Semantic SLAM - Detecting Objects in a 3D Map

Dev Dipak Ghiya (CB31C4) Fazil Khan (A7DEEE) Yaseen Mohemed (8C91A8)
Rahul Nunna (CEFBEF2)

1 Introduction

1.1 Goal and Purpose

The primary goal of this project is to enhance robotic navigation by integrating Simultaneous Localization and Mapping (SLAM) with object detection. We aimed to create a system that generates detailed 3D maps with semantic information, allowing robots to identify and understand their environment better. By using ORB-SLAM3 for mapping and YOLO for real-time object detection, we sought to improve both mapping accuracy and localization by adding object-awareness to traditional SLAM methods.

1.2 Motivation

Robots often struggle in dynamic environments where understanding the type and context of objects is as important as knowing their locations. For example, an autonomous vehicle must differentiate between static obstacles like buildings and dynamic ones like pedestrians. Traditional SLAM systems, while effective at creating maps, lack this ability to classify and interpret their surroundings. By integrating object detection with SLAM, we bridge this gap, allowing robots to navigate intelligently, avoid obstacles, and prioritize interactions with key objects. This project addresses these challenges, bringing us closer to smarter, more adaptive robotic systems. in

1.3 Background Information

Why SLAM Matters Simultaneous Localization and Mapping (SLAM) is a critical technology for autonomous robots, enabling them to navigate and map unknown environments. Over the years, SLAM has evolved from using expensive, sensor-heavy setups to more accessible, vision-based approaches that mimic human perception. This shift makes SLAM systems not only cost-effective but also versatile, suitable for a wide range of applications.

Advances in Vision-Based SLAM ORB-SLAM3 is a leading example of modern SLAM techniques. It is a robust, feature-based system capable of handling different types of cameras, including monocular, stereo, and RGB-D. It uses ORB (Oriented FAST and Rotated BRIEF) features for reliable tracking and mapping while employing advanced algorithms for data association and place recognition, making it highly accurate and efficient.

Adding Semantic Awareness While traditional SLAM methods focus on mapping obstacles and features, they lack the ability to understand what these features are. Incorporating object detection into SLAM adds a layer of semantic awareness, enabling robots to distinguish between different types of objects in their environment. This added intelligence allows for more informed decision-making, such as recognizing dynamic objects and avoiding collisions.

1.3.1 ORB-SLAM3

ORB-SLAM3 is a feature-based SLAM system designed to handle monocular, stereo, and RGB-D cameras. It combines multiple novel components to achieve highly accurate and robust real-time mapping and localization.

- **Feature Extraction and Matching:** ORB-SLAM3 uses ORB (Oriented FAST and Rotated BRIEF) features to identify key points in each frame. These features are tracked across frames, enabling pose estimation and map generation.
- **Data Association:** ORB-SLAM3 integrates short-term, mid-term, and long-term data associations:
  - Short-term matches elements visible in consecutive frames.
  - Mid-term matches keyframes in the same area to minimize drift.

  - Long-term associations, such as loop closures, correct errors over time and ensure global consistency.

- **Place Recognition and Relocalization:** It employs a robust DBoW2 (Bag of Words) algorithm for identifying previously visited locations, ensuring the system can recover if tracking is lost or return to mapped areas with high accuracy.
- **Optimization:** ORB-SLAM3 uses Maximum-a-Posteriori (MAP) estimation to perform bundle adjustment. This minimizes reprojection error for features across frames, ensuring the map is both accurate and consistent.
- **Real-Time Multi-Map Capability:** ORB-SLAM3 can handle tracking across disconnected maps. When revisiting previously mapped areas, it merges the maps seamlessly, ensuring accurate localization and continuous mapping.

These features make ORB-SLAM3 a robust choice for mapping environments ranging from small indoor spaces to large outdoor scenes, even under challenging conditions.

1.3.2 YOLOv7

YOLOv7 (You Only Look Once, version 7) is a real-time object detection algorithm that excels in speed and accuracy. It operates by dividing an image into a grid and predicting bounding boxes, object classes, and confidence scores for objects within each grid cell.

- **Architecture:** YOLOv7 is built on a convolutional neural network (CNN) backbone optimized for both speed and precision. It introduces dynamic label assignment and efficient decoupled heads for classification and localization tasks, improving detection accuracy without increasing computational load.
- **Real-Time Object Detection:** Each grid cell in the image predicts bounding boxes for objects it detects, along with class probabilities. The bounding boxes are refined through non-maximum suppression to eliminate redundant detections.

YOLOv7's speed and precision, combined with its ability to classify objects in diverse settings, make it an ideal tool for integrating semantic awareness into traditional SLAM systems.
By combining the strengths of ORB-SLAM3 and YOLO, we aim to create a system that not only maps its environment but also understands it, paving the way for more capable and autonomous robots.

2 Methods

Our approach integrates real-time monocular mapping with semantic labeling, combining ORB-SLAM3 for mapping and YOLOv7 for object detection and depth estimation. This section outlines the data, methods, and algorithms employed in our system.

2.1 Data Sources

The data used in our system is divided into two main categories: the dataset used for testing the SLAM system and a custom dataset used for training the YOLOv7 model. The SLAM testing dataset provides real-world sensor data to validate the mapping and localization process, while the custom semantic dataset is specifically designed for object detection and localization in a controlled environment.

2.1.1 SLAM Testing

The ORB-SLAM3 library was validated using NTU VIRAL's `eee_01.bag` dataset. This high-quality dataset provides real-world sensor data, enabling robust testing of mapping and localization under realistic conditions.

2.1.2 Custom Semantic Dataset

We trained a YOLOv7 model on a custom dataset comprising four types of wooden blocks:

- **Cube**: Class label `cube`
- **Pyramid**: Class label `triangle`
- **Arch**: Class label `arch`

- **Semicircle**: Class label `circle`

The dataset includes 1,088 images, divided into an 80/20 training/testing split. Images were annotated in YOLO format, which stores bounding boxes normalized to the range [0, 1]. Annotations for each image follow this structure:

```
class_id center_x center_y width height
```

A label map is used to translate numeric class IDs into human-readable labels during downstream processing.

2.2 Algorithms

This section outlines the steps and algorithms used to integrate real-time monocular mapping with semantic labeling. The system combines ORB-SLAM3 for mapping and YOLOv7 for object detection and depth estimation.

2.2.1 Video Streaming

Image data is acquired using a Raspberry Pi camera, which streams live video frames to a ROS computation graph. The images are published as `sensor_msgs/Image` messages to the `/camera/left/image_raw` topic.
Two methods are used for transferring images:

- **Real-time streaming:** Frames are sent directly to the processing pipeline via ROS nodes.
- **ROS bag files:** Pre-recorded `.bag` files store serialized ROS messages, providing consistent input for testing and analysis.

These methods ensure flexibility and reproducibility for system evaluation.

2.2.2 Training the YOLOv7 Model

Before integrating YOLOv7 for object detection, the model is trained on a custom dataset. The training is conducted using the following steps:

- The custom dataset, comprising 1,088 images of four types of wooden blocks (cube, pyramid, arch, and semicircle), is split into an 80/20 training/testing ratio.
- Images are annotated in YOLO format, with bounding boxes normalized to the range [0, 1].
- The model is trained using a Python script, adjusting hyperparameters like learning rate, batch size, and epochs. Monitoring tools like TensorBoard are used to track the loss and mean Average Precision (mAP) during training.

Once trained, the YOLOv7 model is ready for integration with the system to detect and localize objects in real-time.

2.2.3 SLAM Integration

ORB-SLAM3 processes the monocular image stream, generating a sparse point cloud map while estimating the camera's pose. This operates in three stages:

- **Tracking:** Detects and tracks keypoints in successive frames to estimate the camera's pose relative to the map.
- **Local Mapping:** Refines the map locally by optimizing the structure and adding new keyframes.
- **Loop Closing and Relocalization:** Ensures global map consistency by correcting loops and maintaining robustness in dynamic scenarios.

ORB-SLAM3 publishes the following outputs as ROS topics:

- `/orb_slam3/tracked_points`: Keypoints in the current sliding window, used for dynamic map updates.
- `/orb_slam3/all_points`: All mapped points, representing the global spatial structure.
- `/tf`: The camera's relative position and orientation (quaternions) in the world frame.

The quaternion data is converted into roll, pitch, and yaw values, which are later used for semantic mapping.

2.2.4 Preprocessing (Letterbox Resizing)

Before object detection, images are resized using the letterbox method to align with YOLOv7's input requirements. The method:

- **Maintains aspect ratio:** Resizes the image without distortion.
- **Adds padding:** Borders (e.g., gray) are added to achieve the required dimensions (e.g., 640×640 pixels).
- **Maps bounding boxes:** Padding and scaling factors are stored to remap bounding boxes to the original image scale, ensuring accurate object localization.

This step ensures that object detections remain aligned with the original scene geometry.

2.2.5 Object Detection and Depth Estimation

This section outlines the process of object detection and depth estimation in the system. Object detection is performed using the YOLOv7 model, which is integrated with the ROS framework. The system processes images from the camera, detects objects in real-time, and estimates their 3D depth using a pinhole camera model.

- **ROS Node Setup:** The ROS node subscribes to the `/left/image_raw` topic, receiving raw image frames. These frames are then converted into a format compatible with the YOLOv7 model using the `cv_bridge` package, which bridges the gap between ROS image messages and OpenCV image formats.
- **Image Preprocessing:** The images are preprocessed to match the input size expected by YOLOv7. This involves resizing and padding the images while maintaining the aspect ratio using the letterbox method.

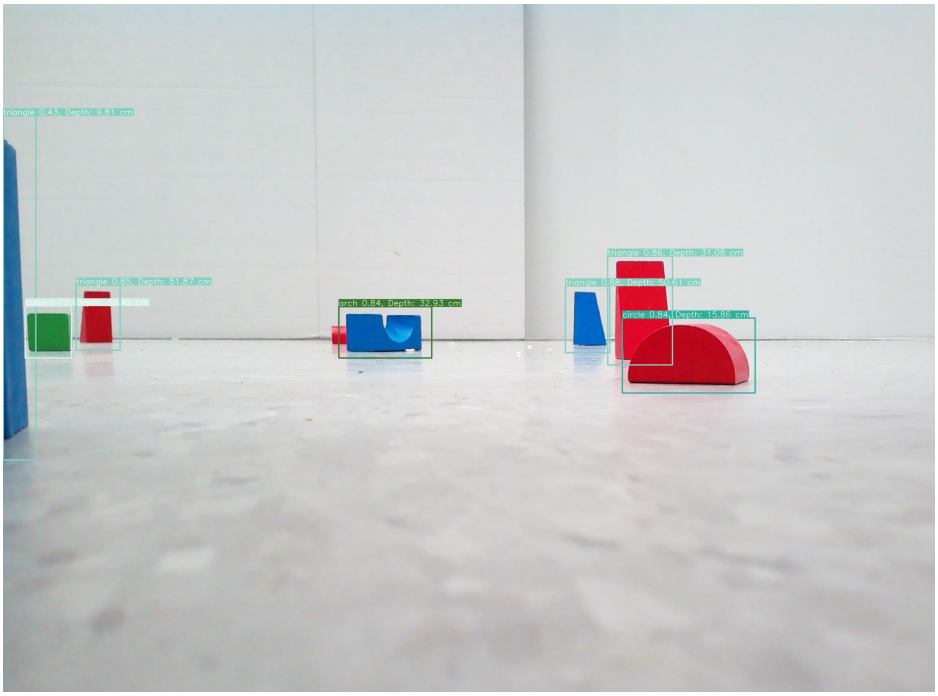


Figure 1: Sample image from YOLOv7 detection.

- **YOLOv7 Detection:** The preprocessed image is passed through the YOLOv7 model for object detection. The model uses non-maximum suppression (NMS) to filter overlapping bounding boxes and returns the most confident detections. Each detected object is classified, and bounding boxes are drawn around the detected objects.
- **Depth Estimation (Pinhole Camera Model):** Once the objects are detected, their real-world depth is estimated using the pinhole camera model. This is done by calculating the depth based on the known height of the objects and the height of the object in the image. Depth is estimated for detected objects using the pinhole camera model:

$$\text{Depth} = \frac{f \cdot h_{\text{real}}}{h_{\text{image}}}$$

Where:

- f: Focal length, extracted from the camera intrinsics matrix K.
- h_{real}: Known real-world object height (e.g., cube height = 3.0 cm).
- h_{image}: Height in image coordinates $(y_2 - y_1)$.

This calculation uses the intrinsic matrix K to ensure precise depth estimation based on camera parameters.

- **Semantic Mapping:** Semantic mapping integrates camera pose, object detection, and depth estimation to generate a 3D map enriched with object information:
  - **Camera Pose and Orientation Extraction:** The camera's relative position and quaternion data are obtained from the `/tf` topic. Quaternions are converted into roll, pitch, and yaw angles.
  - **Relative Vector Calculation:** Depth and the (X, Y) centroid of the detected bounding box are used to calculate the relative vector between the camera and the object in 3D space.

2.3 Software and Tools

- **Robot Operating System (ROS):** Facilitates communication between the SLAM and YOLO pipelines.
- **ORB-SLAM3:** Provides real-time monocular SLAM capabilities, including sparse point cloud generation and pose estimation.
- **YOLOv7:** Detects objects in real time, fine-tuned on a custom dataset tailored for controlled environments.
- **OpenCV:** Enables image preprocessing, visualization, and transformations.
- **PyTorch:** Powers the YOLOv7 inference for efficient real-time object detection.
- **MakeSense.ai:** A web-based annotation tool used to label the custom dataset for YOLOv7. Its user-friendly interface and support for YOLO-format annotations streamlined the data preparation process.

3 Results

3.1 YOLO Detection Accuracy

Table 1 shows the average accuracy of the YOLO model across each of the tested shape types. For each object class, the following metrics were calculated:

- Precision: $\frac{true\_positives}{true\_positives + false\_positives}$
- Recall: $\frac{true\_positives}{false\_positives}$

Class	Images	Labels	P	R	mAP@.5
all	101	184	0.887	0.943	0.957
cube	101	40	0.791	0.975	0.961
triangle	101	57	0.918	0.982	0.98
circle	101	40	0.864	0.949	0.921
arch	101	47	0.976	0.866	0.965

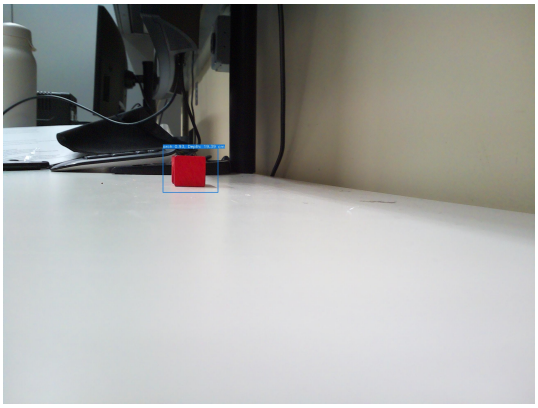
Table 1: YOLO performance across all object types

3.2 Distance Measurement Accuracy

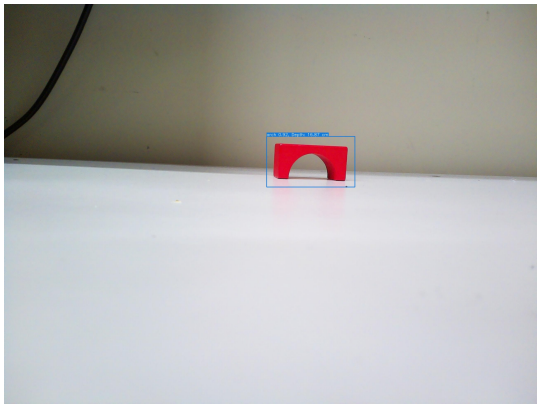
Visual distance measurements were compared against true measurements between camera and object. Average measurement error between ground truth data and visual distance measurements was **1.5cm**.

3.3 ORB-SLAM3/YOLO Combined Performance

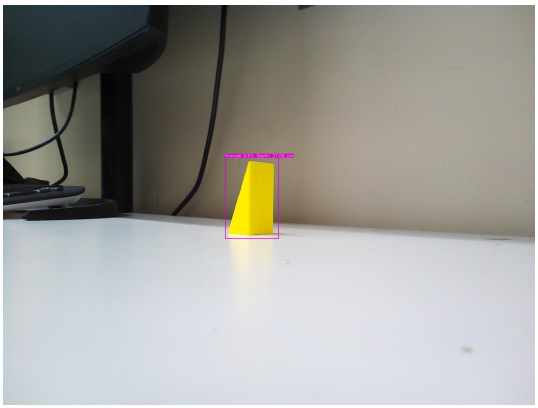
ORB-SLAM3 was run with YOLO in three different environments. Figure 2 shows the output of ORB-SLAM3 and YOLO in each environment. Table 2 shows the performance of ORB-SLAM3 and YOLO in each of the tested environments.



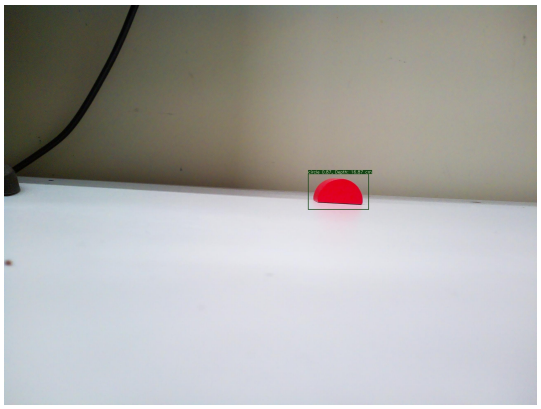
(a) Detection 1



(b) Detection 2



(c) Detection 3



(d) Detection 4

Figure 2: Object detections with measured distances.

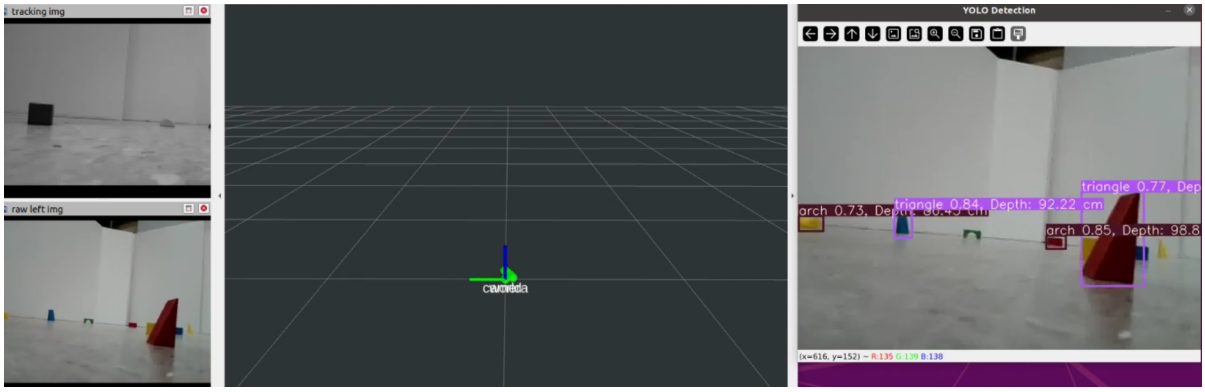
	Lab	**Living Room**	**Kitchen**
Lighting	Overhead, white	Uneven, white light	Even, overhead, yellow, shadows
Scene Density	Sparse	Dense	Dense
Detection/Depth Performance	High	Moderate	Low
Mapping Performance	Low	Moderate	High

Table 2: ORB-SLAM3/YOLO Performance across 3 different environments

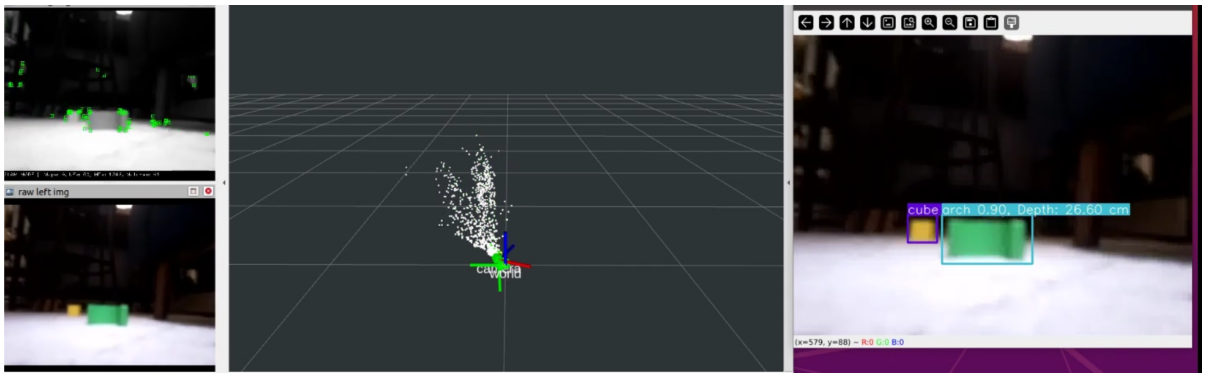
4 Discussion

It is apparent through our testing that environment plays a large role in the performance of both the SLAM algorithm and YOLO's CNN detection model. In the lab environment, obstacles were placed in front of a blank background, with little-to-no identifiable features. Due to this, ORB-SlAM3 was unable to effectively generate a map of the environment. However, in the denser kitchen and living room environments, ORB was able to successfully generate a map.

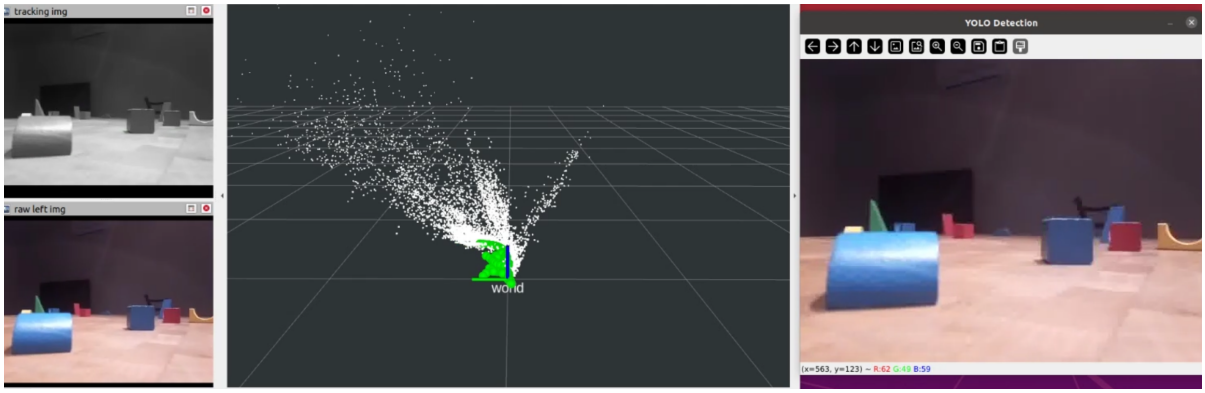
Conversely, YOLO's performance was heavily tied to lighting within the environment. As the entire shape image dataset was trained in the lab environment under even, white light, it is expected that the YOLO model would perform best in the same environment. In the living room environment, while the overhead light matched the color temperature of the lab, the lighting was far more uneven and contained more shadows, resulting in far fewer detections. In the kitchen environment, although the lighting was bright and overhead, the warmer color temperature and increased shadows made the shapes appear completely different to the model, resulting in minimal or no detections.



(a) Lab Environment



(b) Living Room Environment



(c) Kitchen Environment

5 Conclusion

We developed a map in various environments using ORB-SLAM3 integrated with ROS Noetic and object detection using YOLOv7. Based on the results, it is evident that the sparse model is incompatible with ORB-SLAM3 (as shown in the lab environment), as it requires more features/key points to generate an accurate 3D map. To build a more robust dataset, the YOLO model must also be trained under diverse conditions, such as various lighting scenarios, low lighting, different color tones, and shadows. To test this hypothesis, we tested our YOLO model in environments where it was not trained; based on our results, YOLO was able to detect objects but with false labels (as shown in Kitchen environment) and less label detection from ideal conditions.
Another point worth mentioning is that the map must be sufficiently dense (enough key points) to ensure accurate mapping. However, if the camera-equipped vehicle operates in an environment it has not been trained for, further deep-learning techniques may be necessary to adapt the model for training and testing purposes. Our model is trained in a stable/fixed environment and with limited shapes to detect, so it cannot be used in an outer environment.