# Real-time Object Detection and Semantic Mapping on CPU-Powered Mobile Robot

Mikihisa Ishino[1], Ryuto Ishibashi[2], Zhenling Su[2], Lin Meng[1]
{ri0135ve@ed.ritsumei.ac.jp, ri0097fx@ed.ritsumei.ac.jp, gr0663hi@ed.ritsumei.ac.jp, menglin@fc.ritsumei.ac.jp}

[1]Department of Electronic and Computer Engineering, Ritsumeikan University, Kusatsu, Shiga, Japan
[2]Graduate School of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga, Japan

**Abstract.** Deploying deep learning object detection models on resource-constrained mobile robots is a significant challenge because these models typically require power-hungry GPUs for high-performance inference. This requirement creates a critical bottleneck for mobile robots equipped only with CPU-based computers, such as Raspberry Pi 4B. We propose a semantic mapping system integrating an int8-quantized YOLOv10p model with ROS2 and SLAM. The system runs the detection node projecting 2D detections from a monocular camera into a map using a ground-plane assumption. Our real-world experiments provide the critical performance baseline. The system achieves 100% Recall. However, this system suffers from low 54.5% Precision. We demonstrate through qualitative analysis that the low precision is caused by clusters of False Positives, stemming from the instability of the monocular 3D projection method and inaccuracy in estimation of object detection. Our work confirms the feasibility of integrating YOLOv10 into the CPU-only ROS2 system.

**Keywords:** SLAM, Object Detection, YOLOv10, ROS2, Semantic Mapping, Edge Computing, Mobile Robotics

## 1 Introduction

Integration of artificial intelligence (AI) into mobile robotics has led to significant advances in automation. Autonomous robots are increasingly expected to operate in complex, human-centric environments, requiring not only navigation capabilities, but also understanding of the semantics of their surroundings [1]. However, despite rapid progress in AI, particularly in object detection, deploying AI models on edge devices remains a challenge in practice due to constraints on computation, memory, and power consumption [2, 3].

The social demand for autonomous systems, such as domestic service robots and warehouse logistics robots, is growing. The applications demand real-time perception and decision-making. However, many edge devices, especially low-cost mobile robots, have limited resources and operate

IIKI 2025, December 18-21, Chengdu, People's Republic of China

DOI 10.4108/eai.18-12-2025.2365265

only with Central Processing Unit (CPU) [4]. High-performance detection models often rely on power-hungry Graphics Processing Units (GPUs), making it unsuitable for utilization in CPU-only platforms. This disparity creates a critical bottleneck, as the high computational cost of advanced models leads to unacceptable latency, compromising real-time capabilities, and increasing the risk of misdetections in dynamic settings [5]. To mitigate these computational costs, model compression techniques such as pruning and quantization have been researched to enable efficient inference on edge hardware [6, 7].

From a technical perspective, there is little research validating the performance of state-of-the-art object detection models in CPU-only environments. This gap is particularly pronounced for recent models such as YOLOv10, introduced in 2024 [8]. Although YOLOv10 promises high efficiency, performance data on CPU platforms is severely lacking. Our study directly addresses this gap by verifying the real-world operation of the lightweight model on the CPU-powered mobile robot.

Recent advances in Semantic Mapping have focused on integrating object information by deep learning with traditional Simultaneous Localization and Mapping (SLAM) frameworks [9]. The system aims to build maps that are not just geometric, but also enriched with semantic information. However, critical reviews of the existing literature reveal that the majority of these high-performance systems, such as those built on ORB-SLAM3 [10] or DynaSLAM [11], depend heavily on powerful GPUs or high-performance CPU to achieve real-time processing. The reliance on GPUs hinders their widespread adoption in cost-effective, low-power mobile robots lacking dedicated graphics hardware.

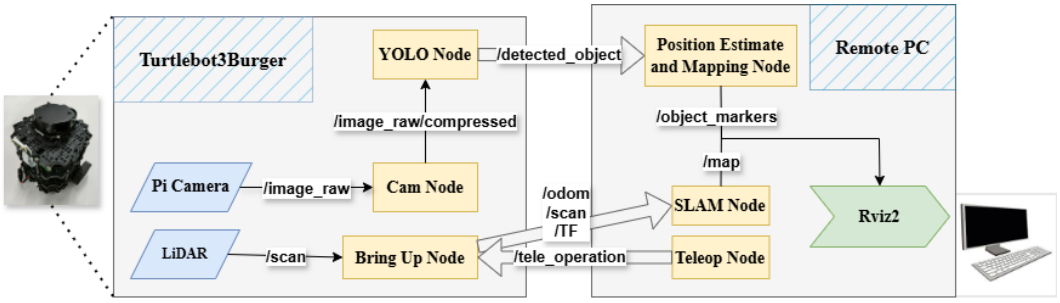



**Fig. 1.** This figure shows the overall structure of our proposed system, the functions of the nodes running a TurtleBot3 and a Remote PC, and the topics each node publishes and subscribes to. On the mobile robot side, the TurtleBot acquires hardware information, sensor data from the onboard LiDAR, and image data from the Pi camera. Furthermore, YOLO Node performs object detection on the image data using YOLOv10p and publishes the necessary information to the Remote PC. On the Remote PC side, the system estimates the map positions of detected objects from the received data, integrates the received data, and provides visualization to the user using Rviz.

Furthermore, while lightweight models such as YOLOv7-tiny [12], YOLOv8n [13], or YOLOv9t [14] have been tested on edge devices, performance on CPU-only platforms such as Raspberry Pi remains a significant bottleneck [15]. Although the latest YOLOv10 model offers architectural im-

provements for efficiency, performance benchmarks on CPU-only systems, especially when integrated into full robotics stack with ROS2 and SLAM, have rarely been established yet.

To bridge the critical gap, our paper proposes a real-time object detection and semantic mapping system specifically designed for CPU-powered mobile robots. We integrate a lightweight YOLOv10 model with the ROS2 framework and the SLAM algorithm, enabling a robot to build the semantic map of the environment. The main contributions of our work are threefold.

- We demonstrate the integration of the state-of-the-art object detection model into the ROS2-based SLAM system that operates entirely on a CPU.
- We provide a comprehensive performance evaluation of this system's real-time processing capability (FPS) in the real-world environment.
- We validate the system's accuracy by evaluating mapping Precision and Recall, and by qualitatively assessing the positional accuracy of the final semantic map, confirming the feasibility on resource-constrained platforms.

## 2 Proposed System Architecture

Our proposed system is designed to perform real-time object detection and semantic mapping on a mobile robot with constrained computational resources. The architecture is distributed between a mobile robot platform which handles sensing and detection, and a Remote PC which manages localization and visualization. The entire system is integrated using Robot Operating System 2 (ROS2).

### 2.1 System Overview

Our hardware platform consists of a TurtleBot3 Burger (TB3) and a Remote PC. TB3 is equipped with a Raspberry Pi 4B and a monocular camera. The robot serves as the primary edge device. The remote PC is responsible for computationally intensive tasks that are not performed on the robot, as well as for system control and map visualization.

The data flow is managed by ROS2 topics, as illustrated in Fig. 1. The architecture is composed of the TB3 and the Remote PC, and each device handles distinct tasks.

The TB3 executes three main nodes. The first is Bring Up Node. This node launches the necessary applications to enable control and monitoring of the robot's hardware from the ROS2 system. The second is the Camera Publish Node. This node continuously captures images from the monocular camera and publishes images as sensor_msgs/CompressedImage on /image_raw/compressed topic. The third is Object Detection Node. This node subscribes to /image_raw/compressed topic, performs real-time object detection, and publishes the results of bounding boxes and class IDs as the custom detection message on /yolo/detections topic and the image with the detection results drawn as sensor_msgs/CompressedImage on /yolo/image_with_boxes topic.

The Remote PC executes the three main nodes. The first node is the Image Subscribe Node. This node subscribes to image messages published by /yolo/image_with_boxes topic and /image_raw /compressed topic of TB3 to view on the Remote PC to check whether the monocular camera and

Object Detection Node work correctly. The second node is SLAM Node. This node processes laser scan data from the LiDAR of TB3. The scan data generates the 2D occupancy grid map published on /map topic and provides TransFormation tree (TF) that localizes the robot(base_link) within the map frame. The third node is the Transform and Mapping Node. This node is the core of our semantic mapping logic and subscribes to /yolo/detections topic and /tf tree. The purpose is to calculate the 3D position of detected objects in the map frame and publish them as visualization_msgs/Marker on /object_markers topic for display in RViz.

### 2.2 Lightweight Object Detection on the Edge

To achieve real-time performance on the resource-constrained Raspberry Pi 4B, we adopt a YOLOv10 model. The standard minimal YOLOv10 model has approximately 2,700,000 parameters. However, in this experiment, we further reduce the model size, cutting the number of parameters to approximately 980,000 by modifying the depth multiple factor to 0.15 and the width multiple factor to 0.12. We name this variant YOLOv10p. This process resolves the impractical model size issue of YOLOv10n in CPU-only environments. The detection node runs directly on the Raspberry Pi, minimizing latency by processing the image stream at the source.

### 2.3 3D Position Estimation and Semantic Mapping

The key challenge of using the monocular camera is the lack of direct depth information. To project 2D detected objects into the 3D map, we implement a method based on ray-casting and a ground plane assumption. The entire calculation is performed within the Transform and Mapping Node's callback function. The process is divided into five stages.

The first stage is Robot Pose Acquisition. For each incoming detection message from /yolo /detections, the system first acquires the current transformation from the map frame to the camera_link frame using ROS2 tf_buffer. The frame provides the camera's 3D position $P_{cam} = (x_c, y_c, z_c)$ and the orientation as a rotation matrix $R_{map \leftarrow cam}$ relative to the global map generated by the SLAM node.

The second stage is Target Pixel Selection. Instead of using the center of the detection's bounding box, we select the bottom-center pixel $(u, v)$ of the box. This selection is based on the critical assumption that the detected object is resting on the ground. Here, $(c_x, c_y)$ represent the center coordinates of the bounding box, and $h$ represents the height of the bounding box. The pixel coordinates are calculated as follows.

$$u = c_x \tag{1}$$

$$v = c_y + \frac{h}{2} \tag{2}$$

The third stage is 3D Ray Back-Projection. Using the intrinsic parameters of the camera from the CameraInfo topic, we back-project the 2D pixel $(u, v)$ into the 3D unit vector ray $\vec{r}_{cam\_optical}$ in the camera's optical frame (Z-axis forward). This ray is then converted to the standard robot mechanical frame (X-axis forward) using fixed rotation $R_{mech \leftarrow optical}$:

$$\vec{r}_{cam\_mech} = R_{mech \leftarrow optical} \cdot \vec{r}_{cam\_optical} \tag{3}$$

The fourth stage is Ground Plane Intersection. The ray is transformed into the global map frame using the camera's orientation: $\vec{r}_{map} = R_{map \leftarrow cam} \cdot \vec{r}_{cam\_mech}$. The 3D line representing the ray can be parameterized in the map frame, originating from $P_{cam}$ with direction $\vec{r}_{map}$.

We assume that the ground is a flat plane at $z = 0$ in the map frame. The 3D position of the object $P_{obj} = (x, y, z)$ is found by calculating the intersection of this ray with the $z = 0$ plane. The parametric equation for the ray is the following.

$$P(\lambda) = P_{cam} + \lambda \cdot \vec{r}_{map} \tag{4}$$

We solve for the parameter $\lambda$ where the Z-component is zero:

$$P(\lambda)_z = z_c + \lambda \cdot (\vec{r}_{map})_z = 0 \tag{5}$$

This yields:

$$\lambda = -z_c / (\vec{r}_{map})_z \tag{6}$$

The intersection is only considered valid if $\lambda > 0$ (in front of the camera) and $(\vec{r}_{map})_z$ is not near zero. Thus, the ray is not parallel to the ground.

The fifth stage is Semantic Map Plotting. The final 3D object position $P_{obj}$ is calculated by substituting $\lambda$ back into Equation (6). To avoid redundant plotting of the same object, we implement a spatial filter. A new object marker is only published if the 2D planar distance $(P_{obj})_x, (P_{obj})_y$ is greater than a predefined threshold, min_distance, from all previously plotted objects. This method ensures a clean and uncluttered semantic representation of this experiment environment.

## 3 Experimental Setup

To validate the performance of our proposed system, we conduct experiments using a real-world mobile robot. We focus on evaluating our system's real-time processing capability, detection accuracy, and mapping quality.

### 3.1 Hardware and Software Setup

Our experimental platform consists of a mobile robot and a Remote PC. On the mobile robot, we use the TurtleBot3 Burger, equipped with a Raspberry Pi 4B (1GB RAM, augmented with 2GB of swap memory) as the onboard CPU. Onboard Sensors, a Raspberry Pi Camera Module v2.1 is used for visual input, and the standard TurtleBot3 LDS-01 LiDAR is used by the SLAM algorithm. Regarding the Remote PC, a desktop PC with an Intel Core i5-14500 CPU (14 cores, 20 threads, 24MB L3 cache) is used to run the SLAM Node, the Transform and Mapping Node, and RViz2. In terms of Software, our system operates on ROS2 Humble on Ubuntu 22.04. The object detection model is trained and evaluated using the open-source Ultralytics framework, converted to an ONNX format, and deployed as a node in the TB3.

### 3.2 Dataset and Model Training

Our study uses a dataset comprising 6,495 images. The dataset includes publicly available images containing items such as plastic bottles, cans, and glassed bottles[16, 17, 18, 19, 20], supplemented by data collected in an experimental environment. These images are divided into 5,058 for training, 719 for validation, and 718 for testing to train the model. The training is conducted with a batch size of 16 and an image size of 640 pixels. The training continues using early stopping until the accuracy is not updated for 30 consecutive iterations and the weights achieving the highest accuracy up to that point are used. To optimize the model for deployment on TB3, we apply static int8 post-training quantization (PTQ) provide by the ultralytics framework. The training dataset described in this section is used as calibration data for the quantization process. The final quantized model is then converted to the ONNX format for deployment on the edge device.

### 3.3 Evaluation Metrics

We evaluate the performance of our system based on three key metrics: real-time capability, detection and mapping accuracy, and positional map quality.

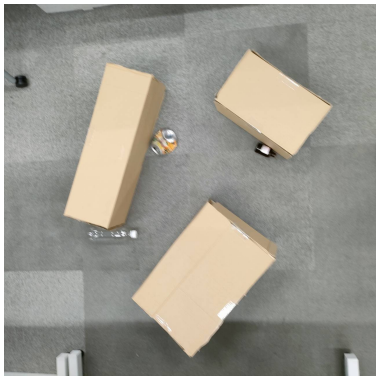
In terms of Real-time Performance (FPS), the real-time capability of YOLOv10p is measured. We calculate the total latency by summing the time required for image preprocessing, model inference, and detection post-processing. The effective Frames Per Second (FPS) is then derived from this total latency. We also measure the image processing capability of YOLOv10p on the Raspberry Pi 4B disconnected from the system.

In terms of Detection and Mapping Accuracy, we assess accuracy at two levels. The first level is Model Accuracy. In this level, YOLOv10p is evaluated in the custom dataset to establish the baseline mAP (mean Average Precision). The second level is System Accuracy. We evaluate the ability of the entire semantic mapping system to correctly identify and plot objects. We count the number of True Positives (TP, correctly plotted object), False Positives (FP, incorrectly plotted objects), and False Negatives (FN, the existing object not plotted) to calculate the overall Precision and Recall of our system.

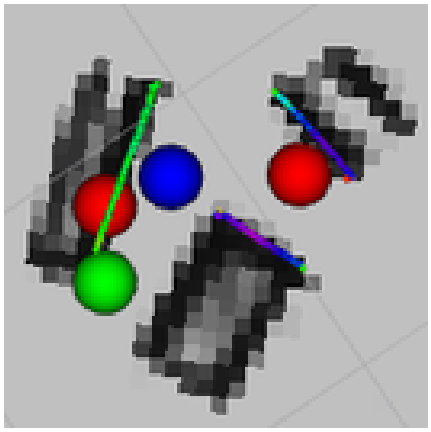
In terms of Positional Map Quality, due to the difficulty in establishing precise ground truth coordinates for objects in the real-world test environment, we perform a qualitative evaluation of positional accuracy. The final semantic map generated by the system is visually compared with the actual physical layout of the objects to assess the overall correctness and plausibility of the map. When plotting objects on the map, we impose the following two conditions. The first condition is the coordination of y and x. The y-coordinate of the midpoint of the bottom edge of the bounding box must be within the bottom third of the image, and the x-coordinate must be at least 80 pixels away from the left or right edge of the image.

The second condition is the distance between the plot markers. An object must be at least 15 cm away from any previously plotted coordinates.

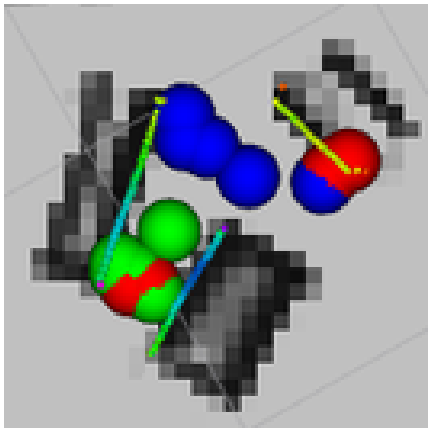
These conditions are crucial to the reliability of our proposed system. The results when these conditions are absent or relaxed are shown in Fig. 2. In this figure, (a) is the Real-world environment for this test. (b) is the case of correct conditions. (c) and (d) are the cases of removing each condition from case (b), and (e) is the case of removing both conditions.



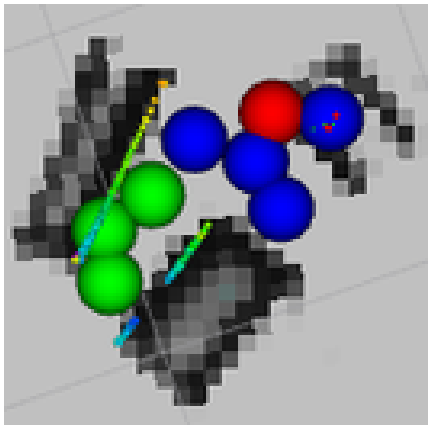
**(a)** Real-world environment



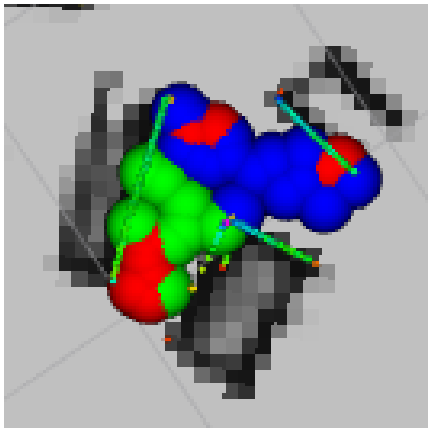
**(b)** With proposed constraints



**(c)** Reduced distance threshold



**(d)** Unrestricted coordinates



**(e)** Both constrains relaxed

**Fig. 2.** This figure confirms the validity of the distance conditions between plot markers and the coordinate conditions for detecting objects to plot, as specified in the operational test of the experiment. Green plots represent plastic bottles, blue plots represent cans, and red plots represent glassed bottles. This figure shows we can dramatically reduce the number of false positives.

## 4 Results and Discussion

This section presents the experimental results based on the evaluation metrics defined in Section 3.3. We analyze the system's real-time performance, detection and mapping accuracy, and the qualitative evaluation of the generate semantic map.

### 4.1 Real-time Performance on CPU

Table 1 compares the real-time performance of YOLOv10p with competitive models of the YOLO family. Notably, despite the theoretical efficiency of the standard YOLOv10n, it exhibits higher latency than YOLOv9t on the Raspberry Pi 4B, likely due to memory access bottlenecks inherent to its depthwise-heavy architecture on edge CPU. In contrast, YOLOv10p achieves the lowest latency of 520.41 ms. This result demonstrates that YOLOv10p successfully retains the benefits of the attention mechanism found in YOLOv10n while overcoming memory access inefficiencies, thereby outperforming even the hardware-friendly YOLOv9t.

**Table 1:** Comparison of Real-Time Performance for Individual Models on Raspberry Pi 4B

| Model | Average Latency (ms) | Average FPS |
|---|---|---|
| YOLOv8n | 1220.16 | 0.82 |
| YOLOv9t | 679.92 | 1.47 |
| YOLOv10n | 1115.38 | 0.90 |
| YOLOv10p | 520.41 | 1.92 |

To validate our main claim of real-time capability, we measure the performance of YOLOv10p Node operating directly on the Raspberry Pi 4B. Our system is tasked with processing the continuous video stream while the robot is navigating through the test environment.

The average latency for each stage of the object detection pipeline is detailed in Table 2. The total latency is measured at 1223.24 ms, resulting in an effective processing speed of 0.82 FPS. This result confirms that our system demonstrates the feasibility of integrating SLAM and real-time object detection using YOLOv10p on a CPU-only edge device.

**Table 2:** Onboard Detection Performance (YOLOv10p on Raspberry Pi 4B)

| Processing Stage | Average Latency (ms) |
|---|---|
| Image Pre-processing | 21.60 |
| Model Inference | 1198.45 |
| Post-processing | 3.19 |
| **Total Latency** | 1223.24 |
| **Effective FPS** | 0.82 |

### 4.2 Detection and Mapping Accuracy

We evaluate the accuracy of our system by first assessing the model's standalone performance and then the entire mapping precision and recall.

In terms of Model Accuracy, Table 3 compares competitive models within YOLO family with YOLOv10p used in our study. The table shows that YOLOv8n achieves the highest accuracy. The reason is YOLOv8's maturity. YOLOv8 has been maintained for a long time by Ultralytics. It suggests that the hyperparameters, learning rate, and augment settings have been finely tuned to adapt well to a very wide range of custom datasets. In contrast, YOLOv9 and YOLOv10 have focused on adapting new architectures to YOLO family, and the learning parameters have not been adjusted as thoroughly as those of YOLOv8. This factor should be addressed as the future task. Regarding YOLOv10p adopted in the system, Precision remains nearly unchanged compared to other models, and it even shows improvement when compared to YOLOv10n. However, significant decline is observed in Recall and mAP@50-95 specifically. The result indicates that while the model demonstrates considerable accuracy in object detection itself, the reduction in model size suggests it struggles to extract features effectively from challenging test images.

**Table 3:** Comparison of Detection Accuracy on Custom Dataset

| **Model** | **Precision** | **Recall** | **mAP@50** | **mAP@50-95** |
|---|---|---|---|---|
| YOLOv8n | 96.2 | 94.0 | 97.8 | 87.8 |
| YOLOv9t | 95.4 | 92.7 | 96.3 | 86.1 |
| YOLOv10n | 94.0 | 92.5 | 96.8 | 86.3 |
| YOLOv10p | 95.0 | 86.0 | 94.6 | 80.7 |

In terms of Semantic Mapping Accuracy, more critical to our application is the system's ability to correctly plot detected objects. We conduct 10 repeated tests of the proposed system in a real-world environment, placing one object from each class in the test setup. We manually count the number of TP, FP, and FN, calculate Precision and Recall based on the counts, and summarize the results in Table 4.

The system achieves a Precision of 54.5% and a Recall of 100%. Notably, the experiment succeeds in reducing the number of false negatives (FN) to zero. However, 25 false positives (FP) occur, significantly lowering the Precision value. Two factors are suggested as causes for the FP occurrence. The first is the position estimation error of the monocular camera. Preliminary experiments confirm that the area estimation using the monocular camera employed in the experiment suffers from significant error. Considering this, during the test phase of this experiment, we impose restrictions on object position plotting and configure settings with practicality in mind. However, we are unable to completely eliminate these errors during the test phase. The second factor is the error inherent to the object detection task itself. The bounding boxes used for object position estimation in the experiment are insufficient for accurately estimating complete ground contact points, suggesting they only provide simplified position estimate.

**Table 4:** Mapping Accuracy (Precision and Recall)

| Metric | Plastic Bottle | Can | Glassed Bottle | All |
|---|---|---|---|---|
| TP | 10 | 10 | 10 | 30 |
| FP | 12 | 7 | 6 | 25 |
| FN | 0 | 0 | 0 | 0 |
| **Precision** | **45.5%** | **58.8%** | **62.5%** | **54.5%** |
| **Recall** | **100%** | **100%** | **100%** | **100%** |

### 4.3 Qualitative Map Evaluation

Finally, we perform a qualitative assessment of the positional accuracy of the semantic map to visually interpret the quantitative result. Fig. 3 provides a side-by-side comparison of the system's output with the ground truth.

The top row, (a), (b), and (c), displays the final semantic maps generated by our system and visualized in RViz. The bottom row, (d), (e), and (f), shows photographs of the corresponding ground truth layout of the experimental environment.

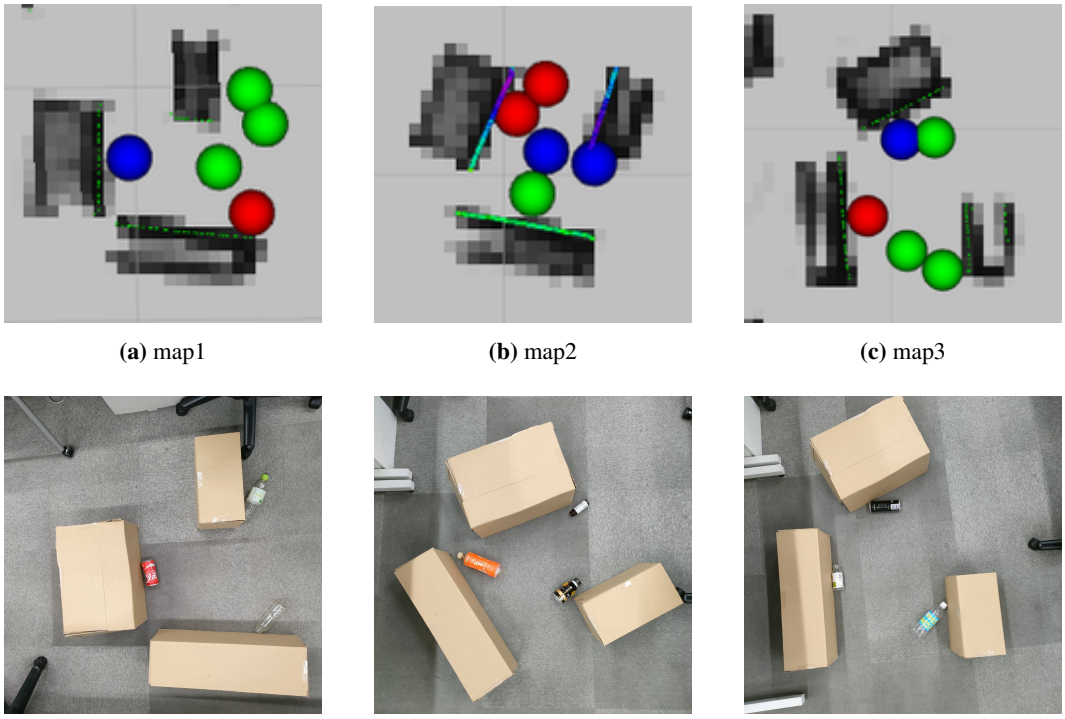
As the figure demonstrates, the system successfully generates a 2D occupancy grid map via SLAM and overlays it with semantic markers. The clear correspondence exists between the actual object locations as shown on the bottle and can in Fig. 3(d) and the correctly plotted TP markers on the map Fig. 3(a). Visual evidence corroborates the 100% Recall reported in Table 4, confirming that the system reliably detects and plots all target objects it encounters.

However, the maps also clearly illustrate the critical issue identified in our quantitative analysis: the low 54.5% Precision. Around each true object's location, clusters of FP markers are visible. This phenomenon comes from two primary factors. First, motion-induced blur during robot navigation leads to FP, where the model incorrectly assigns classes to background features. Second, slight variations in the robot's pose or minor inaccuracies in the bounding box's ground contact point are amplified by the ray-casting projection, causing position estimates to scatter.

While our 15 cm spatial filter prevents identical detections from being replotted, it is not sufficient to consolidate these closely-grouped but erroneous FP clusters. Therefore, the qualitative evaluation confirms that while our system successfully achieves the primary goal of mapping true objects, the practical utility is hindered by projection-induced false positives. This highlights that improving the stability of the 3D estimation is the most critical area for future work.

## 5 Conclusion

By integrating the lightweight YOLOv10p model with the ROS2 framework and a SLAM algorithm, we propose the complete data pipeline running on the TurtleBot3 equipped with the Raspberry Pi 4B. We demonstrate that while the standalone YOLOv10p model is the fastest among tested variants, the fully integrated ROS2 system operates at 0.82 FPS. The result, while insufficient for applications demanding high-speed interaction, provides a critical performance baseline for CPU-only



**(a)** map1 **(b)** map2 **(c)** map3

**(d)** map1 **(e)** map2 **(f)** map3

**Fig. 3.** This figure presents three examples comparing the operational test environment with the semantic maps obtained within that environment. Each pair of figures (top and bottom) represents a single test. As shown, object detection succeeds in each test, but false detections also occur frequently.

semantic systems.

The system's mapping evaluation reveals the significant trade-off. We achieve excellent 100% Recall. However, the result is coupled with a very low 54.5% Precision. This phenomenon is a direct consequence of the instability of our ray-casting projection method amplifying minor errors in robot pose and bounding box estimation.

Future work must address the limitations. To improve precision, we must move beyond simple bounding boxes, which this study confirms are insufficient for stable ground-contact estimation. We plan to incorporate semantic segmentation or other Grounding Assumption tasks to extract precise ground contact points. Furthermore, we plan to explore data association techniques to merge the FP clusters into single high-confidence object hypotheses. To address latency, we investigate advanced model optimization to further accelerate CPU performance.

## References


[1] Kostavelis I, Gasteratos A. Semantic mapping for mobile robotics tasks: A survey. Robotics and Autonomous Systems. 2015;66:86-103.

[2] Chen J, Ran X. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. ACM Computing Surveys (CSUR). 2020;53(4):1-37.

[3] Sze V, Chen YH, Yang TJ, Emer JS. Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE. 2017;105(12):2295-329.

[4] Bianco S, Cadene R, Celona L, Napoletano P. Benchmark analysis of representative deep neural network architectures. IEEE Access. 2018;6:64270-7.

[5] Mittal P. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. Artificial Intelligence Review. 2024;57(9):242.

[6] Han S, Mao H, Dally WJ. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:151000149. 2015.

[7] Choudhary T, Mishra V, Goswami A, Sarangapani J. A comprehensive survey on model compression and acceleration. Artificial Intelligence Review. 2020;53(7):5113-55.

[8] Wang A, Chen H, Liu L, Chen K, Lin Z, Han J, et al. Yolov10: Real-time end-to-end object detection. Advances in Neural Information Processing Systems. 2024;37:107984-8011.

[9] McCormac J, Handa A, Davison A, Leutenegger S. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In: 2017 IEEE International Conference on Robotics and automation (ICRA). IEEE; 2017. p. 4628-35.

[10] Campos C, Elvira R, Rodríguez JJG, Montiel JM, Tardós JD. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. IEEE transactions on robotics. 2021;37(6):1874-90.

[11] Bescos B, Fácil JM, Civera J, Neira J. DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes. IEEE robotics and automation letters. 2018;3(4):4076-83.

[12] Wang CY, Bochkovskiy A, Liao HYM. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition; 2023. p. 7464-75.

[13] Ultralytics. Ultralytics YOLOv8. GitHub; 2023. `https://github.com/ultralytics/ultralytics`.

[14] Wang CY, Yeh IH, Mark Liao HY. Yolov9: Learning what you want to learn using programmable gradient information. In: European conference on computer vision. Springer; 2024. p. 1-21.

[15] Gu S, Meng W, Sun G. Streamlining YOLOv7 for Rapid and Accurate Detection of Rapeseed Varieties on Embedded Device. Sensors (Basel, Switzerland). 2024;24(17):5585.

[16] Labs D. Alcohol Bottle Images — Glass bottles. Kaggle; 2021. `https://www.kaggle.com/datasets/dataclusterlabs/alcohol-bottle-images-glass-bottles`.

[17] Students EDS. Bottles and Cans Images. Kaggle; 2020. https://www.kaggle.com/datasets/moezabid/bottles-and-cans.

[18] Yusron U. Bottle Dataset. Kaggle; 2024. https://www.kaggle.com/datasets/muhamadusriyusron/bottle-dataset.

[19] Pivnenko A. Glass and Plastic Bottles. Kaggle; 2025. https://www.kaggle.com/datasets/antonpivnenko/glass-and-plastic-bottles.

[20] Sah S. Plastic Bottles in the wild Image Dataset. Kaggle; 2022. https://www.kaggle.com/datasets/siddharthkumarsah/plastic-bottles-image-dataset.