22
3.4	Hardwares	24
3.4.1	TurtleBot3 Waffle Pi with OpenManipulator-X	24
3.4.2	Camera: Basler a2A1920-160ucBAS	24
3.4.3	Server	25
4	Experiments and results	26
4.1	Camera Calibrartion	27
4.2	Object Detection Evaluation	27

4.2.1	Color-Based ObjectDetection	27
4.2.2	YOLO11 Model	28
4.2.3	Fixed Points	29
4.3	SLAM Evaluation	30
4.4	NLP Evaluation	31
4.5	Motion Planning Evaluation	32
5	Conclusion	34
5.1	Experimental Validation	34
5.2	Challenges and Future Works	35
	Bibliography	36
	List of Figures	41
	List of Tables	43
	List of Algorithms	44
	List of Codes	45

Chapter 1

Introduction

In modern industrial settings, automation plays a crucial role in streamlining operations and increasing productivity. Mobile robots are becoming integral to material distribution tasks, particularly in large factories where manual handling is time-consuming and prone to errors [1]. These robots are tasked with navigating complex environments, avoiding obstacles, and efficiently transporting materials between different workstations. However, achieving smooth, autonomous navigation and task execution in a dynamic factory environment poses significant challenges, particularly in path planning and human-robot interaction.

One of the key aspects of autonomous robots is their ability to interpret and act on human commands. In traditional systems, robot navigation is typically pre-programmed, limiting flexibility in fast-changing industrial scenarios. To bridge this gap, incorporating Natural Language Processing (NLP) and Large Language Models (LLMs) allows robots to understand and execute tasks based on spoken or written instructions. NLP enables human operators to directly communicate with robots, simplifying task assignment. By processing natural language queries, robots can identify target materials using camera capture and determine their exact location within the environment [2]. Recent advancements in NLP have made it possible for robots to interpret a wide range of instructions and perform complex tasks with high accuracy, as seen in applications such as smart factory systems [3].

Once the target coordinates are identified, efficient path planning becomes critical. Path planning algorithms like Rapidly-exploring Random Trees (RRT) [4] and its optimized version, RRT* [5], have been widely used for mobile robot navigation. However, these methods often require large map sizes and may struggle in dynamic environments

with frequent changes. To address these limitations, we propose the use of an Improved RRT* with Reduced Random Map Size (IRRT*-RRMS) [6], an algorithm that enhances path planning efficiency by reducing the complexity of the random map while maintaining optimal path finding performance. This approach is especially beneficial in factory settings where obstacles and layout changes are common.

In addition to optimizing path planning, we propose a novel method to create the robot's navigation map without relying on conventional SLAM (Simultaneous Localization and Mapping) techniques [7]. Instead of SLAM, the map is generated by taking a snapshot from a top-view camera and using color-based object detection to define key areas in the environment. These areas include the workspace, obstacles, and pick-and-place stations, which are color-coded to simplify detection and recognition. This technique significantly reduces the complexity of map generation and allows for rapid updates when the environment changes.

In this paper, we present a system that integrates NLP for task interpretation with the IRRT*-RRMS algorithm for path planning, and a top-view-based color detection method for map creation. This integrated system enables mobile robots to autonomously navigate and distribute materials in a factory environment. The combination of these technologies offers a flexible and scalable solution for real-time, autonomous material handling in industrial applications.

Chapter 2

Literature Review

2.1 Mobile Robot for Material Distribution

Mobile robots have become a cornerstone in modern industrial automation, particularly for material distribution tasks in factories and warehouses [8]. Their application spans industries, enhancing efficiency, reducing labor demands, and optimizing logistics operations. By employing mobile robots, businesses address labor shortages, lower operational costs, and improve productivity and safety in material handling.

Mobile robots transport raw materials, semi-finished goods, and finished products between workstations, ensuring a seamless workflow. Similarly, mobile robots facilitate dynamic storage and retrieval in warehouses, streamline inventory management, and boost throughput. Beyond industrial settings, healthcare facilities use mobile robots to deliver supplies, meals, and linens, easing staff workloads and ensuring timely service. These diverse applications highlight the adaptability of mobile robots across various material distribution needs.

The effectiveness of mobile robots in material distribution relies on advancements in several core technologies. Localization and mapping methods, such as traditional SLAM, enable robots to navigate complex environments [9]. More recently, vision-based mapping techniques have emerged as alternatives to predefined grid maps [10]. Path planning is another critical area, with classical methods like Dijkstra's and A* algorithms paving the way for advanced approaches such as IRRT* and Genetic Algorithm-based planners. Robots use sensors like LiDAR, cameras, and ultrasonics to detect obstacles and identify targets. Navigation systems combine planners for real-time obstacle avoidance and route optimization [11].

2.1.1 Challenges in Material Distribution with Mobile Robots

Despite their capabilities, mobile robots face several challenges in material distribution. One significant issue is adapting to dynamic environments where factory layouts and obstacles can change unpredictably. Another challenge is achieving effective load balancing, which requires optimizing robot utilization to minimize idle times and ensure efficient task distribution. Furthermore, inter-robot collaboration poses a complex problem, as coordinating fleets of robots demands robust communication protocols and intelligent decision-making algorithms to prevent collisions and optimize task division [12].

Although mobile robots have demonstrated remarkable progress in material distribution, certain research gaps remain. Fully autonomous and adaptive systems are yet to be realized, as current technologies struggle with real-time adaptation to dynamic and unpredictable factory environments. Further research is needed to enhance the integration of NLP systems and perception capabilities, enabling more intuitive human-robot interactions. Additionally, improving multi-robot coordination algorithms and developing scalable solutions for inter-robot collaboration represent key areas for future exploration.

2.2 SLAM Techniques for Mobile Robots

Simultaneous Localization and Mapping (SLAM) is a foundational technique in robotics, enabling mobile robots to navigate unknown environments by constructing a map while simultaneously estimating their location within it [13]. SLAM has been extensively researched and applied across diverse domains, from autonomous vehicles to indoor delivery robots [14]. In the context of material distribution, robust SLAM algorithms ensure precise navigation and efficient task execution. This section explores the evolution of SLAM, highlighting traditional methods, modern advancements, and their relevance to material distribution tasks.

In the realm of indoor mobile robotics, SLAM is pivotal, employing sensors like cameras, sonar, and lasers to create navigable maps that facilitate tasks such as autonomous navigation and search-and-rescue missions. Central to ROS, SLAM dynamically updates maps and tracks a robot's position, greatly enhancing mapping accuracy and reducing location-based errors, a critical feature for indoor settings like office environments.

With advances in sensor technology and computational power, modern SLAM techniques have integrated deep learning and multi-modal sensor fusion. Visual SLAM (vS-

LAM), which uses cameras as primary sensors, has gained significant traction due to its ability to generate rich, detailed maps [15]. Methods like ORB-SLAM and DSO (Direct Sparse Odometry) leverage feature-based and direct image analysis, respectively, to achieve robust performance in various scenarios [16] [17].

Lidar-based SLAM remains a preferred choice for environments requiring high precision, such as warehouses. Techniques like Cartographer by Google and Hector SLAM provide accurate mapping and localization using laser scans. The integration of Lidar with other sensors, such as cameras and IMUs (Inertial Measurement Units), has further enhanced SLAM performance, enabling robots to operate in dynamic and complex environments.

Despite significant progress, SLAM techniques face several challenges in real-world applications. Dynamic environments, such as factories, introduce moving objects that can confuse traditional mapping algorithms. To address this, researchers have proposed Dynamic SLAM techniques, which filter out moving objects during the mapping process [18]. Additionally, the reliance on high-quality sensors, such as Lidar, can increase system costs, limiting scalability in cost-sensitive applications.

Another limitation is map maintenance. In large-scale environments, maintaining and updating maps as the workspace evolves remains challenging. Techniques like lifelong SLAM and active SLAM have been developed to address these issues, allowing robots to explore and adapt their maps over time actively. While SLAM has reached a high level of maturity, there remain opportunities for further research and innovation. One promising direction is the development of lightweight SLAM algorithms tailored for resource-constrained robots. These algorithms must balance computational efficiency with mapping accuracy, enabling real-time operation on embedded systems.

2.3 Path Planning

For efficient global path planning, the Improved RRT* with Reduced Random Map Size (IRRT*-RRMS) algorithm [19] is employed. This approach builds on the standard RRT algorithm by narrowing the random sampling region to a reduced map, specifically focused on the robot's immediate operational area. This reduction in map size decreases computational load while maintaining high path optimality. Furthermore, IRRT*-RRMS incorporates path refinement techniques that optimize the generated trajectory, reducing travel distance and enhancing overall efficiency. These improvements make the IRRT*-

RRMS particularly well-suited for real-time navigation in complex environments, such as those found in industrial material distribution [20]. Figure 2.1 illustrates the path-planning process using the proposed algorithm. The blue triangle and green circle represent the start and goal points, respectively. The green lines depict potential paths explored by the algorithm, while the magenta line shows the initial solution generated by IRRT*-RRMS. This solution is then refined using Bacterial Mutation and Node Deletion algorithms [21], resulting in the final optimized path, represented by the red line.

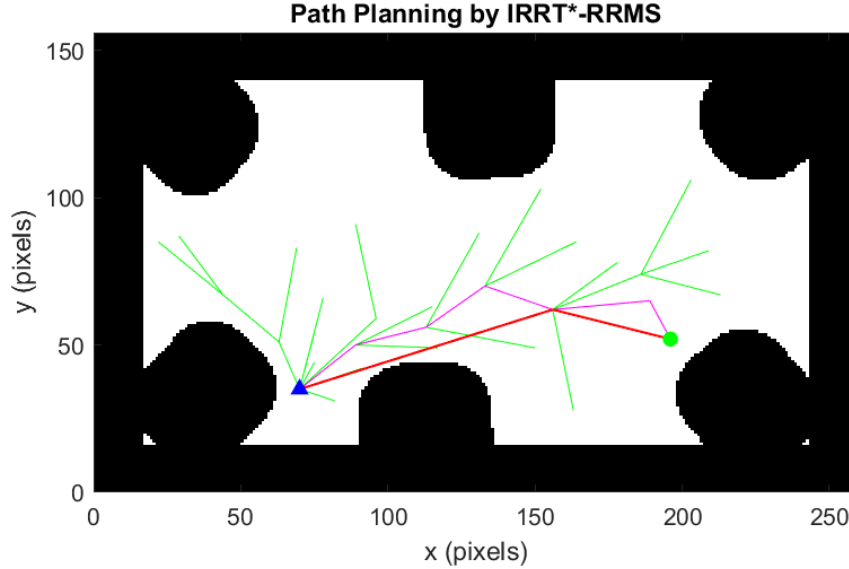


Figure 2.1: IRRT*-RRMS for Path Planning

2.4 NLP and LLM

NLP and LLM technologies have become pivotal in enhancing human-robot interaction, enabling robots to understand and execute complex commands. For mobile robots in material distribution, these tools offer intuitive communication interfaces and advanced reasoning capabilities [22].

2.4.1 Role of NLP in Material Distribution by Mobile Robots

NLP serves as a critical interface between humans and robots, enabling users to issue verbal or written instructions in natural language [23]. In material distribution tasks, it interprets user requirements, clarifies material specifications, and provides feedback on availability. Additionally, it relays environment mapping data and geometric information—generated through a novel SLAM method—to mobile robots via communication

protocols. This simplifies material distribution processes, eliminating the need for complex programming and making robotics accessible to non-experts [24].

Mobile robots utilize NLP to:

1. *Task Conversion*: Transform commands into tasks with accurate geometry.
2. *Sequential Planning*: Interpret and execute multi-step instructions.
3. *Interactive Feedback*: Allow robots to clarify or update, e.g., “Item unavailable. Proceed with next task?”

2.4.2 Challenges and Gaps

NLP and LLM-based systems face challenges in material distribution due to domain adaptation, high computational demands, error handling, and integration with physical systems. Large language models often struggle with specialized industrial terminology and require resource-intensive fine-tuning. Their computational requirements hinder real-time decision-making, and misinterpretation of commands can lead to task failures. Translating natural language instructions into robotic actions further complicates integration, requiring coordination between perception, planning, and control.

To overcome these issues, research should focus on developing lightweight LLMs for edge devices, enhancing context awareness for better handling of ambiguous instructions, and integrating NLP with vision-based systems for multi-modal understanding. Robust training with synthetic datasets and reinforcement learning can also boost task-specific performance, enabling efficient and adaptable mobile robots for material distribution in dynamic environments.

Chapter 3

Methodology

The proposed system workflow, depicted in Fig. 3.1, begins with capturing a top-view image of the environment using a fisheye lens-equipped camera. This image undergoes calibration and correction, followed by color-based and object detection to identify obstacles and key stations, such as the main and fastener stations. A SLAM-based occupancy map is then generated, establishing fixed navigation points. Users interact with the system via an NLP module that interprets fastener requests based on assembly models, leveraging a large language model trained on assembly manuals.

The Middleware, incorporating ROSbridge, processes the fixed-point map and material list, forwarding them to the motion planning module. Using the IRRT*-RRMS algorithm, an optimized path is generated for the robot to execute pick-and-place tasks at the designated stations. A motion controller ensures precise task execution, while real-time feedback to the Middleware and user enables smooth operation and obstacle avoidance, adapting seamlessly to dynamic scenarios.

3.1 Proposed SLAM

Conventional SLAM techniques are often computationally demanding and may struggle to adapt to fast-changing factory environments [25]. These challenges arise from the need for simultaneous localization and mapping, which are further complicated by uncertainty and noise in sensor data as well as the dynamic motion of the agent [26]. To address this, a novel method for map creation using a top-view snapshot and color-based object detection is proposed.

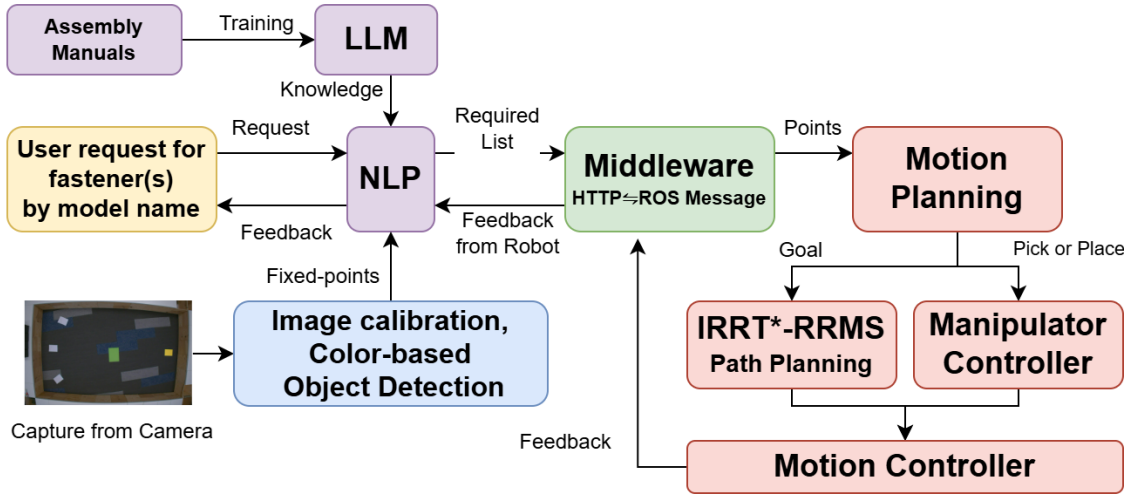


Figure 3.1: System Overview

3.1.1 Sensor and Image Calibration

Fisheye lenses are widely used in vision systems because they capture a wide field of view. However, these lenses introduce significant geometric distortions. The fisheye lens projections are characterized by an approximately linear relationship between the incidence angle of an object point's beam and the distance from the corresponding image point to the principal point [27]. This unique characteristic necessitates specific calibration techniques to model and rectify the distortion effects accurately.



Figure 3.2: Fisheye Calibration Chessboard Patterns

The provided Figure 3.2 showcases a series of calibration patterns using a chessboard, where each sub-image illustrates how the fisheye lens projects the chessboard onto the sensor plane, resulting in visible distortions, particularly along the edges of the images. These distortions highlight the importance of precise calibration and rectification techniques. The calibration process involves calculating the intrinsic parameters (focal length,

principal point) and distortion coefficients specific to the fisheye lens model, allowing the mapping of distorted points to their undistorted counterparts in real-world space. The fish-eye lens distorts grid lines, however, the chessboard pattern, with its well-defined corners, makes it an ideal calibration target. Therefore, corner detection becomes the first step in correcting the distortion.

The colored lines in the image represent the connections between the detected corners, demonstrating how the fisheye distortion warps the straight grid lines. These detected corner points are then mapped to their known 3D coordinates on the calibration pattern, forming the input data for the calibration algorithm. The intrinsic parameters (focal lengths and principal point offsets) and distortion coefficients are computed using the OpenCV fisheye calibration model. The calibration accuracy is evaluated using the re-projection error, which quantifies the difference between the observed points and the projected points, ensuring the validity of the calibration results. The Code 3.1 shows the process of obtaining the `OptimalCameraMatrix`.

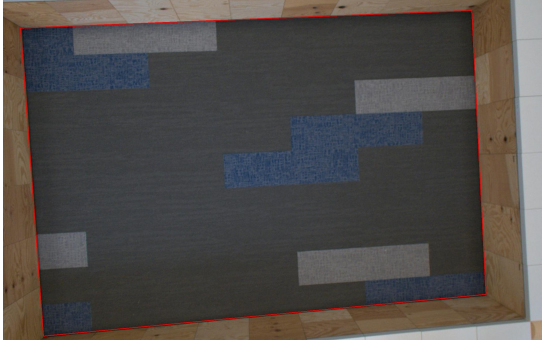
```
1 def getOptimal():
2     ...
3     for fname in glob.glob('*.jpg'): ...
4         ret, corners = cv2.findChessboardCorners(gray, (8, 9), None)
5         if ret: ...
6             img_points.append(cv2.cornerSubPix(
7                 gray, corners, (5, 5), (-1, -1), criteria) or corners)
8     ret, mtx, dist, _, _ = cv2.calibrateCamera(
9         obj_points, img_points, (w, h), None, None)
10    ...
```

Code 3.1: Partial Code of Undistort

3.1.2 Image Correction

After fisheye correction, a perspective view is obtained. To achieve an accurate overhead view for mapping purposes, further calibration is performed on the perspective view to convert it into an orthogonal top-down view. The corrected image provides a comprehensive overview of the robot's workspace, including critical areas such as obstacles and task stations. Additionally, the top-down camera can capture regions that are sometimes beyond the reach of the robot's LiDAR scanner, thereby compensating for the limitations

of the robot's perspective. Figure 3.3 shows the conversion from perspective (Figure 3.3a) to accurate top view (Figure 3.3b).



(a) Perspective Image



(b) Orthogonal Top-Down view

Figure 3.3: Comparison of point selected image and cropped image

3.1.3 Object Detection

Color-based Object Detection

This method identifies and marks key features on the map, including stations, main stations, and obstacles, as shown in Figure 3.4a. The image is converted to the HSV color space, where a defined color range is used for segmentation. This process involves creating color masks and extracting different contours to detect and classify the various elements in the environment. By utilizing color segmentation, the occupancy map is generated, simplifying the overall task of environment classification.

The use of fisheye lens calibration combined with color-based object detection creates a flexible and efficient map generation method. It is well-suited for dynamic factory environments, eliminating the need for complex SLAM systems while ensuring accurate, up-to-date maps. An example of color-based object detection is shown in Figure 3.4b, which identifies and classifies objects within the workspace. Not only can different shapes and colors be detected, but objects can also be classified based on their orientation and type, such as stations, main stations, circular obstacles, and square obstacles.

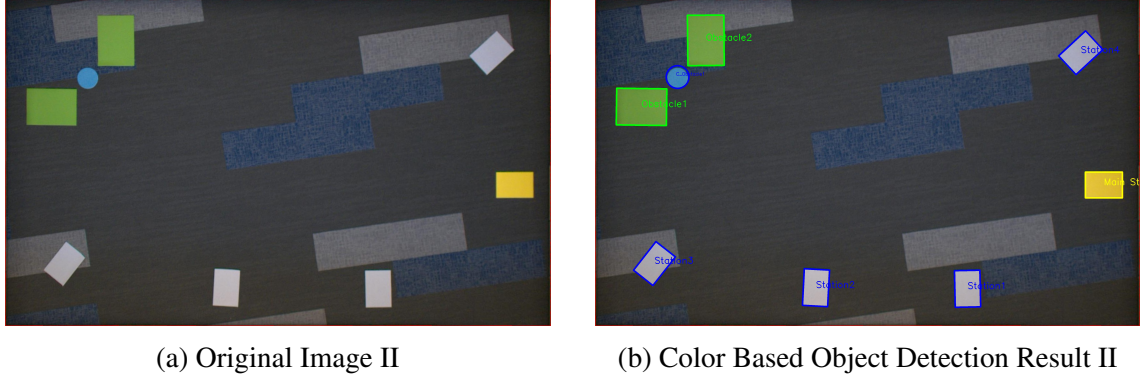


Figure 3.4: Comparison of Original Image and Color-Based Detection

Object Detection with YOLOv11

YOLOv11, the latest in the Ultralytics YOLO series, offers cutting-edge accuracy, speed, and efficiency with improved architecture and training methods, making it versatile for diverse computer vision tasks [28].

```

1 def train():
2     model = YOLO("yolo11m-obb.pt")
3     results = model.train(data="data.yaml", workers=0, epochs=50, batch=8)

```

Code 3.2: YOLO Simple Train

YOLOv11m Oriented Bounding Box (OBB) model was selected due to the poor performance of zero-shot models in detecting small objects commonly found in map images. To enhance its effectiveness, YOLO was fine-tuned on a customized dataset. The dataset was annotated using CVAT [29], and the model was trained with the code provided in Code 3.2. The OBB model, incorporating orientation information, was specifically chosen to meet the requirements of the Fixed Points algorithm, enabling the identification of the specific side of a station where a robot can dock.

```

1 @app.route('/Detect', methods=['POST'])
2 def post_detect():
3     np_arr = np.frombuffer(request.data, np.uint8)
4     cv_img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
5     return jsonify(get_bbox(YOLO("best.pt").predict(cv_img))[0])

```

Code 3.3: Server /Detect

After training, the optimized model was deployed as a RESTful service on the server, enabling seamless integration with other systems. The deployment code is provided in Code 3.3 for reference. Additionally, since the Fixed Points algorithm requires bounding

boxes to be defined in clockwise order, conversion code was implemented, as shown in Code 3.4, to transform YOLO's output into the required format.

```

1 def get_bbox(results: Results):
2     return [[{
3         "bbox": list(map(int, obb)), "class": result.names[int(cls)],
4         "confidence": conf} for obb, conf, cls in
5         zip(result.obb.xyxyxyxy, result.obb.conf, result.obb.cls)
6         if conf >= 0.5] for result in results]
```

Code 3.4: Transform Bounding Box

3.1.4 Fixed-points Generation

The fixed-point calculation method integrates the results from color-based object detection and YOLOv11 object detection, systematically storing the vertex and centroid coordinates of stations, along with the geometric details of the bounding box and centroid of the detected objects, in a dictionary format. By leveraging the outputs from these detection methods, rather than directly performing fixed-point calculations on a map containing objects of varying heights (such as stations and obstacles), this approach effectively mitigates errors arising from height differences, as illustrated in Figure 3.5a, Figure 3.5b and Figure 3.5c. Consequently, the resulting fixed-point coordinates exhibit a high degree of accuracy. This optimization not only improves the reliability of coordinate calculations but also lays a robust foundation for subsequent path planning and task execution.

For each detected object, the Euclidean distance between its centroid and the centroids of all stations is calculated to identify the nearest station (Eq. 3.1):

$$\text{distance} = \sqrt{(x_{\text{station}} - x_{\text{object}})^2 + (y_{\text{station}} - y_{\text{object}})^2} \quad (3.1)$$

This allows precise association of the detected object's material with its corresponding station, and this relationship is systematically recorded in the mapping dictionary.

To complete the geometric representation, different colored points are used to highlight key positions. The blue and green points represent the center coordinates derived from geometric calculations, while the red point denotes the midpoint of the station edge where the robot is located during path planning. This red point is determined using vector calculations. From this red point, a fixed vector direction is extended to compute the fixed-point coordinate information. The resulting blue, green, red, and yellow points are visually represented and annotated in the image to validate the calculation results.

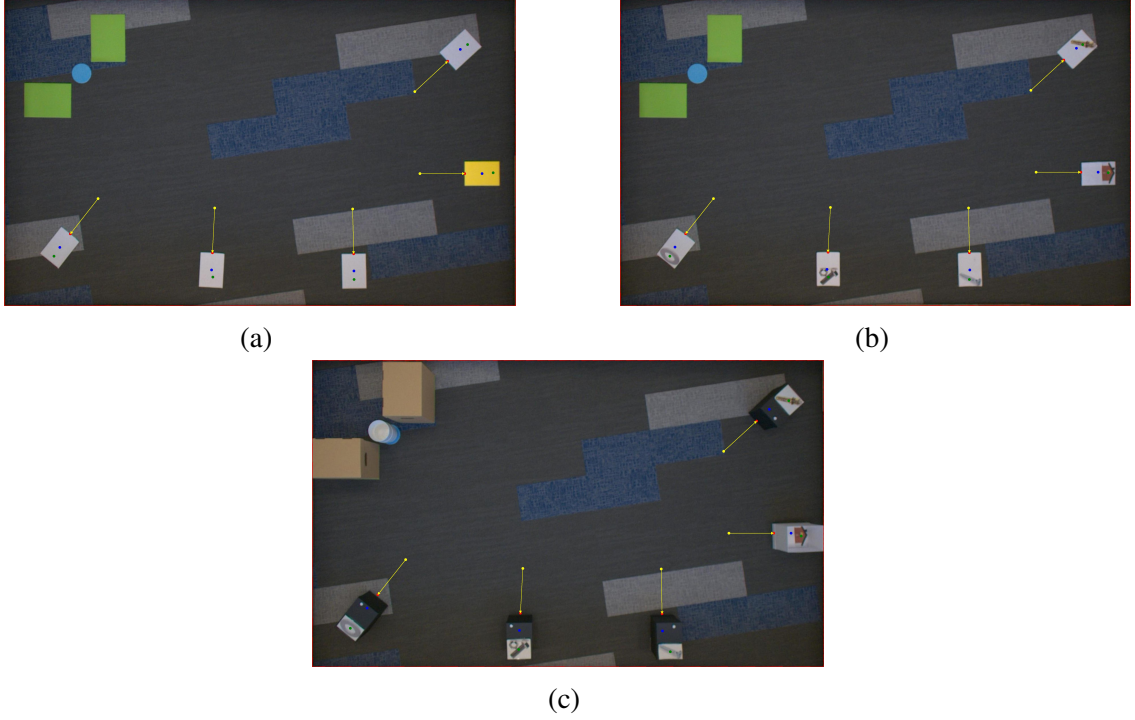


Figure 3.5: Fix-Point with Arrow: (a) Fixed points based on Color-Based Object Detection, (b) Fixed points for navigation tasks that accurately eliminate altitude errors, (c) Fixed point display in a map with different altitudes

Arrows are added to the visualization to intuitively represent orientation information for path planning and Pick-Place tasks performed by the robot at different stations. These arrow directions are calculated and converted into angular values, which are then adjusted to align with the ROS coordinate system.

The final output consists of the fixed-point coordinates and orientation of each station, characterized by three parameters: coordinates (x, y) and orientation θ_z . The flowchart of this process is illustrated in Figure 3.6. The orientation θ_z for each station is determined using the vector dot product method. This data is subsequently transmitted to the server through a /Coordinates POST request for integration into the robot's navigation system.

- **Blue Point:** Represents the center of the station, calculated as the geometric center of the rectangular bounding box obtained through color-based object detection.
- **Green Point:** Represents the center of the detected object, calculated as the geometric center of the bounding box identified by YOLOv11.
- **Red Point:** For each edge of the station, calculate its center and compare the direction vector from the station center to the edge center with the vector from the object

center to the station center. Select the edge whose direction vector has the highest dot product (i.e., the most consistent direction) and mark its center as the red point.

- **Yellow Point (Fix Point):** Using the red point as a reference, extend a vector from the blue point to the red point. Convert the distance measured by the robot in the actual map into pixel values to generate a fixed point. This fixed point indicates the optimal placement and orientation of the object relative to the station.

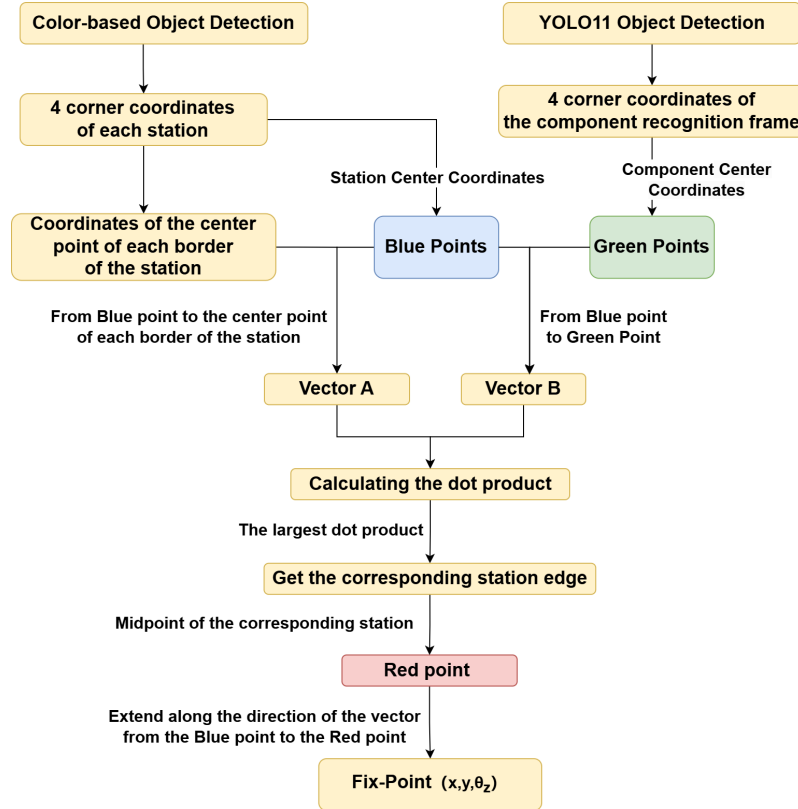


Figure 3.6: Fix point generation Flowchart

3.1.5 Occupancy Map Generation By Using Novel SLAM Method

A novel SLAM method is introduced to generate occupancy grid maps from input images for robotic navigation, with the results saved as .pgm image files and .yaml configuration files compatible with ROS. This innovative approach leverages real-time footage captured by a top-mounted camera, combined with the Color-Based Object Detection techniques detailed in Section 3.1.3, effectively addressing the limitations of conventional LIDAR-based methods.

Traditional LIDAR systems rely on laser reflections to gather environmental data. However, their inability to penetrate obstacles often leads to blind spots and incomplete

mapping of critical areas. Furthermore, LIDAR-generated maps are typically based on point cloud data, providing only contour information, which can be less intuitive for users, especially in complex environments. This lack of visual detail can hinder quick comprehension of the spatial structure.

In contrast, the proposed method employs a top-mounted camera to capture comprehensive visual information of the entire environment. By integrating color detection, it not only accurately determines the occupancy status of target areas but also generates clearer and more detailed maps. These maps provide a richer representation of the environment, making them more user-friendly and effective for robotic path planning.

The reliance on real-time footage also enhances adaptability in dynamic environments. For example, in scenarios involving multiple moving obstacles or rapidly changing conditions, this method maintains high precision in updating maps. By combining improved completeness, real-time performance, and enhanced readability, this vision-based SLAM approach offers a robust solution for material distribution tasks, setting a new benchmark in map generation for robotic navigation.

During implementation, as described in Algorithm 1, the input image is converted to the HSV color space, and contours are detected using predefined color thresholds. Noise is filtered out, and occupied areas are marked with black borders and gray fills, while a black outer border defines the map's boundaries. The map is resized to the target dimensions and centered within a 384×384 gray background. Pixel values are standardized to include only black (occupied), gray (free), and white (unoccupied) regions.

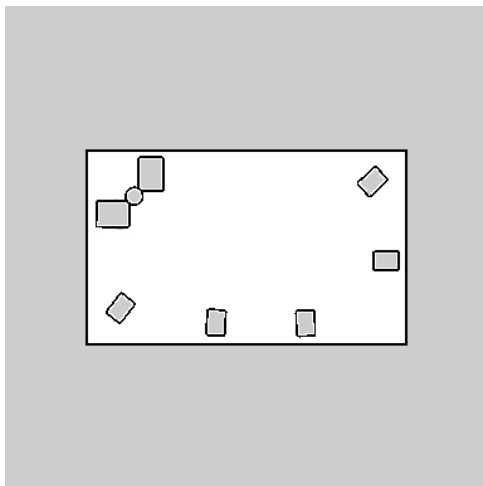
The resulting occupancy grid map is saved as a .pgm file, accompanied by a .yaml configuration file that contains metadata such as file path, resolution, and origin. These files are crucial for robotic navigation tasks, supporting functions like path planning and localization. As illustrated in Fig. 3.7, the upper left corner of the map is blocked by obstacles and all the stations in the map are close to the wall, forming a visual blind spot that the traditional mobile robot's LiDAR cannot map. In contrast, the proposed Novel SLAM method effectively addresses this issue, providing complete map information of the previously blind area. This demonstrates a clear advantage over the traditional LiDAR-based mapping approach in generating more accurate and comprehensive occupancy maps.

Algorithm 1 Occupancy Grid Map Generation

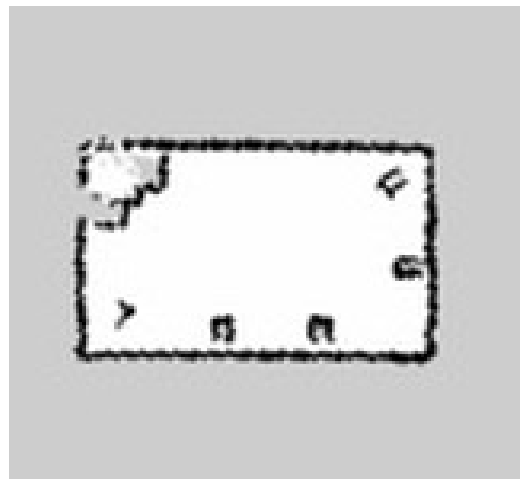
```

1: Input: Image file
2: Convert image to HSV
3: Define color thresholds
4: Initialize white occupancy map
5: for each color and thresholds do
6:     Create binary mask
7:     Find contours and filter by area
8:     if color is blue then
9:         Draw gray-filled circles with black borders
10:    else
11:        Draw gray-filled rectangles with black borders
12:    end if
13: end for
14: Add black border to map
15: Resize to  $156 \times 256$  pixels
16: Center on  $384 \times 384$  gray canvas
17: Quantize to black, gray, white
18: Save as map.pgm
19: Create map.yaml with metadata:
20:     - Image path
21:     - Resolution
22:     - Origin
23:     - Occupied/free thresholds
24: Save YAML file
25: Output: map.pgm, map.yaml

```



(a) Occupancy Map Generated by Novel SLAM Method



(b) Occupancy Map Generated by Traditional SLAM Method

Figure 3.7: Comparison of Occupancy Map

3.2 NLP and LLM Orchestration

As illustrated in Figure 3.8, when a user initiates a request through the Web UI, it is transmitted to the Dify [30] orchestration backend for execution following the workflow depicted in the figure. The system first retrieves the list of items currently present on the map by sending a GET request to the /Stations server endpoint, which is initialized by the coordinate generator shown in Code 3.5. This retrieved item list is then incorporated into LLM1's context for further processing.

```

1 @app.route('/Coordinates', methods=['POST'])
2 def post_coordinates():
3     ...
4     curr_stations.extend({
5         "Station": k, "Position": {"X": v[0], "Y": v[1], "Theta": v[2]}
6     } for k, v in {"main": data.pop("main"), **data}.items())
7     ...

```

Code 3.5: Partial Code of /Coordinates

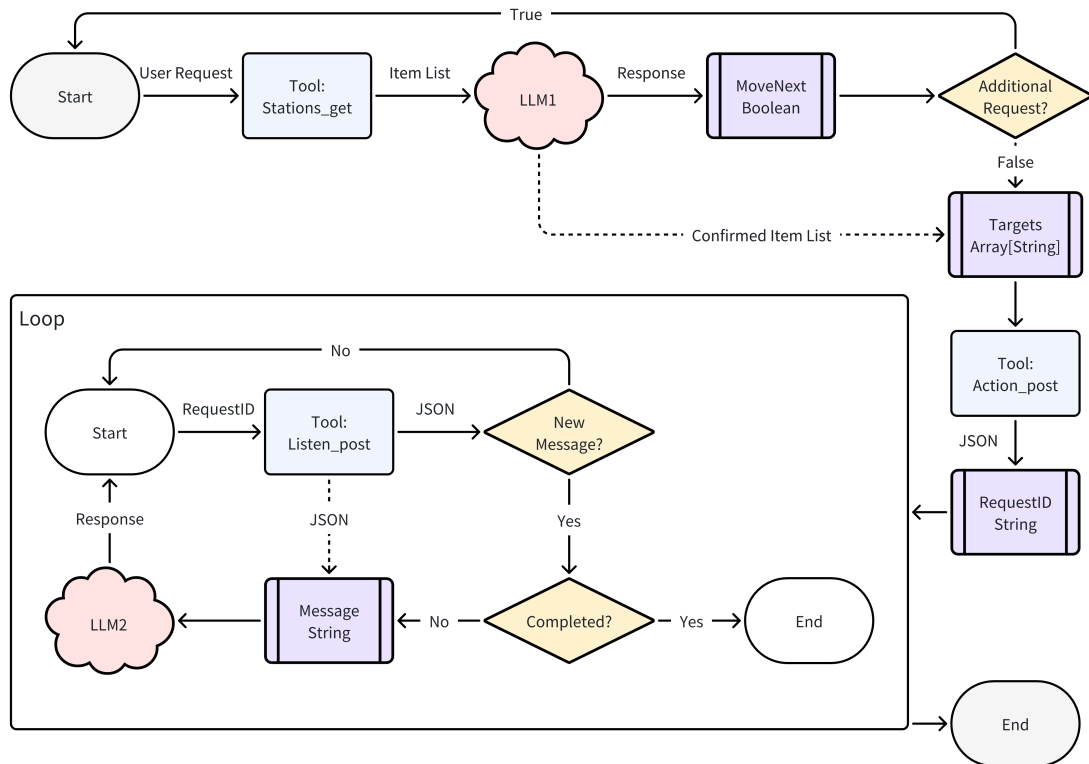


Figure 3.8: NLP Orchestration Flowchart

The LLM1 model is the main channel for user interaction, assuming responsibility

for the collection and confirmation of user requirements. Once the information has been organized, it is forwarded to the subsequent components.

```

1 <instruction>
2     You are a material handling assistant. Your task is to accept user requests
      for items and send them to the next component.
3 </instruction>
4 <stations>
5     {{#context#}}
6 </stations>
7 <instructions>
8     Step 1: If <stations/> is empty:
9         Sorry, no station information is available. Please try later.
10    Step 2: When the user inputs items, check if they exist in the station list.
          The main station cannot be requested.
11        - If all items exist:
12            Okay, the items you need are {list of existing items}. Do you need any
              more items?
13        - If some items do not exist:
14            Sorry, {list of non-existing items} do not exist. The items you need are {
              list of existing items}. Do you need any more items?
15    Step 3: Repeat Step 2 until the user does not want more items.
16        - If no more items are needed:
17            Okay, the final items you need are {final item list}.
18 </instructions>

```

Code 3.6: Partial Prompt of LLM1

As shown in Code 3.6, LLM1 reiterates the user's requirements and performs a secondary confirmation. The subsequent component, consisting of a parameter extractor and a conditional branch, determines whether to proceed with processing the user's request. The parameter extractor generates a Boolean flag, `MoveNext`, to indicate whether the system should initiate an action request to the server or remain idle. This flag is then passed to the discriminator. If user confirmation is not obtained, the system transitions to the start state with memory, awaiting further instructions.

```

1 @app.route('/Action', methods=['POST'])
2 def post_action():
3     ...
4     req = {'op': 'publish', 'topic': '/json_input', 'type': 'std_msgs/String',
5           'msg': {'data': {"RequestID": id, "Stations":
6                     [s for s in curr_stations if s["Station"] in targets]}}}

```

```

7   for ws in connected_clients:
8       try: ws.send(json.dumps(req))
9       ...

```

Code 3.7: Partial Code of /Action

Once the confirmation state is obtained, LLM1's output is passed to the next parameter extractor, which generates an `Array[String]` named `Targets` containing the list of items requested by the user. This array is used to submit a POST request to the `/Action` endpoint shown in Code 3.7, initiating the robot's action. Another parameter extractor then processes the server's response to retrieve the `RequestID` needed for subsequent status queries. At the same time, the system informs the user that the task has been successfully issued, confirming that the operation has been initiated successfully.

```

1  @app.route('/Listen', methods=['POST'])
2  def listen():
3      ...
4      while time.time() - start_time < 30:
5          if pending_messages:
6              msg = pending_messages.pop(0)
7              if "Completed" in msg: finished = True
8              return jsonify({"ROS message": msg})
9      ...

```

Code 3.8: Partial Code of /Listen

The system then enters a loop, using the `RequestID` to send POST requests to the `/Listen` endpoint shown in Code 3.8 with a timeout of 30 seconds. If no new message from ROS is received within this period, the system re-sends the request. During this process, the user can terminate the requests at any time. Upon receiving a message, the system determines whether it is a task completion message. If so, the system outputs the relevant information and ends the conversation. Otherwise, the ROS message content is extracted and converted into a user-friendly format using LLM2, which is then delivered to the user. The system then initiates another `/Listen` request, repeating the process.

3.3 Communications Architecture

As illustrated in Figure 3.9, the communication between NLP and ROS undergoes multiple intermediate layer transformations. This is primarily because ROS and NLP operate within different network environments. In the figure, the HTTP server and NLP are

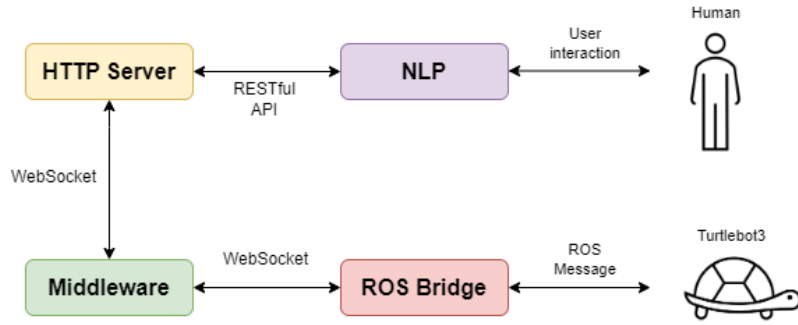


Figure 3.9: Messaging Flow

deployed on the public network, while the other components reside within the internal network. The middleware has access to internal and external networks; therefore, all messages must be forwarded through the middleware, as shown in Code 3.9.

```

1  async with websockets.connect("ws://192.168.0.1:9090") as ws_ros, \
2      websockets.connect("ws://example.com:5000/Bridge") as ws_api:
3      await asyncio.gather(
4          ros_to_api(ws_ros, ws_api), api_to_ros(ws_ros, ws_api))

```

Code 3.9: Partial Code of Middleware

The ROSBridge framework is utilized for the bidirectional translation of ROS messages. As illustrated in Code 3.10, upon receipt of a message from the middleware, the content is encoded into a ROS-compatible format and subsequently forwarded through a publisher. In contrast, ROS state messages are encapsulated in the JSON format via a subscriber and transmitted to the server.

```

1  def json_callback(msg, pub):
2      ...
3      pub.publish(json.dumps({
4          "RequestID": request_id,
5          "Response": "Data received",
6          "Stations": stations,
7          "AdditionalInfo": "Processed successfully"
8      }))
9      ...
10 def json_listener_publisher():
11     pub = rospy.Publisher('/json_output', String, queue_size=10)
12     rospy.Subscriber('/json_input', String, json_callback, pub)

```

Code 3.10: Partial Code of ROSBridge

On the server side, it simply listens to the middleware's WebSocket and temporarily stores incoming messages for use by the /Listen endpoint, as shown in Code 3.11.

```
1 @socket.route('/Bridge')
2 def handle_message(ws):
3     ...
4     while True:
5         pending_messages.append(ws.receive())
6     ...
```

Code 3.11: Partial Code of /Bridge

3.4 Hardwares

3.4.1 TurtleBot3 Waffle Pi with OpenManipulator-X

The experiments were conducted using a TurtleBot3 Waffle Pi, an open-source, compact, and modular robotic platform, equipped with the OpenManipulator-X robotic arm as shown in Fig. 3.10. The TurtleBot3 is powered by a Raspberry Pi 4 for onboard computation, while the OpenManipulator-X utilizes Dynamixel servos for precise manipulation [31]. The robot's modular design facilitates the seamless integration of sensors, actuators, and controllers. ROS Noetic was installed, enabling the system to perform autonomous navigation, object manipulation, and real-time data acquisition in indoor environments [32]. The motion controller was implemented in MATLAB using the Navigation Toolbox and ROS Toolbox [33] [34]. The `controllerVFH` object calculated obstacle-avoiding directions while maintaining forward movement, with the toolboxes supporting motion planning and communication with the robot's ROS 2 system.

3.4.2 Camera: Basler a2A1920-160ucBAS

A Basler a2A1920-160ucBAS camera with a fisheye lens (as shown in Fig. 3.11) was mounted on the top-view of the environment. The camera features a CMOS sensor with a resolution of 1920×1200 pixels and provides high-speed image acquisition at up to 160 frames per second. The camera plays a crucial role in the experimental setup by supporting object detection, recording, and SLAM-based navigation.



Figure 3.10: TurtleBot3 Waffle Pi with OpenManipulator-X



Figure 3.11: Camera Basler [35].

3.4.3 Server

A server equipped with an Intel Xeon Gold 6278C CPU, 32GB RAM, and an NVIDIA T4 is utilized for Dify, Phi3, and YOLO model training and inference, as well as serving as a Flask HTTP server.

Chapter 4

Experiments and results

To evaluate the performance of the proposed system, as it was mention in the section 3.4, experimental trials were conducted using a TurtleBot3 Waffle Pi equipped with an OpenManipulator-X robotic arm. The OpenManipulator-X, enabled precise material handling during pick-and-place tasks. The integrated system seamlessly combined mobility and manipulation, allowing for efficient retrieval of items from multiple stations.

The trials were conducted in an environment designed to emulate real world conditions, incorporating obstacles and varied pathways. Ideal sized dies ($2\text{ cm} \times 2\text{ cm} \times 2\text{ cm}$) were employed as package proxies for fasteners, ensuring compatibility with the robot arm’s grippers. The surface used for the trials had a low coefficient of static friction, optimizing the robot’s mobility.

The primary task required the robot to retrieve components necessary for assembling model H—a Bolt & Nut and Screw—from designated stations. Table 4.1 provides a clear overview of the fasteners needed for different models, facilitating efficient retrieval and assembly planning. Each row in the table represents a specific type of fastener, while each column corresponds to a model. A checkmark (✓) indicates that the fastener is required for the respective model.

Table 4.1: Model Name and Required Fasteners.

Model Name	A	B	C	D	E	F	G	H
Bolt & Nut	✓			✓	✓		✓	✓
Screw		✓				✓	✓	✓
Washer				✓	✓			
Stud			✓		✓	✓	✓	

4.1 Camera Calibration

Figure 4.1 shows the map correction effect in the Model-H experiment. Figure 4.1a is a map captured in real time by the camera above the workspace. Figure 4.1b is a perspective view after fisheye correction. Figure 4.1c first interacts with the user to select the four corners of the required workspace map and automatically surround the red area. Figure 4.1d converts the perspective view to a standard top view.

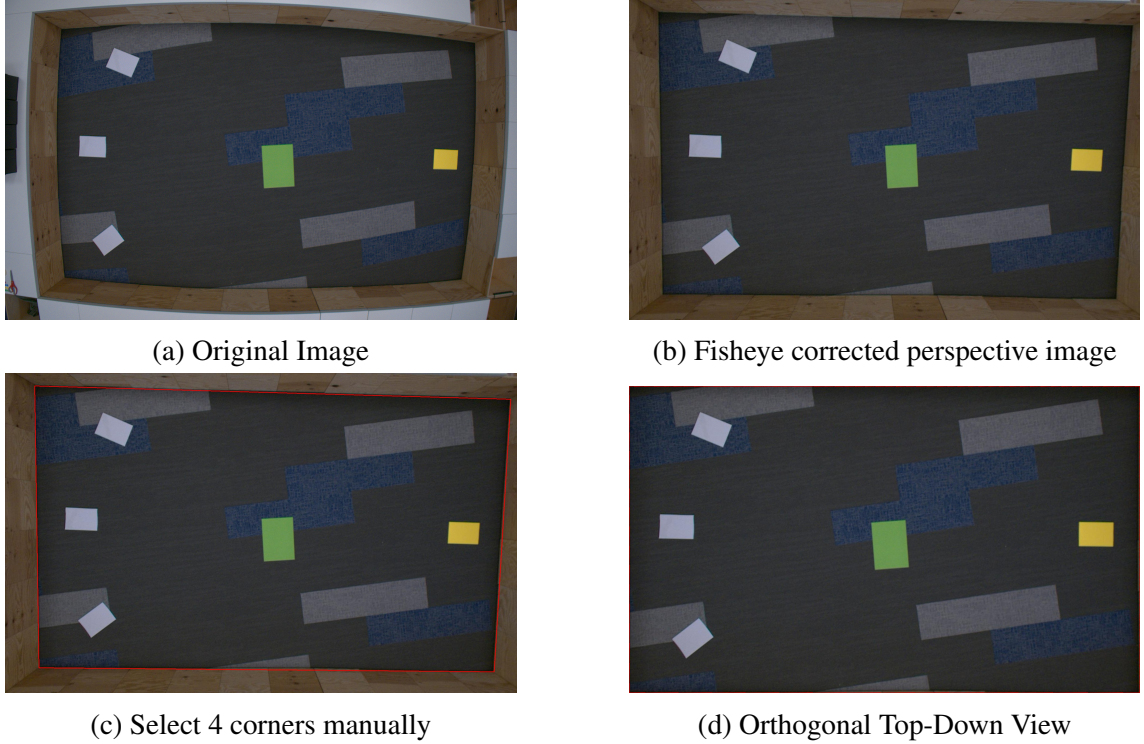


Figure 4.1: Model-H: Comparison of original image and Corrected image

4.2 Object Detection Evaluation

4.2.1 Color-Based ObjectDetection

Figure 4.2 illustrates Model-H, which presents the color-based method for object detection, where the blue, green, and yellow areas represent stations, obstacles, and the main station, respectively.

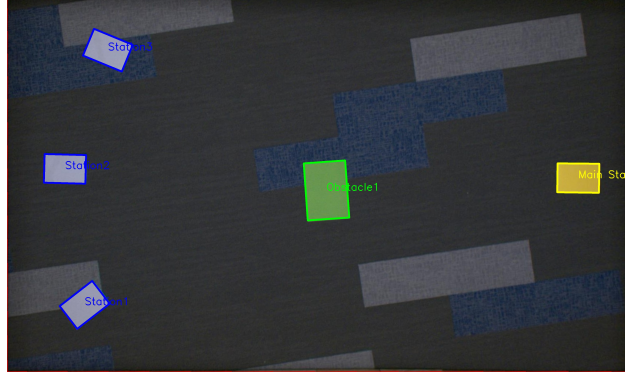


Figure 4.2: Color-Based Detection of Model-H

4.2.2 YOLO11 Model

The model was fine-tuned on an NVIDIA T4 GPU, with the training results shown in Figure 4.3. The model converged rapidly, indicating that the dataset is of high quality and the model size is well-suited for the task. In practical applications, as illustrated in Figure 4.4, the model achieved an average confidence level of 90% for each target item. This demonstrates the robustness of the model and its ability to accurately detect objects in the given domain. Furthermore, the high confidence scores suggest that the model's predictions are reliable, making it suitable for deployment in real-world scenarios.

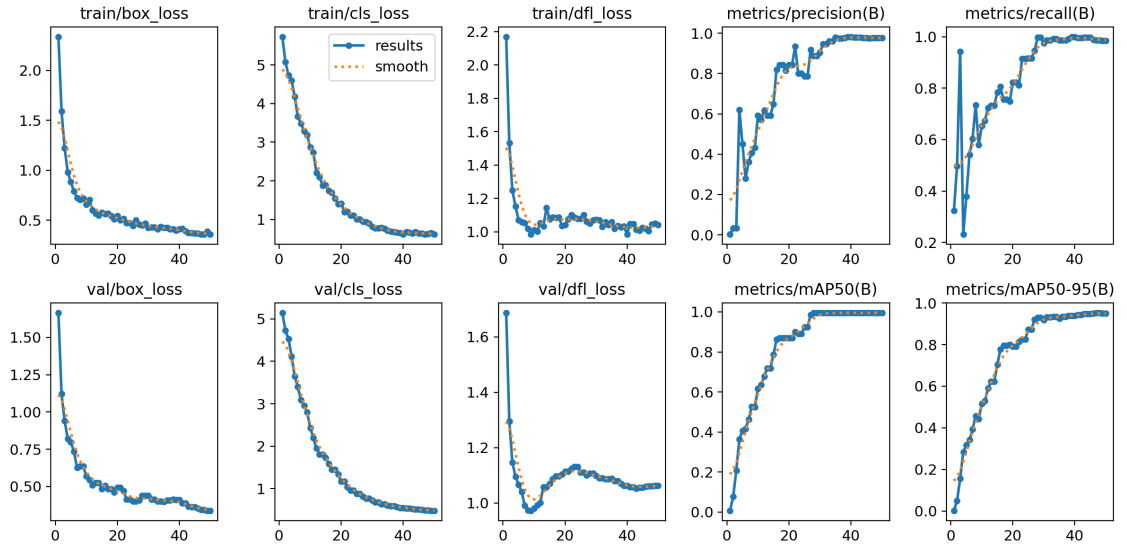


Figure 4.3: YOLO Train Results

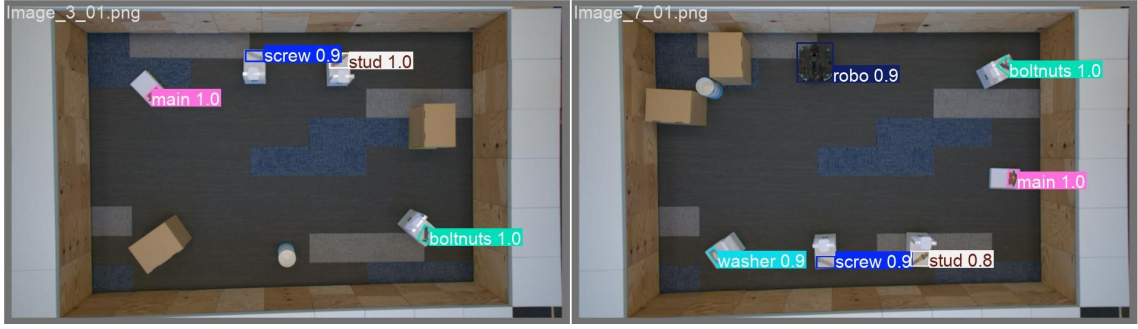


Figure 4.4: YOLO Predict Results

4.2.3 Fixed Points

Figure 4.5 illustrates the fixed-points generation, where green points mark fastener centers, while blue and red points represent station and front-edge centers. Yellow arrows in the figure indicate the position and orientation of each station, guiding the robot to these points for material pick-up or placement. Finally, this generated map is converted into a PGM file for ROS localization, as can be seen in Figure 4.6.

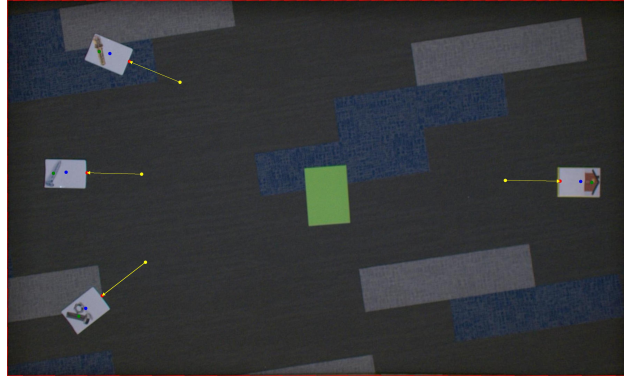
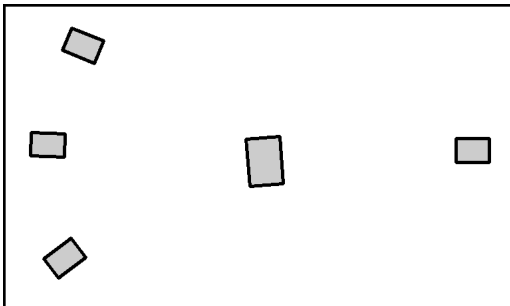
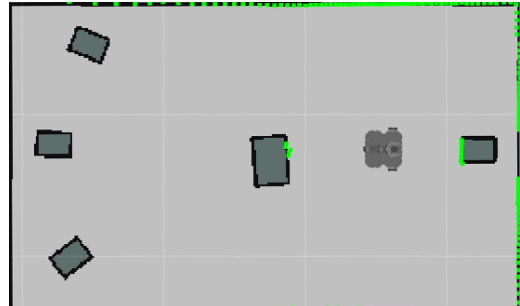


Figure 4.5: Fix point generation with orientation



(a) Occupancy Map by Proposed Method



(b) Robot perception through the LiDAR

Figure 4.6: Localization of the robot in the map of the proposed SLAM method.

4.3 SLAM Evaluation

Occupancy maps were generated for Model-H as shown in Figure 4.6 to facilitate a comparison between the Cartographer SLAM model (A) in ROS2 Foxy and the proposed SLAM model (B). Due to differences in map resolution, a nearest-neighbor interpolation method was applied to standardize the maps. Additionally, Figure 4.7 presents the binary matrices for each model, along with their overlap, to aid in a more comprehensive evaluation.

The performance of both SLAM models was evaluated using two key parameters:

1. *Computational time*: Model (A) took 123 seconds, while Model (B) required 28.34 seconds.
2. *Map Area Coverage*: Model (A) covered 26.18% of the total map area only (including obstacles and free area), while Model (B) the 99.99%.

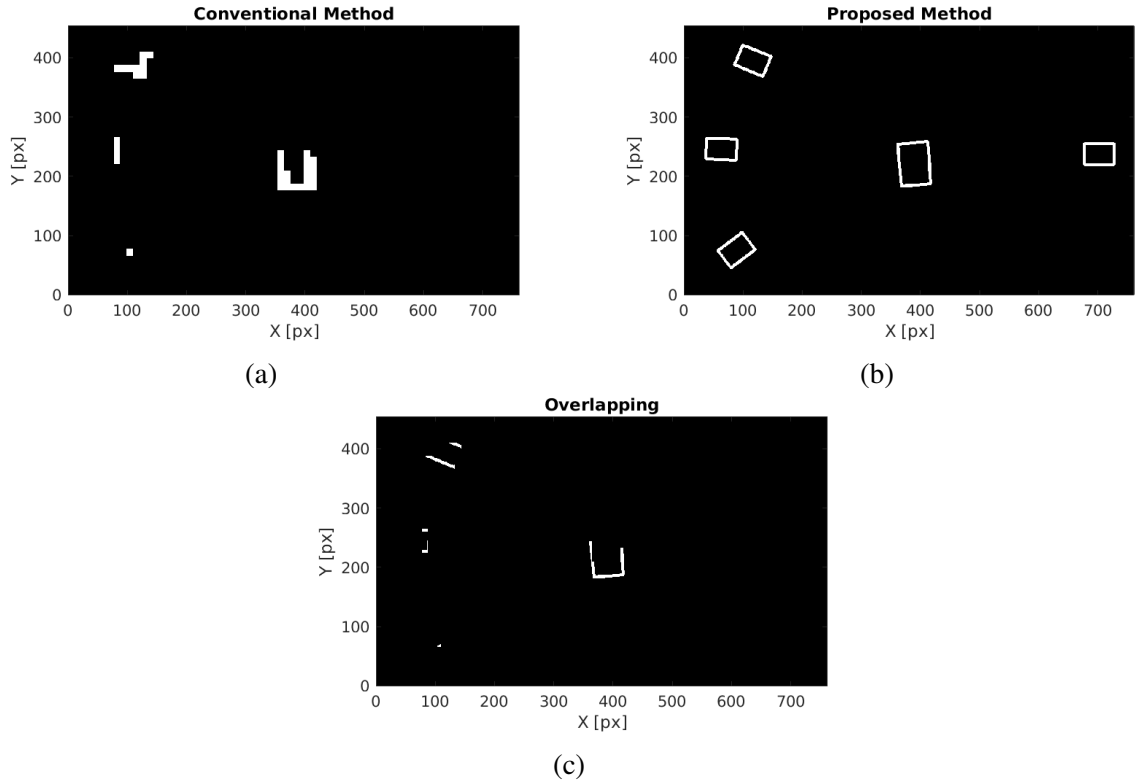


Figure 4.7: Binary Matrices: (a) Cartographer SLAM method, (b) Proposed SLAM method, (c) Overlapping of two methods

4.4 NLP Evaluation

The experiment evaluated the performance of MODEL-D (without a washer on the map) and MODEL-H. As depicted in Figure 4.8, the dialogue outcomes demonstrate the system's ability to handle errors, recover, and maintain operational continuity. This highlights the effectiveness of the system architecture, particularly the middleware, in ensuring real-time and accurate message transmission. Additionally, the LLM accurately interprets user requirements and gathers necessary information, showcasing the efficiency of the NLP workflow.

For the experiments, Meta-Llama-3.1-70B-Instruct [36] was deployed on an NVIDIA A100 GPU, while phi3-14b-medium-128k-instruct [37] ran on an NVIDIA T4 GPU as a comparative setup. The results revealed that both models produced similar outcomes, further validating the robustness and consistency of the system's NLP components.

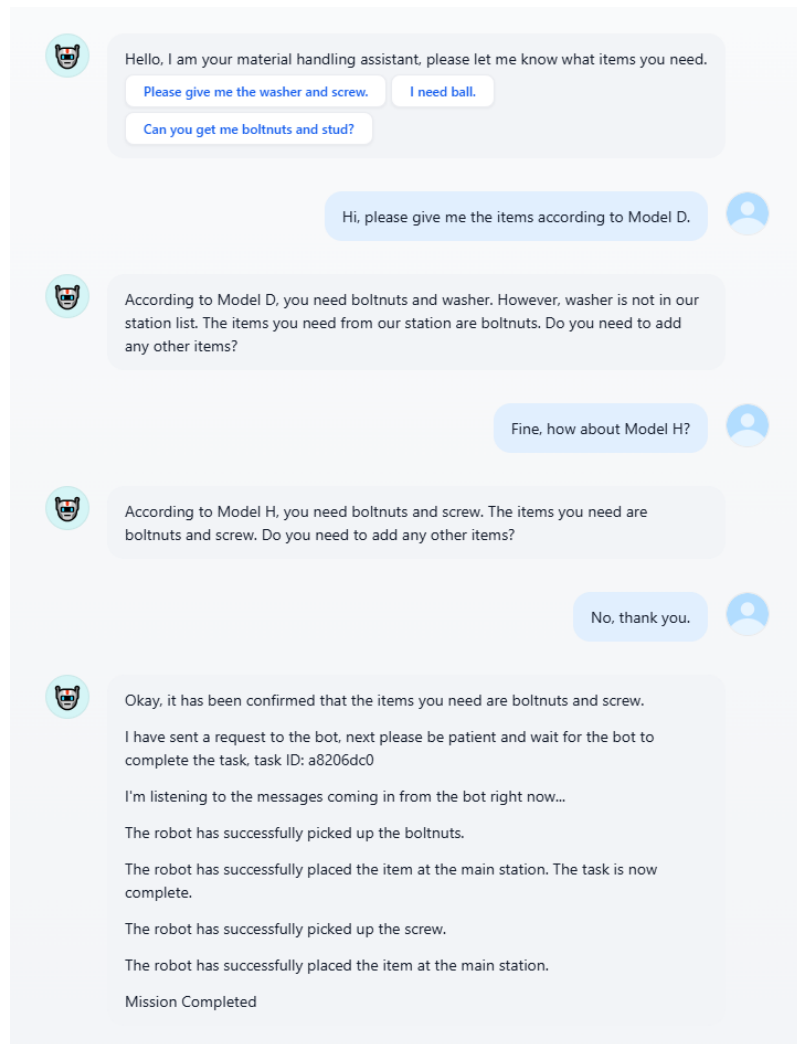


Figure 4.8: Chat History D & H

4.5 Motion Planning Evaluation

The IRRT*-RRMS algorithm guided the robot's path planning. The motion planning for TurtleBot3 and the OpenManipulator-X is implemented in MATLAB programming. This approach provided efficient navigation. Upon receiving a user command specifying model H, the LLM interpreted the request, identified the required materials (bolt, nut, screw), and directed the robot to each station in sequence as follows.

1. From current location to pick-up bolt and nut
2. Place bolt and nut in the main station
3. From main station to pick-up screw
4. Place screw in the main station

The solution paths from each step above are shown in Figure 4.9. The numerical results on path planning are listed in Table. 4.2. The robot can complete the mission to deliver the required bolt, nut, and screw to the user at the main station. The path length for each step is between 2.21 – 2.39 meters. The total computational time is 16.8563 seconds. The total travel length is 9.22 meters. The recording capture is shown in Figure 4.10. The full recording can be viewed using the attached link (Video Link).

Table 4.2: Numerical Result for Model H Case

Step	Pick or Place	Path Length (m)	Computational Time (s)
Current $\Rightarrow$ Bolt&Nut	Pick	2.33	3.8560
Bolt&Nut $\Rightarrow$ Main Station	Place	2.21	5.6085
Main Station $\Rightarrow$ Screw	Pick	2.29	6.5136
Screw $\Rightarrow$ Main Station	Place	2.39	0.8781
Overall	✓	9.22	0.8781

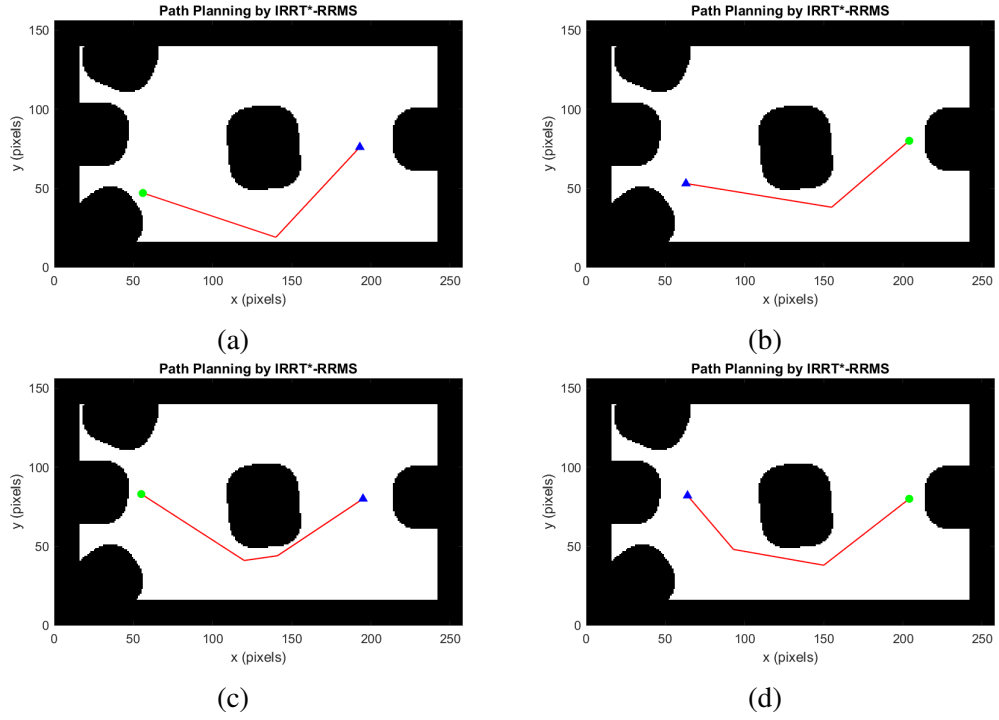


Figure 4.9: Model H: (a) From current position to bolt and nut station, (b) From bolt and nut station to main station, (c) From main station to screw station, (d) From screw station to main station.



Figure 4.10: Recording Captured

Chapter 5

Conclusion

This research has demonstrated the effectiveness of an integrated mobile robotic system for material distribution by employing advancements in Novel SLAM, path planning, and NLP technologies. The results have shown that the proposed system can achieve real-time adaptability, efficient navigation, and intuitive human-robot interaction in industrial environments. The integration of Novel SLAM, IRRT*-RRMS, and NLP enables mobile robots to operate autonomously and interact seamlessly with human operators, thereby addressing key challenges in industrial automation.

5.1 Experimental Validation

The integrated system was validated through extensive trials with TurtleBot3 and the OpenManipulator-X in an environment designed to emulate real-world industrial conditions. The following key observations were made:

- The proposed SLAM approach demonstrated superior area coverage and faster computational times compared to the Cartographer SLAM model. The system achieved a nearly complete map area coverage (99.99%) with reduced computational time (28.34 seconds).
- Path planning experiments indicated that the IRRT*-RRMS algorithm was effective in generating feasible paths in complex environments. The total computational time for model H was 16.86 seconds, with a total travel distance of 9.22 meters across four steps, showcasing efficiency and responsiveness.

- NLP-based evaluations confirmed that the system could effectively interpret user requirements, even when confronted with incomplete or ambiguous instructions. The system demonstrated resilience through error recovery capabilities, thereby ensuring continuity in material distribution tasks.

5.2 Challenges and Future Works

Future developments will focus on further optimizing the system for scalability, real-time adaptability, and improved perception capabilities. There are still several challenges and areas for future improvement:

1. *Real-Time Adaptation in Dynamic Environments:* Although the SLAM method proposed in this work provides significant advancements, further improvements are needed to enhance real-time adaptability to highly dynamic industrial environments where obstacles can change unpredictably.
2. *Scalability for Multi-Robot Systems:* Effective coordination of multiple robots remains an area requiring further research. Future work will focus on enhancing communication protocols and multi-robot coordination algorithms to facilitate collaborative efforts in large-scale industrial operations.
3. *Domain-Specific NLP Models:* Current NLP and LLM models face difficulties adapting to specific industrial terminologies and contexts. Future research will include developing lightweight, domain-specific NLP models to reduce computational requirements and increase system robustness for real-time interaction in edge computing environments.
4. *Multi-Modal Integration for Enhanced Perception:* Integrating visual, LiDAR, and sensor data to achieve a truly multi-modal perception system will be another key area for future work. Improved fusion of sensor data can further enhance object detection and localization accuracy in complex environments.

Bibliography

- [1] Kadir Aram, Gokhan Erdemir, and Burhanettin Can. “Formation Control of Multiple Autonomous Mobile Robots Using Turkish Natural Language Processing”. In: *Applied Sciences* 14.9 (2024). ISSN: 2076-3417. DOI: 10.3390/app14093722. URL: <https://www.mdpi.com/2076-3417/14/9/3722>.
- [2] Pascal Sikorski et al. *Deployment of Large Language Models to Control Mobile Robots at the Edge*. 2024. arXiv: 2405.17670 [cs.R0]. URL: <https://arxiv.org/abs/2405.17670>.
- [3] Jay Lee, Behrad Bagheri, and Hung-An Kao. “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems”. In: *SME Manufacturing Letters* 3 (Dec. 2014). DOI: 10.1016/j.mfglet.2014.12.001.
- [4] S. M. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. Tech. rep. Computer Science Department, Iowa State University, 1998.
- [5] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761. eprint: <https://doi.org/10.1177/0278364911406761>. URL: <https://doi.org/10.1177/0278364911406761>.
- [6] Aphilak Lonklang and Janos Botzheim. “A Rapidly-Exploring Random Tree Algorithm with Reduced Random Map Size”. In: *Proc. of the 9th International Conference on Automation, Robotics and Applications (ICARA)*. 2023, pp. 356–361. DOI: 10.1109/ICARA56516.2023.10125934.
- [7] Andrew Yarovoi and Yong Kwon Cho. “Review of simultaneous localization and mapping (SLAM) for construction robotics applications”. In: *Automation in Construction* 162 (2024), p. 105344. ISSN: 0926-5805. DOI: <https://doi.org/>

- 10.1016/j.autcon.2024.105344. URL: <https://www.sciencedirect.com/science/article/pii/S0926580524000803>.
- [8] Chenggong Zhai et al. “The Application of Robot Technology in the Field of Material Support”. In: *2023 6th International Conference on Robotics, Control and Automation Engineering (RCAE)*. 2023, pp. 275–279. DOI: 10.1109/RCAE59706.2023.10398814.
- [9] Ensong Liu et al. “Centralized Multi-Robot Collaborative LiDAR SLAM Utilizing Loop Closure Selection”. In: *2023 China Automation Congress (CAC)*. 2023, pp. 463–468. DOI: 10.1109/CAC59555.2023.10450881.
- [10] Jinming Yao. “Path Planning Algorithm of Indoor Mobile Robot Based on ROS System”. In: *2023 IEEE International Conference on Image Processing and Computer Applications (ICIPCA)*. 2023, pp. 523–529. DOI: 10.1109/ICIPCA59209.2023.10257966.
- [11] Howie Choset et al. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.
- [12] Ruchir Patel et al. “Decentralized Task Allocation in Multi-Agent Systems Using a Decentralized Genetic Algorithm”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 3770–3776. DOI: 10.1109/ICRA40945.2020.9197314.
- [13] H. Durrant-Whyte and T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics & Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.
- [14] Guillaume Bresson et al. “Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving”. In: *IEEE Transactions on Intelligent Vehicles* 2.3 (2017), pp. 194–220. DOI: 10.1109/TIV.2017.2749181.
- [15] Saad Mokssit et al. “Deep Learning Techniques for Visual SLAM: A Survey”. In: *IEEE Access* 11 (2023), pp. 20026–20050. DOI: 10.1109/ACCESS.2023.3249661.
- [16] Youssef Al-Ozaibi, Georges Chahine, and Oussama El-Hamzaoui. “Monocular ORB-SLAM and Pose Retrieval for Autonomous Driving”. In: *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big*

- Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. 2023, pp. 950–955. DOI: 10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00137.
- [17] Abdelhak Bougouffa et al. “Evaluation of a novel DSO-based Indoor Ceiling-Vision Odometry System”. In: *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2022, pp. 47–53. DOI: 10.1109/ICARCV57592.2022.10004272.
- [18] Ying Li. “Researches advanced in the SLAM for dynamic environment”. In: *Journal of Physics: Conference Series* 2634 (Nov. 2023), p. 012054. DOI: 10.1088/1742-6596/2634/1/012054.
- [19] Aphilak Lonklang and János Botzheim. “Improved Rapidly Exploring Random Tree with Bacterial Mutation and Node Deletion for Offline Path Planning of Mobile Robot”. In: *Electronics* 11.9 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11091459. URL: <https://www.mdpi.com/2079-9292/11/9/1459>.
- [20] Aphilak Lonklang and János Botzheim. “Mobile Robot Path Planning for Unknown Static Obstacle Avoidance by Improved RRT* Algorithm”. In: *2024 10th International Conference on Automation, Robotics and Applications (ICARA)*. 2024, pp. 155–159. DOI: 10.1109/ICARA60736.2024.10553042.
- [21] Ferenc Adam Szili, János Botzheim, and Balázs Nagy. “Bacterial Evolutionary Algorithm-Trained Interpolative Fuzzy System for Mobile Robot Navigation”. In: *Electronics* 11.11 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11111734. URL: <https://www.mdpi.com/2079-9292/11/11/1734>.
- [22] Chao Wang et al. “LaMI: Large Language Models for Multi-Modal Human-Robot Interaction”. In: *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. CHI EA ’24. Honolulu, HI, USA: Association for Computing Machinery, 2024. ISBN: 9798400703317. DOI: 10.1145/3613905.3651029. URL: <https://doi.org/10.1145/3613905.3651029>.
- [23] Rui Liu et al. “A Review of Natural-Language-Instructed Robot Execution Systems”. In: *AI* 5.3 (2024), pp. 948–989. ISSN: 2673-2688. DOI: 10.3390/ai5030048. URL: <https://www.mdpi.com/2673-2688/5/3/48>.

- [24] S. V. Tresa Sangeetha et al. “AI Integrated Mobile Robot for Hospital Reception and Registration Services with Environment Mapping and Autonomous Navigation”. In: *2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. 2024, pp. 1073–1079. DOI: 10.1109/ICACCS60874.2024.10717171.
- [25] Hamid Taheri and Zhao Chun Xia. “SLAM; definition and evolution”. In: *Engineering Applications of Artificial Intelligence* 97 (2021), p. 104032. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2020.104032>. URL: <https://www.sciencedirect.com/science/article/pii/S0952197620303092>.
- [26] Muhammad Diginsa, Noraimi Shafie, and Nazir Yusuf. “Review: Issues and Challenges of Simultaneous Localization and Mapping (SLAM) Technology in Autonomous Robot”. In: *International Journal of Innovative Computing* 13 (Nov. 2023), pp. 59–63. DOI: 10.11113/ijic.v13n2.408.
- [27] Ellen Schwalbe. “Geometric modelling and calibration of fisheye lens camera systems”. In: *Proc. 2nd Panoramic Photogrammetry Workshop, Int. Archives of Photogrammetry and Remote Sensing*. Vol. 36. Part 5. Citeseer. 2005, W8.
- [28] Rahima Khanam and Muhammad Hussain. *YOLOv11: An Overview of the Key Architectural Enhancements*. 2024. arXiv: 2410.17725 [cs.CV]. URL: <https://arxiv.org/abs/2410.17725>.
- [29] Boris Sekachev et al. *opencv/cvat: v1.1.0*. Version v1.1.0. Aug. 2020. DOI: 10.5281/zenodo.4009388. URL: <https://doi.org/10.5281/zenodo.4009388>.
- [30] Dify.AI. *Dify: Open-Source LLM Application Development Platform*. Version 0.10.0. Accessed: 2024-11-28. 2024. URL: <https://github.com/langgenius/dify>.
- [31] ROBOTIS. *TurtleBot3 e-Manual*. Accessed: 2024-11-29. 2021. URL: <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>.
- [32] ROBOTIS. *OpenManipulator-X e-Manual*. Accessed: 2024-11-29. 2021. URL: https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/.
- [33] Inc. The MathWorks. *Navigation Toolbox*. Available at: <https://www.mathworks.com/products/navigation.html>. 2024.

- [34] Inc. The MathWorks. *ROS Toolbox*. Available at: <https://www.mathworks.com/products/ros.html>. 2024.
- [35] Basler AG. *Basler Camera a2A1920-160ucBAS Documentation*. Accessed: 2024-11-29. 2023. URL: <https://docs.baslerweb.com/a2a1920-160ucbas>.
- [36] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407 . 21783 [cs.AI]. URL: <https://arxiv.org/abs/2407.21783>.
- [37] Marah Abdin et al. *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. 2024. arXiv: 2404 . 14219 [cs.CL]. URL: <https://arxiv.org/abs/2404.14219>.

List of Figures

2.1	IRRT*-RRMS for Path Planning	8
3.1	System Overview	11
3.2	Fisheye Calibration Chessboard Patterns	11
3.3	Comparison of point selected image and cropped image	13
3.4	Comparison of Original Image and Color-Based Detection	14
3.5	Fix-Point with Arrow: (a) Fixed points based on Color-Based Object Detection,(b) Fixed points for navigation tasks that accurately eliminate altitude errors, (c) Fixed point display in a map with different altitudes . .	16
3.6	Fix point generation Flowchart	17
3.7	Comparison of Occupancy Map	19
3.8	NLP Orchestration Flowchart	20
3.9	Messaging Flow	23
3.10	TurtleBot3 Waffle Pi with OpenManipulator-X	25
3.11	Camera Basler [35].	25
4.1	Model-H: Comparison of original image and Corrected image	27
4.2	Color-Based Detection of Model-H	28
4.3	YOLO Train Results	28
4.4	YOLO Predict Results	29
4.5	Fix point generation with orientation	29
4.6	Localization of the robot in the map of the proposed SLAM method. . . .	29
4.7	Binary Matrices: (a) Cartographer SLAM method, (b) Proposed SLAM method, (c) Overlapping of two methods	30
4.8	Chat History D & H	31
4.9	Model H: (a) From current position to bolt and nut station, (b) From bolt and nut station to main station, (c) From main station to screw station, (d) From screw station to main station.	33

4.10 Recording Captured	33
-----------------------------------	----

List of Tables

4.1	Model Name and Required Fasteners.	26
4.2	Numerical Result for Model H Case	32

List of Algorithms

1 Occupancy Grid Map Generation 19

List of Codes

3.1	Partial Code of Undistort	12
3.2	YOLO Simple Train	14
3.3	Server /Detect	14
3.4	Transform Bounding Box	15
3.5	Partial Code of /Coordinates	20
3.6	Partial Prompt of LLM1	21
3.7	Partial Code of /Action	21
3.8	Partial Code of /Listen	22
3.9	Partial Code of Middleware	23
3.10	Partial Code of ROSBridge	23
3.11	Partial Code of /Bridge	24