Sharif University of Technology
Electrical Engineering Department
,

July, 2025

Autonomous Robotic Agent

Matin M.babaei
Pantea Amoei
Ali Ekhterachian
Amin Mirzaei

A Bachelor's Thesis
Bachelor of Science in Elecrical Engineering

Professor:
Prof. Fakharzadeh
Prof. Khalaj

To the Sharif University of Technology,
for shaping our academic foundation.

Contents

List of Figures

List of Tables

Acknowledgments

We would like to express our sincere gratitude to Professor Khalaj for his valuable guidance and academic support throughout this project.

We are also thankful to Mr. Binesh for his technical supervision and continuous assistance during the development and implementation phases.

Chapter 1

Introduction

Robotic Agent is one of the hot spots of the AI industry due to its applications in different domains and Many experts anticipate that they are going to play a crucial role in these domains in the next few years. In the medical domain, robots are utilized as surgeon assistants for their precision, while in transportation, they are used as autonomous vehicles. However, our focus is on their indoor application in houses and offices. In this report, we are going to explain our Autonomous Robotic Agent framework developed for indoor tasks.
Our framework consists of three different parts, **3D Mapping**, **Environment Exploration** and **Object Retrieval**:
The **3D Mapping** part is the task of obtaining 3D information from an unknown environment and storing this information. There are many different ways to acquire 3D information. Hardwares like LiDAR [5] enables us to perform precise 3D mapping. However, these devices are very expensive. But there are some visual methods which utilize cameras and RGB images. To avoid additional expenses, we have decided to exploit another setup, utilizing a stereo camera combined with a feature matching algorithm. The stereo camera enables robot to view the scene from two different views and the feature matching model identifies same features across two images. By measuring the horizontal shift of the features in images and using the camera parameters, we can easily anticipate the 3D position of the point relative to the robot. The point coordinates are relative to the robot's position and view angle and should be transformed to the scene 3D coordinates.

The **Environment Exploration** part is the task of observing the environment from different views to collect the 3D information from all parts of the environment. The locations should be chosen efficiently to reduce the time and the cost of 3D reconstruction. Hence, we utilized a frontier based exploration model to obtain the suggested locations to gather 3D information. This model identify the points that connect the known and unknown part of the environment which are very informative. We also leverage the RRT* [3] model to obtain a path which avoids obstacles and guides the robot from its current location to the target location.
The **Object Retrieval** part is the task of searching for a specific object defined in a few words by the user. The agent should be able to locate the object in the environment. Our framework utilizes some multi-modal deep learning models like CLIP[4] and Grounding-DINO[5] to search for the object in 2D images taken from the environment and exploits the pixels position in the 3D environment to extract the location of the object.
Finally, we have evaluated our framework in a simulation environment and also in a real laboratory space. Our simulation environment is Webots [3] which is a very powerful environment for robotic simulation.

Chapter 2

Related Work

As we discussed earlier, our framework consists of three different parts, 3D Mapping, Exploration and Object-retrieval. Now we want investigate some of the previous methods for these tasks:

2.1 3D Mapping

In this part, we review some recent methods for 3D Mapping by wxploring an unknown environment related to our Autonomous Robotic Agent framework. 3D Maps can be obtained by some hardwares like LiDARs [5]. But these devices are very expensive which makes them an unsuitable choice for indoor tasks. Hence, we have decided to exploit cameras instead of expensive devices and leverage visual depth estimation methods. Monocular depth estimation methods like DepthAnything [6] are some methods which try to estimate depth map of a scene from an specific view from a single RGB image. These methods rely on the perspective phenomena. However, these methods can not accurately acquire depth maps because they just estimate the real size of the object and measure its depth based on its size on the camera image.
Another solution is to use multi-view depth estimation and leverage the fact that the 3D position of objects doesn't change for each view of the camera. In some cases, the model doesn't know the exact position and the heading of the camera in each location. These models try to estimate the camera position and heading. VGGT [7] is a model developed for 3D reconstruction of a scene from 2D RGB images from many different views. Because this model processes many images simultaneously, it requires high capacity GPU to perform its real-time task. While this high capacity GPUs are also expensive and inappropriate for our indoor tasks. There is also another method called MAST3R [8] which process each pair of images together for 3D reconstruction to reduce the computation.
Also a suitable approach is to exploit a stereo cameras and feature matching methods for depth estimation. Stereo camera is a setup of two aligned camera with a specific distance and same headings to capture a scene from two different views. In this case, the relative position of the cameras are fixed and known. There are lots of different methods for feature matching task like SuperGlue [9], Roma [1] and DUSt3R [10] which is a new architecture driven from MAST3R.
There are also some frameworks which utilizes feature matching models not only for depth estimation, but also for pose estimation of the robot to complete their 3D map. MASt3R-SLAM [11] proposes a novel real-time dense SLAM system that leverages deep 3D reconstruction priors. Unlike traditional SLAM methods that rely on sparse features, MASt3R-SLAM uses a learned two-view 3D prior (MASt3R) to directly predict dense geometry.

2.2 Exploration

In this part, we want to investigate some exploration and navigation algorithms related to our project.

2.2.1 Frontier-Based Exploration for Autonomous Robot

Yamauchi [2] proposed frontier-based exploration as a simple yet effective method for autonomous mapping. In this approach, the robot maintains an occupancy grid and identifies *frontiers*, which are cells on the boundary between known free space and unexplored areas. By selecting and moving toward these frontiers, the robot incrementally expands its knowledge of the environment. This strategy guarantees

complete coverage of all reachable space while requiring minimal computation, as it relies only on local sensing and map updates without prior knowledge of the environment.

2.2.2 A* Path Planning Algorithm

The A* algorithm [12] is one of the most widely used graph-based search methods for robot navigation. It extends Dijkstra's algorithm by introducing a heuristic function to guide the search toward the goal more efficiently. Given a cost function $g(n)$ representing the path cost from the start to a node n, and a heuristic $h(n)$ estimating the cost from n to the goal, A* minimizes the total function $f(n) = g(n)+h(n)$. When the heuristic is admissible (never overestimates the true cost), A* guarantees finding the optimal path.

While A* is effective for grid-based maps or static environments, it requires a full known representation of the environment and can become computationally expensive in large or high-dimensional spaces. Its reliance on discretized maps makes it less suitable for online planning in partially unknown environments.

2.2.3 RRT and RRT* Algorithms

For more complex and partially known environments, sampling-based planners such as Rapidly-exploring Random Tree (RRT) [13] have been introduced. RRT incrementally grows a tree by randomly sampling points in the space and connecting them to the nearest existing node, efficiently exploring high-dimensional configuration spaces without requiring an explicit grid map. However, the basic RRT algorithm does not optimize path quality and may produce unnecessarily long paths.

To address this, Karaman and Frazzoli [14] proposed RRT*, an improved variant that includes a *rewiring* step to iteratively optimize paths as new samples are added. RRT* is asymptotically optimal, meaning it converges to the shortest possible path with enough samples. Compared to A*, RRT and RRT* are better suited for real-time planning in continuous and partially known environments, as they do not require discretization and can efficiently handle dynamic changes in the map.

2.3 Object Retrieval

In this section, we review some recent methods relevant to the object retrieval component of our Autonomous Robotic Agent framework. The main goal of object retrieval systems is to detect and localize objects based on user-defined queries, involving textual information.

2.3.1 Point Cloud Encoder-based Approaches

Recently, several methods have emerged that encode raw 3D point clouds into structured representations, enabling fine-grained reasoning from 3D geometry. One prominent example is Scene-LLM [15]. Scene-LLM integrates egocentric and scene-level point cloud features into Large Language Models (LLMs) for interactive planning. It achieves this by combining visual inputs and language context, resulting in improved scene understanding and the ability to reason about spatial relationships. However, Scene-LLM relies heavily on high-quality point cloud data, which may limit its applicability in scenarios with limited or noisy sensor inputs.

2.3.2 Multi-modal Fusion and Graph-based Methods

Another significant approach in the field involves multi-modal fusion and graph-based reasoning. These methods fuse visual, spatial, and language features into hierarchical structures, supporting complex tasks like long-horizon navigation and semantic reasoning. One representative method is the Hierarchical Open-Vocabulary Scene Graph (HOV-SG) [16]. HOV-SG constructs hierarchical 3D scene graphs using open-vocabulary embeddings provided by models like CLIP. This allows robots to navigate environments and perform object retrieval tasks based on natural language queries effectively. Despite its robust reasoning capabilities, HOV-SG may face challenges when dealing with dynamic environments, as maintaining and updating complex graph structures in real-time can become computationally expensive.

Chapter 3

Methodology

As we discussed earlier, our frame work consists of three different parts, **3D Mapping**, **Environment Exploration** and **Object Retrieval**. The **Environment Exploration** also consists of **Location selection** and **Navigation** and all the parts are elaborated here:

3.1 3D Mapping

As we previously explained, there are many different ways for 3D mapping like utilizing LiDAR [5] and multi-view deep learning models. However, to avoid additional costs and GPU capacity which was unnecessary for our indoor robotic tasks, finally we decided to use Stereo vision combined with feature matching algorithm for our purpose.

3.1.1 Stereo vision

Stereo vision enables the robot to observe the scene from two different views with a known distance between two cameras. The projection location of a specific 3D point on one of the camera planes is shifted horizontally related to the other camera by an amount dependent on the perpendicular distance between the point and camera planes. The Depth of the point from the camera plane can be easily calculated from this information and the camera parameters. The two other coordinates can also be calculated by knowing the x and y pixel location on image and the depth of the point.

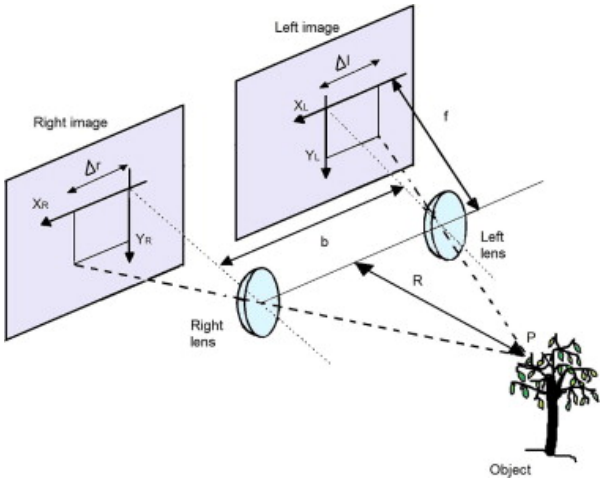

Figure 3.1: Stereo vision

Figure 3.2: An example of robot's stereo image

3.1.2 Feature matching

The Feature matching is defined as to identify similar features across two different images taken from same scenes. We have used the Roma model [1] for our framework. It's a very precise deep learning model used for the feature matching task.

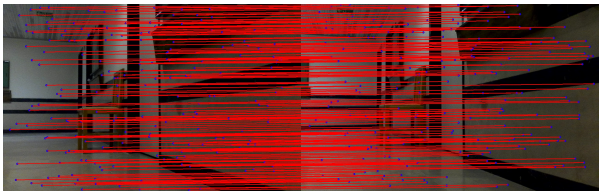

Figure 3.3: An example of feature matching performed by Roma model [1]

3.1.3 3D points' coordinate

The depth of each point is determined by this formula:

$$d = \frac{b * f_l}{disparity}$$

Here the disparity is the shift of the feature in pixels and b and f_l are the baseline and focal length of the stereo camera. The x and y coordinates are also caculated by this formulas:

$$x = \frac{(x_p - c_x)}{f_l} * d$$

$$y = \frac{(y_p - c_y)}{f_l} * d$$

Here, x_p and y_p are pixel coordinates on camera image and the c_x and c_y the coordinates image's center.
As we discussed earlier, these coordinates are relative to the camera position and view's direction. Now we have to transform these coordinates to the real world coordinates. The formulas are demonstrated here:.

$$\mathbf{P}_{\text{base}} = \mathbf{T}_{\text{camera to base}} \cdot \mathbf{P}_{\text{camera}}$$

$$\begin{pmatrix} x_{\text{base}} \\ y_{\text{base}} \\ z_{\text{base}} \\ 1 \end{pmatrix} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{\text{camera}} \\ y_{\text{camera}} \\ z_{\text{camera}} \\ 1 \end{pmatrix}$$

Here the R is the camera 3*3 rotation matrix and the t is the robot's translation vector.

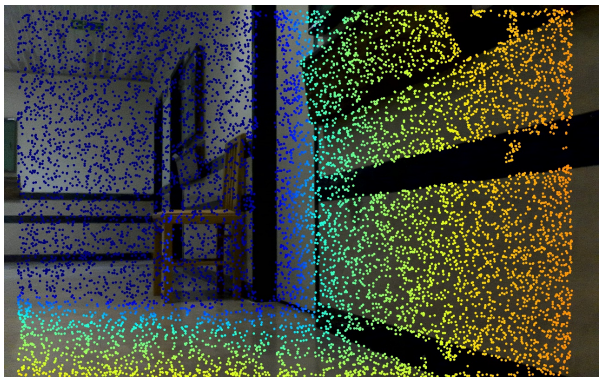

Figure 3.4: An example of depth color map obtained by our method(Closer points are demonstrated by brighter colors and further points are demonstrated by darker colors.)

3.1.4 Localization

Accurate localization is essential for 3D mapping, as the robot must know its position and heading to update the occupancy grid. Since no external positioning system (like GPS) is available indoors, our system relies on odometry from wheel encoders, with vision-based corrections to reduce drift.
Wheel encoders provide continuous estimates of the robot's motion by measuring wheel rotations and integrating them over time to estimate the pose. While wheel encoders are effective for short-term localization, odometry suffers from drift due to slippage and small measurement errors, and integrates this drift over time. Although an IMU could further stabilize orientation estimation, it is not yet integrated and remains a future enhancement.
To improve long-term accuracy, we apply vision-based localization. Feature matching algorithm can also be performed across Consecutive frames. This enables robot to estimate the robot's relative motion by measuring the difference between relative coordinates of similar features. These visual corrections help counteract odometry drift. By fusing odometry for high-frequency updates with vision-based corrections for drift compensation, the robot maintains a stable and consistent pose estimate over time. This hybrid approach allows reliable navigation and mapping during exploration without requiring external

3.1.5 3D Maps

The environment is discretized into an occupancy grid map, where each cell has three possible states: (Figure 3.5)

- **Free** – traversable space
- **Occupied** – obstacle
- **Unknown** – unexplored area

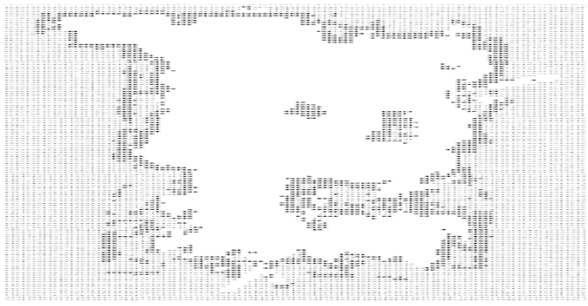

Figure 3.5: Occupency map [2]

The occupied points are the points that we have 3D coordinates for them. The free space is all the points between the camera and the obstacles, because if they were not free space, the camera was unable to observe the obstacles. Finally, all other points are considered as the Unknown points.
At each location where the robot stops, it performs a 240-degrees scan. It stops each 20 degrees and takes an image. An example of the map is depicted in (Figure 3.6). The 3D points are projected on 2D plane for better visualization.

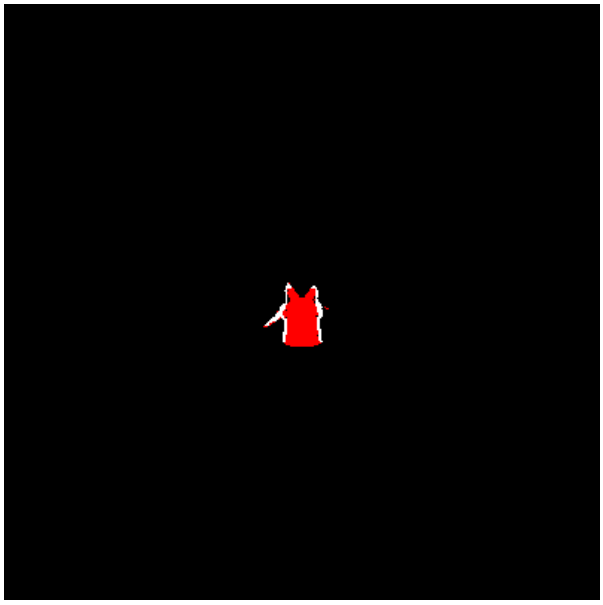

Figure 3.6: Robot's 240-degrees scan(White points corresponds to obstacles while red and black points represent free and unknown points.)

The robot also moves to other locations to complete its map. Here we have demostrated some examples(Figure 3.7):

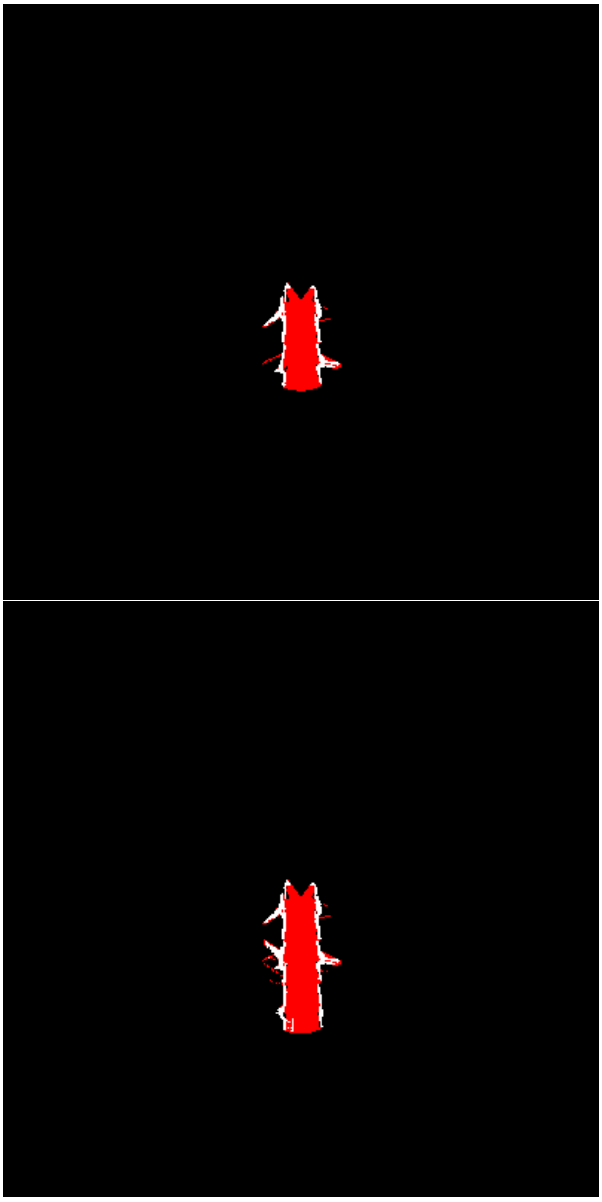

Figure 3.7: Robot's map completion (White points corresponds to obstacles while red and black points represent free and unknown points.)

3.2 Exploration

Efficient exploration of an unknown environment requires strategic decisions to minimize time, energy, and computational effort while maximizing new information gained. In autonomous robotic systems, this decision-making occurs iteratively as the robot selects the next location to explore.

3.2.1 Location Selection

Choosing exploration targets directly impacts how effectively the robot expands its knowledge without unnecessary travel. Poor location selection can lead to:

- Longer travel times with minimal new information gained,
- Higher energy consumption, critical for power-constrained robots,
- Redundant mapping of already known areas.

To avoid these issues, we prioritize **frontier regions**—boundaries between known free space and unexplored areas. This ensures every movement contributes to discovering new space. For safety, the robot only traverses confirmed free-space, avoiding unknown or obstructed areas while advancing exploration toward unknown boundaries.

Definition of Frontier Regions

A frontier is defined as the boundary between known free space and unknown cells in the environment. Formally:

- A *free cell* is a grid cell that has been explored and confirmed to be traversable.
- An *unknown cell* is a grid cell that has not yet been observed by the robot's sensors.
- A *frontier cell* is an unknown cell that is directly adjacent to at least one free cell.

With this definition, any frontier cell is placed at the boundary of known space. Thus, it is the best candidate for exploration since visiting the frontier cell would result in expanding the map in the process.

Algorithm for Frontier-based Exploration

The frontier-based exploration algorithm systematically identifies, selects, and navigates toward frontiers. The algorithm will be explained in these steps.

1. **Frontier Detection**
   The algorithm scans all unknown cells in the occupancy grid. For each unknown cell, it checks the 4 or 8 neighboring cells. If at least one neighboring cell is free, then the unknown cell is marked as a frontier cell.

This process gives us a frontier map, which reveals all the regions where exploration can occur. (Figures 3.8, 3.9, and 3.10)

Figure 3.8: Webots environment [3]

Figure 3.9: An example of frontier results in Webots environment

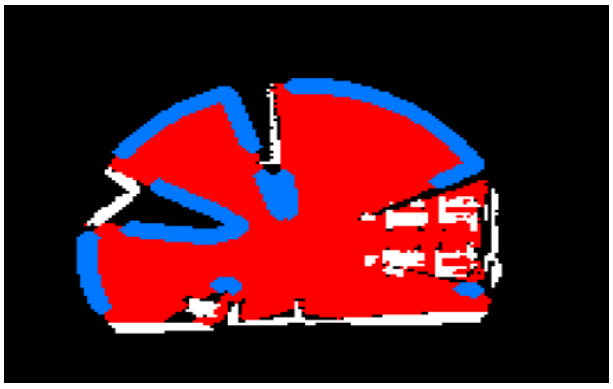

Figure 3.10: Another example of frontier results in Webots environment

2. **Clustering Frontiers using DBSCAN**
   Individual frontier cells are often noisy or too small to be meaningful exploration targets. To group them into larger, useful regions, we apply the DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise [17]).

   DBSCAN clusters densely packed frontier cells based on a distance threshold *eps* and a minimum number of neighboring points *minPts*, while treating isolated points as noise. This reduces thousands of scattered frontier cells into a few significant frontier clusters for easier selection.

3. **Scoring Frontier Clusters**
   After clustering, each frontier cluster is evaluated and assigned a heuristic score that balances two competing factors:

   - *Information Gain Potential* – how much new area could potentially be explored by visiting this cluster, which can be approximated by the length of the frontier.
   - *Traversal Cost* – how far the robot would need to travel to reach this cluster from its current location.

   To combine these factors into a single decision metric, we define the following heuristic scoring function:

   $$\text{Score} = \frac{\text{frontier length}}{\text{distance to frontier}}.$$

   Here:

   - **Frontier length** represents the size of the cluster (larger clusters imply more unexplored space nearby).
   - **Distance to frontier** represents the cost required to reach that cluster.

   A higher score shows a better trade-off between potential information gain and travel effort. Thus, clusters that are both large and close to the robot are preferred over small or distant ones. (Figures 3.11, and 3.12)

4. **Selecting the Best Frontier**
   Once all frontier clusters are scored, the robot selects the cluster with the highest heuristic score as the next exploration target.

   This method allows the robot to target areas that will maximize growth of knowledge and minimize the travel distance, and to reduce the

amount of time and energy spent on exploration, while not wasting time on unstructured or insignificant frontiers.

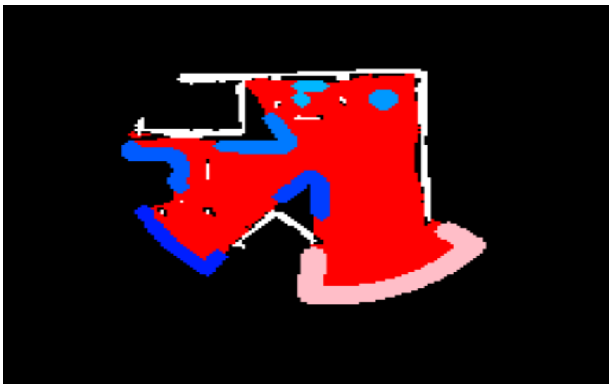

Figure 3.11: An example of clustered frontiers using DBSCAN. The pink cluster represents the frontier with the highest heuristic score and was selected as the next exploration target.

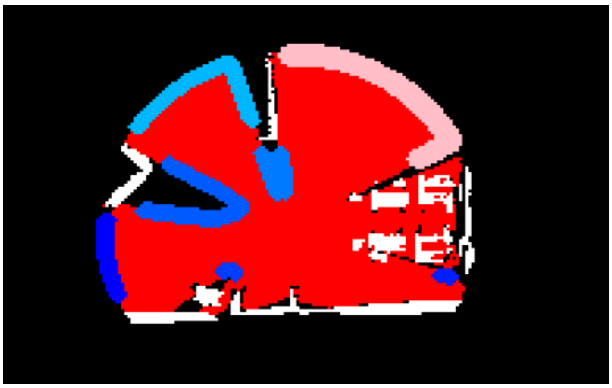

Figure 3.12: Another example of clustered frontiers using DBSCAN. The pink cluster indicates the frontier with the maximum score, chosen as the next goal for exploration.

The chosen frontier cluster's centroid (or another representative point within the cluster) becomes the goal location for the robot. A motion planning algorithm (described in the next section) is then used to compute a safe, collision-free path to reach this target.

3.2.2 Navigation

After selecting a frontier cluster as the next exploration target, the robot must compute a safe and efficient path to reach it. Since the environment is only partially known and may change during exploration, we adopted a sampling-based motion planning approach.

We use the **Rapidly-exploring Random Tree* (RRT*)** [14] algorithm to plan paths. RRT* incrementally builds a tree of collision-free states and improves path quality over time through a rewiring step, ensuring near-optimal solutions. Compared to basic RRT [13], which only focuses on quickly reaching the goal, RRT* produces shorter and smoother paths while still maintaining efficiency.

In our framework, the centroid of the highest-scoring frontier cluster is set as the goal. RRT* then computes a collision-free path from the robot's current position to this goal, minimizing unnecessary detours and travel cost. This is crucial for sequentially visiting multiple frontier clusters while conserving energy and time. (Figure 3.13)

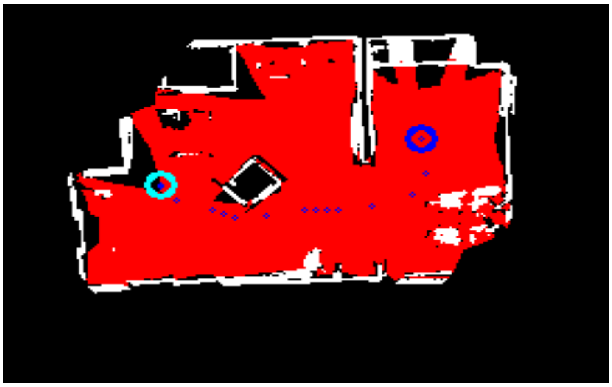

Figure 3.13: An example of the path generated by RRT* towards the robot's goal.

3.3 Object Retrieval

In this part, we present our proposed method for open-set object retrieval within the Autonomous Robotic Agent framework. Our pipeline is specifically designed to efficiently and accurately localize user-defined objects based on natural language prompts in previously unexplored indoor environments. The workflow integrates several state-of-the-art models, including Grounding DINO, CLIP, VLM, and SAM, each serving a unique purpose in the object retrieval pipeline.

3.3.1 Open-set Object Retrieval

A crucial aspect of our framework is the concept of *open-set* object retrieval. Unlike traditional object detection methods such as YOLO [18], which are limited to predefined object categories, open-set retrieval aims at identifying and localizing objects that may belong to previously unseen classes at test time. This flexibility is essential for robotic agents operating in dynamic, real-world environments where the range of possible objects is extensive and constantly evolving.

3.3.2 Grounding DINO Model

We leverage Grounding DINO [19], a powerful vision-language transformer-based model designed for open-set object detection. Grounding DINO effectively identifies new objects through textual prompts provided by the user and outputs bounding boxes with associated confidence scores indicating potential object locations (Figure 3.14). However, despite its impressive capabilities, Grounding DINO faces significant challenges:

- **False positives:** Grounding DINO often produces false positives, assigning high confidence scores to incorrect object detections.
- **Speed limitation:** Running Grounding DINO on large datasets of images is computationally demanding, limiting its usability in real-time scenarios.

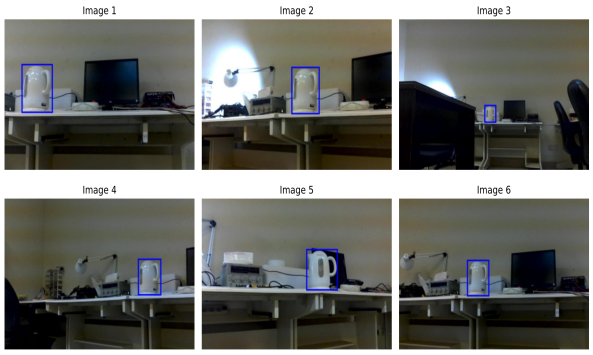

Figure 3.14: An example of object retrieval using Grounding DINO for the query “kettle”.

3.3.3 Accelerating Retrieval with CLIP Pre-filtering

To address Grounding DINO’s speed issue, we introduce an efficient pre-filtering step using CLIP. CLIP is a multi-modal embedding model that encodes both textual prompts and images into a shared embedding space, allowing for rapid similarity computations.

Our method involves the following steps:

1. Offline encoding of all collected images from the robot’s environment into CLIP embeddings.
2. At query time, encoding the user’s textual prompt with CLIP and calculating the cosine similarity between the prompt embedding and precomputed image embeddings.
3. Ranking images based on similarity scores and selecting a subset of top-ranked images for detailed analysis with Grounding DINO.

This significantly reduces the computational overhead by limiting Grounding DINO’s operation to only the most relevant subset of images.

3.3.4 VLM-based Re-ranking to Reduce False Positives

Although the CLIP-based filtering step improves efficiency, false positives from Grounding DINO remain problematic. To mitigate this, we integrate a Vision-Language Model (VLM), such as Qwen2.5-VL 7B Instruct, into the pipeline for a secondary verification step (Figure 3.15).

Our re-ranking approach works as follows:

1. Grounding DINO’s detections, sorted by confidence, are passed to the VLM.
2. Each detection box crop is separately analyzed by the VLM, checking semantic alignment with the user’s query.
3. The VLM confirms or rejects the detections, ensuring higher accuracy and significantly reducing the incidence of hallucinated object detections.

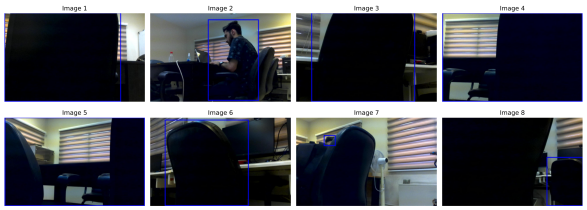

Figure 3.15: VLM-based re-ranking: despite incorrect high-confidence detections by Grounding DINO, the VLM correctly identifies the desired object based on semantic alignment.

3.3.5 Why Not Use VLM Alone?

While Vision-Language Models (VLMs) such as GPT-4V or Qwen-VL offer impressive semantic reasoning capabilities and can theoretically be used for direct object grounding, using them alone for large-scale object retrieval tasks is not practical for several reasons:

- **High computational cost:** VLMs are extremely expensive to run across large datasets, especially when used to process full-resolution images without any region proposals. Their inference time and resource demands make them unsuitable for real-time or scalable applications.
- **Lack of spatial priors:** Unlike traditional object detectors, VLMs are not inherently designed for spatial localization. When prompted with an object name, they may hallucinate object presence in arbitrary image regions, particularly when no bounding box or region of interest is specified.

To mitigate these issues and improve VLM reliability, recent research has explored techniques that guide attention to specific parts of the image. These include:

- **Visual Prompt Engineering:** Overlaying visual cues such as red circles or bounding boxes has been shown to help models like CLIP and GPT-4V focus on relevant regions, improving grounding accuracy [20].
- **Selective Blurring:** Blurring out irrelevant parts of an image can force the model to attend more closely to salient objects, enhancing zero-shot grounding performance [21].
- **Set-of-Mark Region Labels:** Annotating candidate regions with unique visual tags (e.g., A, B, C...) and referring to them explicitly in the text prompt helps VLMs disambiguate between similar-looking objects and localize targets more accurately [22].

These methods highlight the importance of structured visual input and the limitations of using VLMs alone in open-set object detection. By first identifying candidate regions using models like Grounding DINO, and then applying VLMs for semantic validation within those regions, our framework strikes an effective balance between efficiency and accuracy.

3.3.6 Object Segmentation with SAM

Once bounding boxes are verified, our system performs fine-grained object segmentation within these bounding boxes using the Segment Anything Model (SAM). SAM generates multiple potential segmentations within each box, which are then ranked again using CLIP embeddings to identify the segmentation most closely matching the user's original textual query (Figure 3.16).

This segmentation step is essential for precise localization in three-dimensional space, enabling the robot to utilize point cloud data to accurately pinpoint object positions in the environment.

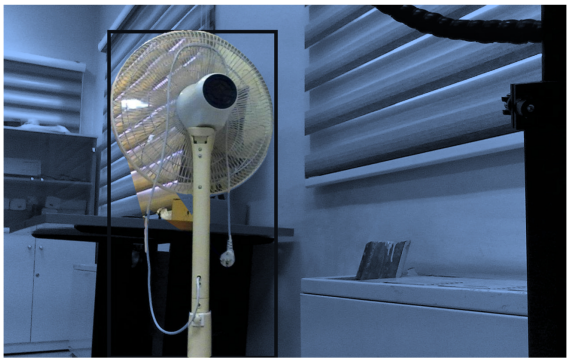

Figure 3.16: An example of SAM-based segmentation for accurately delineating the boundaries of the target object (a fan in this example).

3.3.7 3D Localization of Objects

Finally, the identified object segments are integrated with 3D point cloud data collected during environment exploration to determine precise spatial coordinates. Points from the segmentation mask are projected into the robot's reconstructed 3D voxel grid, enabling accurate determination of the object's location relative to the robot and within the broader environmental map.

3.3.8 Pipeline schema

Our object retrieval pipeline integrates advanced vision-language models and efficient computational strategies, ensuring rapid and reliable localization of user-defined objects in dynamic, open-set environments.(Figure 3.17) This modular and hierarchical approach provides both flexibility and scalability, making it particularly suitable for practical deployment in autonomous indoor robotic applications.

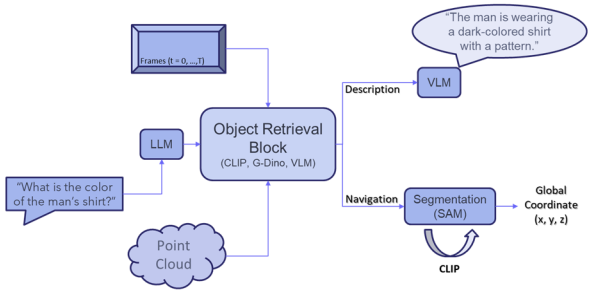

Figure 3.17: Pipeline Summary

Chapter 4

Experiments

4.1 Simulation experiment

The simulation experiments were conducted in the Webots simulator [3], an open-source robotics platform that provides realistic physics and sensor simulation. Webots includes a variety of indoor environments, such as furnished houses, kitchens, and office layouts, allowing us to evaluate exploration and navigation under different spatial configurations.

We designed a custom robot model equipped with four wheels for stable ground mobility and two RGB cameras mounted on a rotating base capable of 360° rotation. This setup allows the robot to capture a full panoramic view of its surroundings, enabling both mapping and visual-based localization. The simulated robot closely resembles a real-world mobile platform while providing a controlled environment for testing 3D mapping, exploration and object retrieval tasks.

Figure 4.1: A sample Webots environment

4.1.1 3D Reconstruction by Environment Exploration

Figure 4.2 shows snapshots from our Webots simulation during the exploration process.

The right window displays the occupancy grid being progressively completed as the robot explores. The 3D points are projected on 2D plane for better visualization of robot's exploration and navigation because our robot just moves on the ground. Initially, only a small area around the starting point is known. At each exploration step, the robot's top-mounted circular plate rotates 360°, allowing the two RGB cameras to collect visual data. This information is used to expand the known free space and refine the occupancy grid.

The left window shows the planned path from the robot's current position to the selected frontier cluster (goal). Once the path is computed using RRT*, the robot follows it to reach the goal. Upon arrival, it rotates again to scan the surroundings and further update the map. This detect–map–plan–move cycle repeats until the environment is fully explored.

4.1.2 Object Retrieval

We evaluated the object retrieval pipeline in Webots by issuing natural language navigation commands.

In this example, the query was *"navigate to tree"*. (The tree is not clearly visible in the images while robot is navigating, therefore a red arrow is used to indicate its position.)

In the first frame (Figure 4.4), the Visual Language Model (VLM) interprets the query and finds the best match for the tree. Figure 4.3 demonstrates the image found as image of tree. After that, the model identifies the spatial coordinates of the target object ("tree"). These coordinates are then

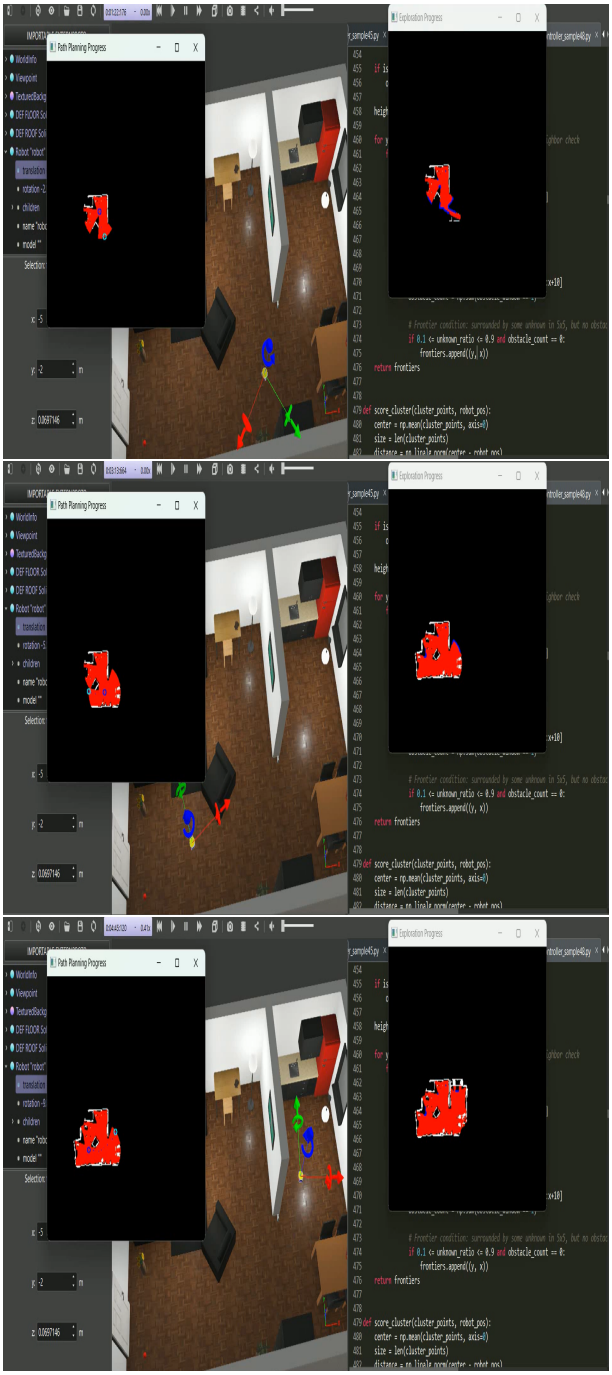

Figure 4.2: Frames from the exploration video: (top) initial map and goal selection, (middle) updated map after a 360° scan, (bottom) planned path and movement toward the next frontier.

passed to our navigation module, which sets the tree's location as the new goal. In the second frame,

Figure 4.3: Object found as the tree

the robot begins navigating toward the goal. The RRT* planner computes a safe path, and the robot follows it while continuously updating its pose. In the final frame, the robot successfully reaches the target object. This demonstrates the integration of language grounding, object localization, and autonomous navigation in a single pipeline.

4.2 Practical experiment

Practical experiments were conducted in a 5 m × 7 m indoor lab (Figure 4.5). Feature-rich textures and diverse lighting conditions simulate real apartment environments.

4.2.1 3D Reconstruction by Environment Exploration

Figure 4.6 demonstrates our robot's estimated path on the environment. Finally, figure 4.7 depicts our robot's map from the lab environment.

4.2.2 Object Retrieval

In this part we are going to show an example prompt an our framework answer/action to this prompt. The robot is asked to answer the prompt:

```
Query > What color is the man
wearing?
```

The pipeline works as follows:

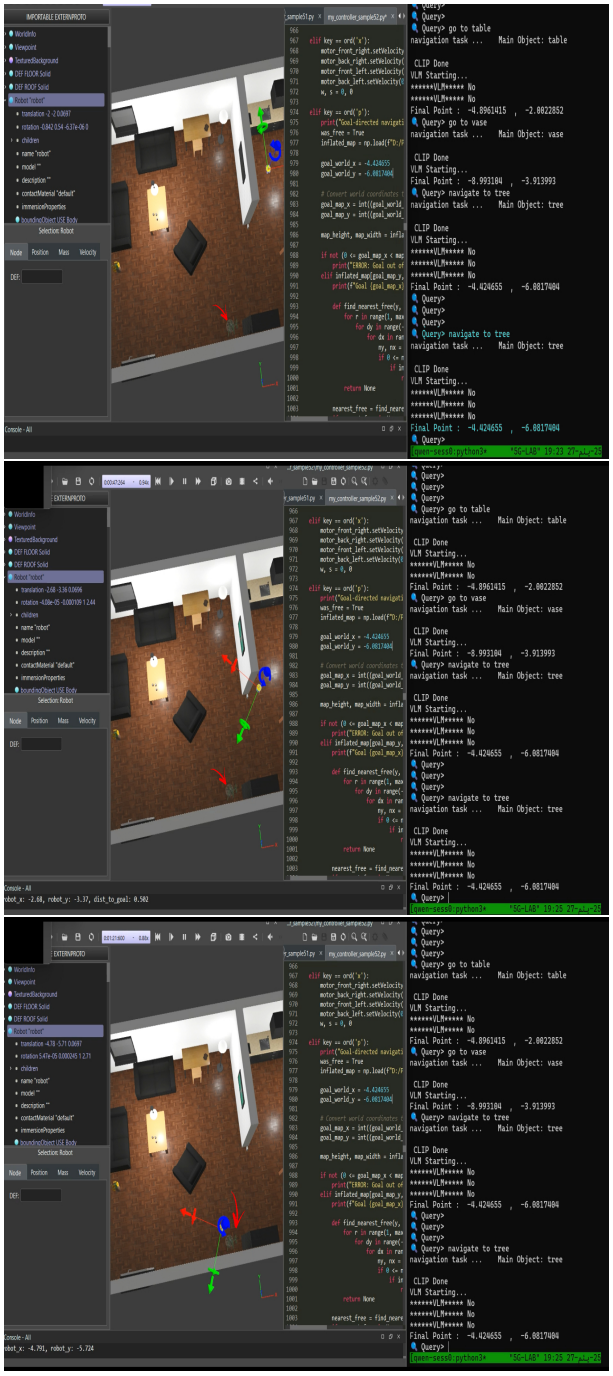

Figure 4.4: Object retrieval sequence: (top) VLM interprets the command *"navigate to tree"* and provides the goal coordinates, (middle) robot moves along the planned path toward the tree, (bottom) robot successfully arrives at the target.

Figure 4.5: Laboratory overview and annotated robot trajectory.

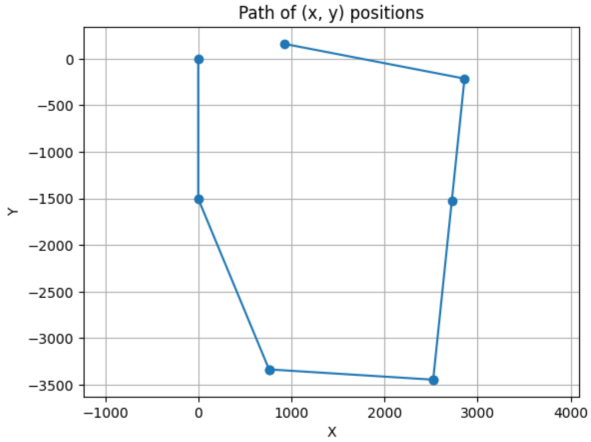

Figure 4.6: Robot path estimated using encoder data.

1. CLIP retrieves the top-matching images for the query.
2. Grounding DINO performs object detection and selects candidate bounding boxes of people.
3. The resulting images and bounding boxes are passed to the Vision-Language Model (VLM).
4. The VLM reasons over the content and provides a final answer to the question.

Figure 4.8 shows the top 8 images retrieved by the CLIP and filtered by Grounding DINO. Among them, only image 2 contains a man, correctly localized.

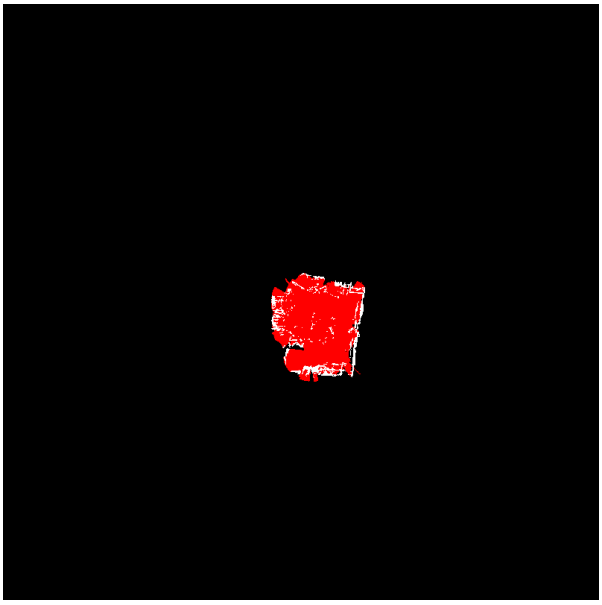

Figure 4.7: 2D projected map

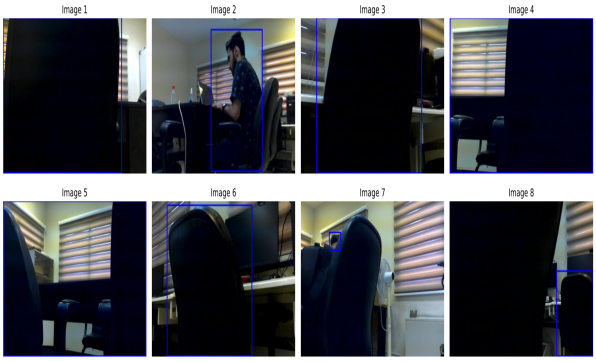

Figure 4.8: Images selected by Grounding DINO. Only one image (Image 2) shows the queried object.

The final output is shown in Figure 4.9, where the VLM selects image 2 as the most relevant and generates the following output:

```
Description > The man is wearing a
dark-colored shirt with a pattern.
```

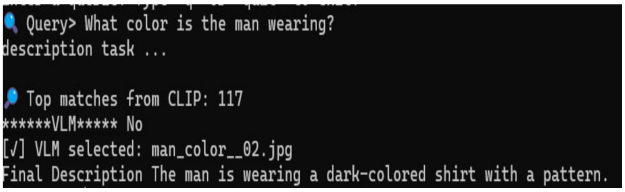

Figure 4.9: Final output of the description task generated by the VLM (The man is wearing a dark-colored shirt with a pattern.)

Chapter 5

Conclusion

In this article, we discussed about our framework which performs three steps, **3D Mapping**, **Exploration** and **Object-retrieval**. By this pipeline, we are able to autonomously explore an unknown environment and also collect semantic information from the environment. After this, when a user prompts the robot to do a specific navigation or description task, the robot can easily respond accordingly to his command.

In future iterations, we propose the following improvements:

- Integration of additional sensors, such as an Inertial Measurement Unit (IMU), to enhance localization accuracy under challenging conditions.
- Implementation of multi-view feature matching for robust object identification, even in scenarios with heavy occlusion or ambiguous visual features.
- Development of methods to support multi-object retrieval, description, and comparison within the environment.
- Introduction of temporal change detection capabilities to identify modifications or updates within explored environments over time.
- Enhancements in the exploration strategy to optimize data collection, focusing on capturing richer and denser point clouds.

Chapter 6

Supplementary Material

6.1 Robotic Platform

Our experiments employ a customized **Hiwonder ArmPi Pro** mobile robot (Figures 6.1–6.2). Key specifications are listed in Table 6.1.

Figure 6.1: Hiwonder ArmPi Pro with original 6-DOF arm.

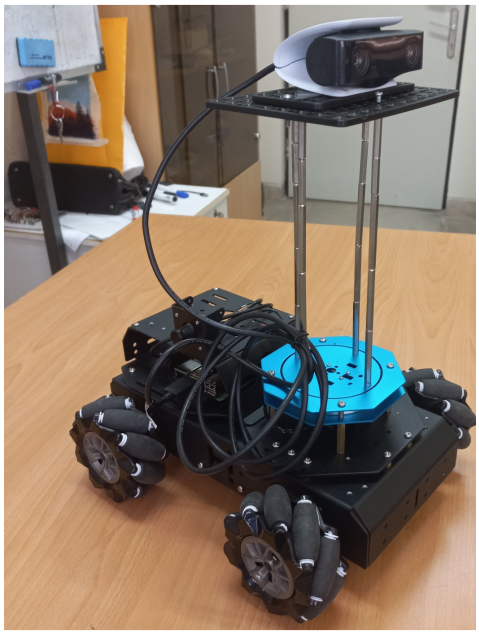

Figure 6.2: Our modified platform with stereo camera replacing the arm.

Table 6.1: Robot Hardware Specifications

Component	**Details**
Chassis	4 × DC encoder motors, Mecanum wheels
Compute	Raspberry Pi 5 (8 GB RAM)
Vision	Stereo camera (1280 × 720@30 FPS)
Aux Sensor	HD wide-angle RGB camera
Aux Sensor	(Planned) 9-axis IMU
Power	12 V Li-ion battery, 5000 mAh
Comms	Dual-band Wi-Fi 6

6.2 Locomotion and Control

The Mecanum wheel configuration enables omnidirectional motion (Figure 6.3). Wheel odometry is estimated from motor encoders; if encoder feedback is unavailable, dead-reckoning forward kinematics is applied. Slippage remains a challenge—future work will fuse the upcoming IMU with wheel odometry for drift correction.

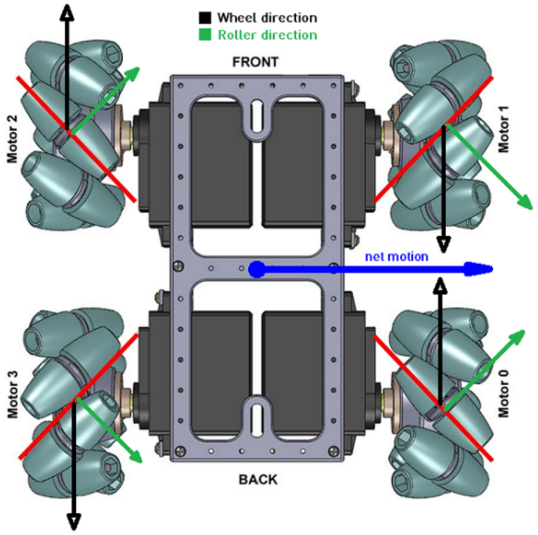

Figure 6.3: Leftward strafing with Mecanum wheels [4].

6.3 Processing Distribution

Only light preprocessing (image capture, compression) runs on-board. All compute-intensive tasks—feature extraction, CLIP ranking, Grounding DINO inference, VLM re-ranking, SAM segmentation—execute on a remote server (NVIDIA RTX3090 24 GB GPU) via Wi-Fi. This split keeps the robot lightweight while ensuring real-time performance.

6.4 Models

- **CLIP ViT-B/32** (2021) – offline image embedding and prompt similarity ranking.
- **Grounding DINO-T** (2024) – open-set detector producing class-agnostic bounding boxes.
- **Qwen2.5-VL 7B Instruct (VLM)** – large-scale vision-language model used for re-ranking and hallucination filtering.
- **Qwen3-235b-a22b (LLM)** - Query handling and Chain-of-Thought.
- **SAM-H (sam_vit_h_4b8939)** (2023) – Segment Anything Model (Hi-Res) for mask generation inside verified boxes.

6.5 Reproducibility

Source code and configuration files are available at: `https://github.com/aliekhtera/BEng_Project`.

References

[1] J. Edstedt, Q. Sun, G. Bökman, M. Wadenbäck, and M. Felsberg, "Roma: Robust dense feature matching," 2023.

[2] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97: Towards New Computational Principles for Robotics and Automation*, pp. 146–151, IEEE, 1997.

[3] Cyberbotics Ltd., "Webots: Open-source robot simulator." `https://cyberbotics.com/`. Accessed: 2025-07-21.

[4] Iron Reign Robotics, "Mecanum diagram - crab drive," 2020. Accessed: 2025-07-22.

[5] Y. Li and J. Ibañez-Guzmán, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *CoRR*, vol. abs/2004.08467, 2020.

[6] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth anything v2," 2024.

[7] J. Wang, M. Chen, N. Karaev, A. Vedaldi, C. Rupprecht, and D. Novotny, "Vggt: Visual geometry grounded transformer," 2025.

[8] V. Leroy, Y. Cabon, and J. Revaud, "Grounding image matching in 3d with mast3r," 2024.

[9] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," 2020.

[10] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, "Dust3r: Geometric 3d vision made easy," 2024.

[11] R. Murai, E. Dexheimer, and A. J. Davison, "Mast3r-slam: Real-time dense slam with 3d reconstruction priors," *arXiv preprint arXiv:2412.12392*, 2024.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[13] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] R. Fu, J. Liu, X. Chen, Y. Nie, and W. Xiong, "Scene-llm: Extending language model for 3d visual understanding and reasoning," 2024.

[16] A. Werby, F. Akhtar, S. Ahmed, Y. Lee, Y. He, S. Srivastava, and J. Kosecka, "Hierarchical open-vocabulary 3d scene graphs for language-grounded robot navigation," *arXiv preprint arXiv:2403.17846*, 2024.

[17] D. Deng, "Dbscan clustering algorithm based on density," in *2020 7th International Forum on Electrical Engineering and Automation (IFEEA)*, pp. 505–508, IEEE, 2020.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[19] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," 2024.

[20] A. Shtedritski, C. Rupprecht, and A. Vedaldi, "What does clip know about a red circle? vi-

sual prompt engineering for vlms," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11987–11997, 2023.

[21] L. Yang, Y. Wang, X. Li, X. Wang, and J. Yang, "Fine-grained visual prompting," *Advances in Neural Information Processing Systems*, vol. 36, pp. 24993–25006, 2023.

[22] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao, "Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v," *arXiv preprint arXiv:2310.11441*, 2023.