THE UNIVERSITY of ADELAIDE

Towards Object-Based Visual SLAM: A Revolution for Urban Tram

A final report submitted for **COMP SCI 7205 Artificial Intelligence and Machine Learning Research Project** at The University of Adelaide

by

Phuong Quoc Anh To
The University of Adelaide
phuongquocanh.to@student.adelaide.edu.au

Supervisors

Dr. Mehdi Hosseinzadeh
The University of Adelaide
mehdi.hosseinzadeh@adelaide.edu.au

SCHOOL OF COMPUTER SCIENCE
THE UNIVERSITY OF ADELAIDE
TRIMESTER 3, 2025

Abstract

In today's world of AI and robotics, self-driving vehicles are no longer a futuristic dream. Their application to public transport, specifically urban tram networks, has become a crucial field of research for digital city development. The core mechanism of an autonomous vehicle is a well-established technique called Visual Simultaneous Localisation and Mapping (VSLAM), which is the process by which the vehicle uses the sensors and images to build environmental maps, and also determine its localised position in the mapping of the environment. However, Visual SLAM frameworks specifically developed for urban tram operations often degrade in specific conditions, demanding precise mapping, high localisation accuracy, and robustness against obstructions. Throughout this work, we introduce an advanced Object-Based Visual SLAM system, which is developed to address the complexity introduced by dynamic obstacles (e.g., vehicles, pedestrians) along with infrastructure components (e.g., traffic lights, stations). Firstly, we leverage deep learning-based techniques (i.e., YOLOv8 model) to significantly improve localisation accuracy. Secondly, we extract 2D bounding boxes from KITTI odometry sequence 08 and convert them into oriented 3D cuboids using CubeSLAM technique. Finally, we integrate these 3D cuboid coordinates into ORB-SLAM3 architecture and we execute its monocular running on KITTI odometry sequence 08 for frame-by-frame visualisation alongside point clouds and poses. Our proposed solution seeks to not only push the boundary of SLAM research but also to facilitate safer, more efficient, and dependable AI-based tram systems in real-world transport.

Keywords: Visual SLAM; SLAM; Object Detection; YOLOv8; Cross-Dataset Generalisation; ORB-SLAM3; CubeSLAM; Multi-Object Tracking; ByteTrack

1. Introduction

Across Europe's vibrant urban centres—and in Australian metropolises like Sydney and Melbourne, where tram lines criss-cross dense city corridors—a major transformation is underway. Rapid developments in Artificial Intelligence (AI) are allowing cities around the world to embed autonomous technology in their tram networks, changing public transport. In the optimistic outlooks surrounding autonomously operated vehicles (AVs) with regard to the social and economic benefits, a report from the UK Government shows they could also add £42 billion to the economy, along with generating approximately 38,000 jobs by 2035 (Financial Times, 2024). With respect to this, the integration of autonomous vehicle technology into urban public transportation systems represents a significant advancement in smart city infrastructure. In the past, the traditional outdoor localisation methods mainly rely on the Global Positioning System (GPS) or Real-Time Kinematic (RTK) positioning system (Siegwart et al., 2011). Standard GPS presents well-known challenges for precision applications. Multipath reflections from nearby structures, intermittent satellite visibility, and ionospheric delays combine to produce positioning errors that frequently exceed several metres—rendering the technology unreliable in urban canyons, tunnelled sections, or areas with heavy tree covers (Li et al., 2024; Hussain et al., 2021). RTK techniques can theoretically provide a higher degree of accuracy via carrier-phase ambiguities against corrections from a local base station. In practice, however,

RTK shares GPS's susceptibility to signal obstruction, depends on maintaining a continuous datalink to the base, and degrades significantly when operating more than 10–20 kilometres from the reference receiver (Wang et al., 2024; Man et al., 2025). Therefore, the Simultaneous Localisation and Mapping (SLAM) solution is considered a cornerstone technology for autonomous systems, enabling vehicles to navigate and map unknown environments concurrently. Durrant-Whyte and Bailey (2006) state that the SLAM method is widely adopted as an effective solution for the localisation in autonomous driving vehicles. Therefore, our target is to investigate the SLAM approach and to take advantage of this approach to develop effective AI-driven urban tram systems. SLAM can be broadly categorised into these types: Light Detection and Ranging (LIDAR) SLAM, Multi-Sensor SLAM, and Visual SLAM. In comparison, Visual SLAM has the advantages of rich information and easy-to-install, as well as making the system less expensive and lightweight (Cheng et al., 2022). In our work, we plan to explore the current landscape of Visual SLAM research, with particular emphasis on object-based approaches.

2. Foundation

2.1 Principles of SLAM

The origins of **Simultaneous Localisation and Mapping (SLAM)** were established at the 1986 IEEE Robotics and Automation Conference in San Francisco. It was described as dual operations, including two significant tasks: localisation and simultaneous mapping of an unknown environment using sensory data (Durrant-Whyte et al., 1996; Durrant-Whyte and Bailey, 2006).
As claimed in the work of Durrant-Whyte and Bailey (2006), the core functionality of SLAM allows a mobile robot to construct an environmental map and at the same time use this map to infer its own location. The general probabilistic form of SLAM is typically represented in this formula:

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{1}$$

where

- $\mathbf{X}_{0:k} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_k\} = \{\mathbf{X}_{0:k-1}, \mathbf{x}_k\}$: The history of vehicle locations.
- $\mathbf{U}_{0:k} = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k\} = \{\mathbf{U}_{0:k-1}, \mathbf{u}_k\}$: The history of control inputs.
- $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_n\}$: The set of all landmarks.
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_k\} = \{\mathbf{Z}_{0:k-1}, \mathbf{z}_k\}$: The set of all landmark observations.

This probabilistic modeling lays the foundation of estimating both the environment (mapping) and the robot's position within it (localisation) simultaneously (Durrant-Whyte and Bailey, 2006). The core dynamics of SLAM problem are illustrated in Fig. 1, depicting how a mobile agent infers its movement and the landmark positions using control inputs and observations over time. This schematic underscores the relationships between estimated and ground truth trajectories, demonstrating the inherent difficulty of preserving precise localisation and mapping when it operates in dynamic settings.

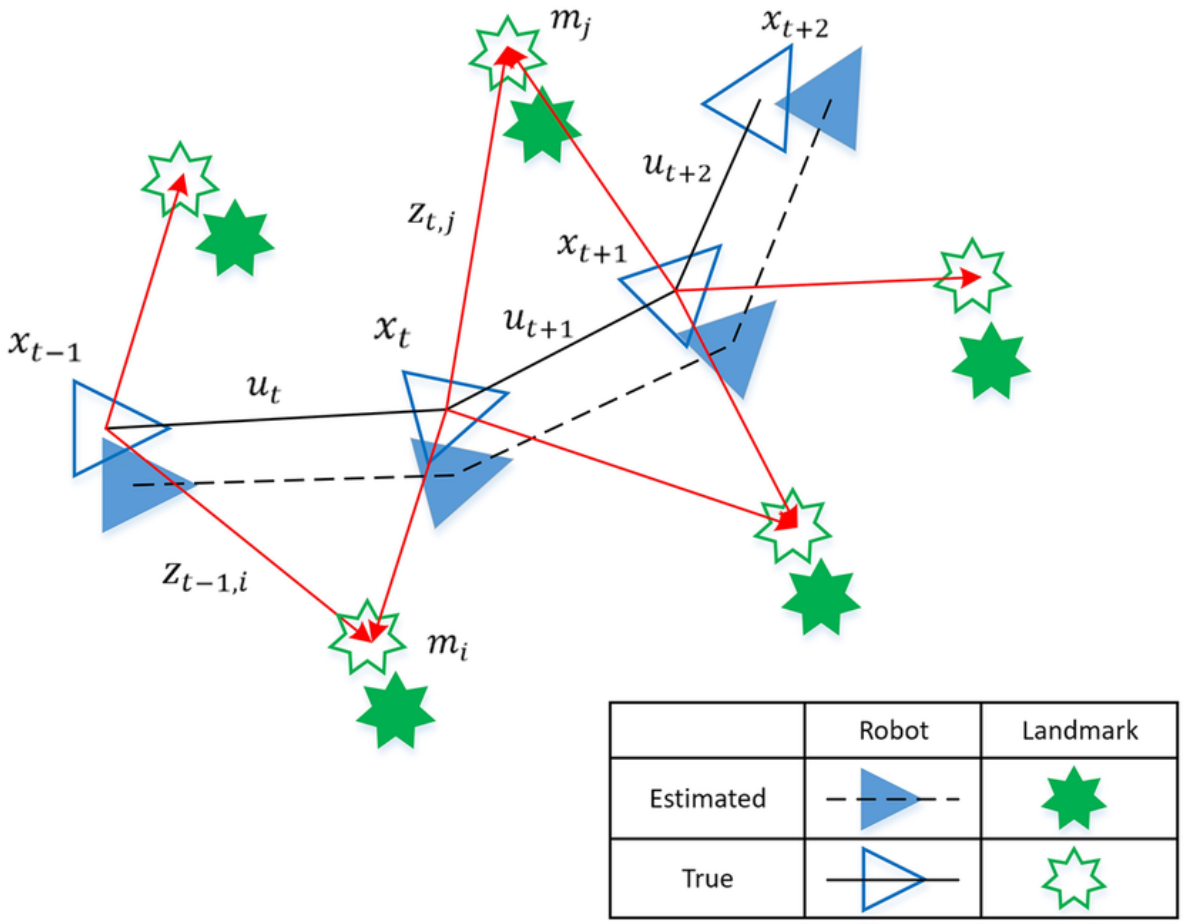


Figure 1: Schematic representation of SLAM process

2.2 Principles of Visual SLAM

Visual Simultaneous Localisation and Mapping (VSLAM) is a prominent branch of SLAM that depends on vision sensor data while utilising projective geometry within its observation model (Liu, 2020). In simpler words, when a SLAM system uses vision sensors to collect information about the sensed surroundings, the system is referred to as Visual SLAM.

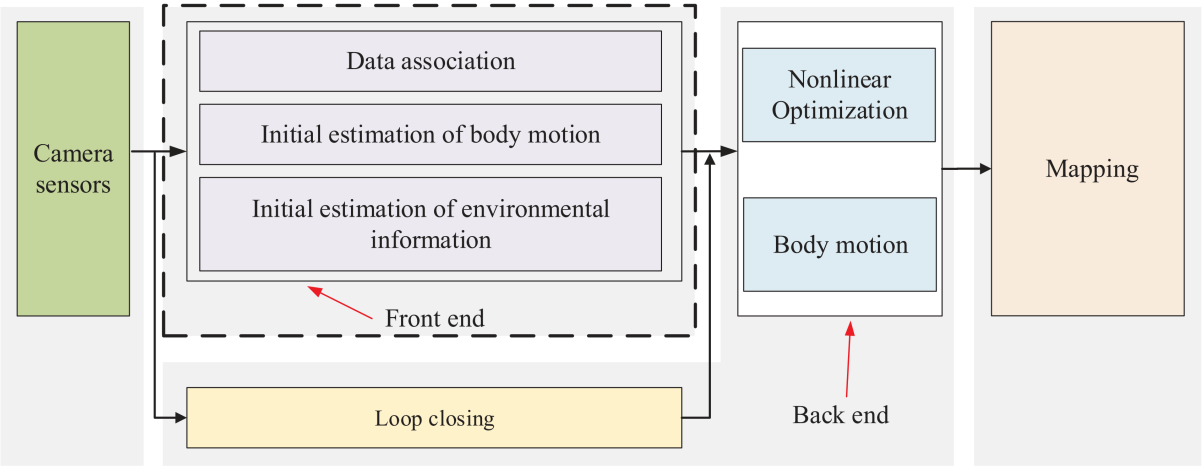


Figure 2: Visual SLAM components

Fig. 2 illustrates the typical architecture of Visual SLAM, categorised into five main parts: Sensor data module, front-end module, back-end module, loop closure module, and mapping module. In particular, according to Zhang (2022), sensor data is collected from the camera frame and passed to the front-end (or Visual Odometry) component, which handles feature extraction and data association between consecutive frames to achieve the initial local mapping and camera motion estimation. The back-end module is responsible for the numerical optimisation of the front-end and further motion estimation. The loop closure module eliminates accumulated drifts by calculating image similarity in a large-scale environment. If a loop is detected, it will provide information to the back-end part for further optimisation. Finally, the mapping module reconstructs the surrounding environment based on the estimated camera trajectory.

3. Literature Review

We summarise several advanced studies in VSLAM methods, with emphasis on systems that embed semantic segmentation into a SLAM pipeline for dynamic and cluttered surroundings. In specific, we review two foundational branches of conventional VSLAM architectures: feature-based and direct approaches. These systems lay the foundation for the emergence of specialised object-based techniques. Basically, recent VSLAM research revolves around several significant themes:

3.1 Feature-based VSLAM: Landmark Detection and Matching

Feature-based methods target unique image areas with distinctive characteristics. These features appear at several abstraction levels, including: low-level (e.g., corners and

edges), mid-level (e.g., super-pixels), and high-level (e.g., semantically labeled objects) across image frames. Repeatability is the principal criterion, which ensures the consistent detection of the similar feature in images recorded from varied viewpoints (Azzam et al., 2020).

VINS-Mono (2018): This system demonstrates robust visual-inertial state estimation by effectively integrating visual and inertial data, resulting in minimal drift. VINS-Mono shows superior accuracy and robustness, especially with loop closure engaged (Qin et al., 2018).

MISD-SLAM (2022): The MISD-SLAM system completes the static background reconstruction that builds upon the backbone of ORB-SLAM3 framework. In addition, MISD-SLAM chooses to remove the detected dynamic feature points, which reduces the influence of dynamic environment on the system and further improves the robustness of the system (You et al., 2022). This is worth noting because it emphasises the need for achieving a robust and high accuracy model that is suitable for our developing of autonomous tram network.

3.2 Direct Visual SLAM: Photometric Optimisation Techniques

Direct approaches determine motion and structure by directly examining photometric errors at pixel scale across images. In contrast to **feature-based (indirect) methods**, which depend solely on a sparse set of different features, the direct methods minimise an error derived directly from *raw pixel intensities*, ranging from brightness-based cross-correlation to image brightness, providing an in-depth analysis of the visual data (Irani and Anandan, 1999).

DSO (2018): Engel et al. (2018) introduced a new advancement in direct SLAM with Direct Sparse Odometry (DSO), a direct method with a sparse map. It combines a fully direct probabilistic model (minimising a photometric error) with consistent, joint optimisation of all model parameters, including geometry—represented as inverse depth in a reference frame—and camera motion. However, since DSO models photometric noise, its performance quickly deteriorates with added geometric noise, whereas ORB-SLAM is much less affected.

NICE-SLAM (2022): Zhu et al. (2022) presented NICE-SLAM, a dense VSLAM architecture that constructs layered spatial information by introducing a hierarchical scene representation. Optimising this representation with pre-trained geometric priors enables detailed reconstruction on large indoor scenes. While NICE-SLAM offers promising results, several limitations still remain, especially in computational efficiency for real-time tram applications. These gaps are relevant to urban tram contexts, including untested rail-based applications and real-time challenges, demand suitable solutions.

Contribution. Compared to existing works, which focus solely on direct or feature-based approaches, and/or single-dataset evaluation, our study hope to contribute a novel cross-dataset evaluation with 3D temporal tracking for monocular Visual SLAM.

4. Methodology

This section provides insight into our Object-Based VSLAM pipeline. We classify our methodology into four consecutive phases to systematically achieve our targets identified earlier.

- **Stage 1: Improved Localisation Accuracy in dynamic urban areas**
  - *Description*: In the first stage, the dynamic obstacles such as vehicles or pedestrians must be detected. We employ the YOLO detector for this purpose to enhance localisation accuracy (Bescós et al., 2018). We also examine the cross-dataset generalisation performance of the YOLO model.
- **Stage 2: Multi-Object Tracking and 3D Cuboid Generation**
  - *Description*: In this stage, it is divided into three submodules. The 2D bounding boxes coordinates are extracted from KITTI odometry sequence 08 using best weight from YOLO model. Subsequently, we integrate the ByteTrack Multi-Object Tracking algorithm between the detection and 3D reconstruction stages. Finally, in order to get 3D reconstruction, our approach employs the CubeSLAM technique, which first generates 3D cuboid proposals from 2D detections, then refine them using multi-view constraints (Yang and Scherer, 2019).
- **Stage 3: SLAM Integration**
  - *Description*: Subsequently, the 3D object cuboids generated from Stage 2 are merged into ORB-SLAM3 framework running in monocular mode on KITTI sequence, utilising Docker container.
- **Stage 4: SLAM Evaluation**
  - *Description*: To assess the performance of the integrated object-based VSLAM system, we implement a comprehensive evaluation using standard metrics commonly used in SLAM research.

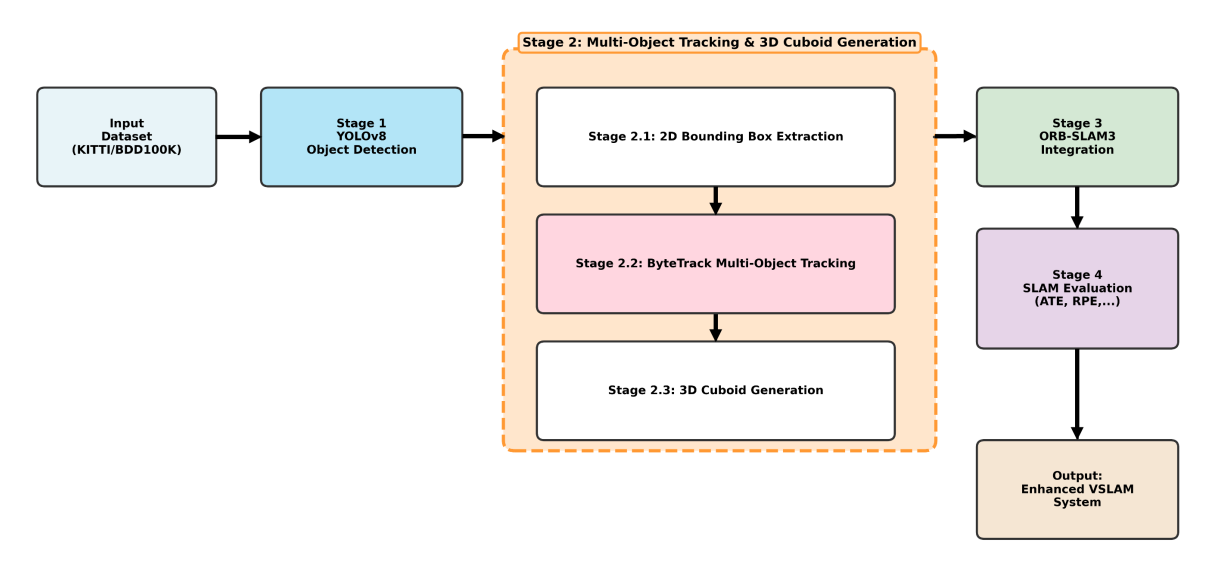


Figure 3: Object-Based Visual SLAM Pipeline

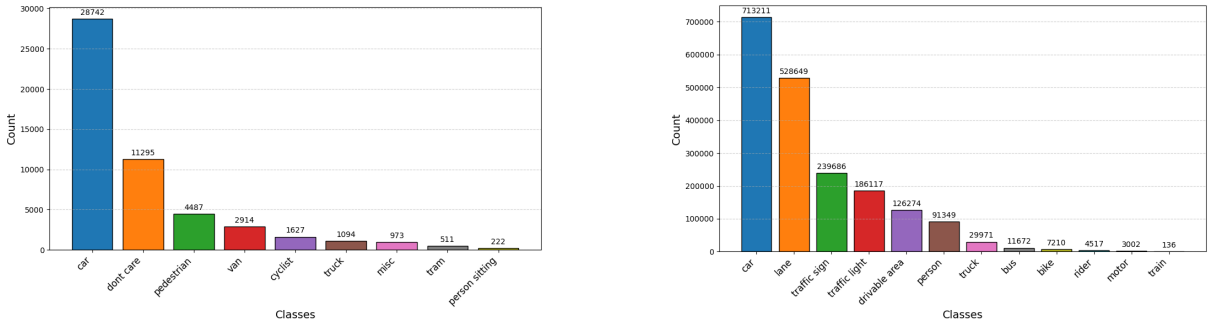
Stage 1: Improved Localisation Accuracy in dynamic urban areas

In this stage, we focus on the integration of deep-learning-based approaches to enhance the localisation accuracy of VSLAM for autonomous trams in dynamic urban environments. Specifically, we adopt an empirical approach using YOLOv8 for object detection, integrating it as a frontend to VSLAM, and we investigate cross-dataset generalisation performance of its variants.

1.1: Datasets

To assess in-domain and cross-dataset generalisation, we used:

- **BDD100K:** The Driving Data Set (BDD100K) includes 100K images/videos of diverse driving scenarios such as day/night and rain/fog with labels for 10 classes (cars, people, traffic lights). This dataset is an excellent source of information regarding the variability of environmental conditions (Yu et al., 2020).
- **KITTI Object Detection Benchmark:** This dataset is a branch of KITTI Vision Benchmark Suite that provides calibrated datasets and benchmarks for examining object detection performance in self-driving cases (Geiger et al., 2012). The dataset includes 7,481 and 7,518 images for training and testing set, respectively, captured from urban drives, with primary concentration on classes such as cars, pedestrians.



(a) KITTI (b) BDD100K

Figure 4: Distribution of Object Types in both dataset

1.2: Conversion to YOLO format

Initially, annotation data is stored in `.json` files, before initiating the training phase for an object detection system, it is essential to transform these annotations into the YOLO format suitable for both `train/val` sets. In our approach, we adhere to the conventional Ultralytics YOLO structure, while also offering detailed explanations of its organisation and the steps involved, drawing from actual datasets. Specifically, the conversion yields `*.txt` that align with the corresponding image filenames. In addition, each generated `*.txt` file contains entries formatted data in this pattern `(class_id x_center y_center width height)`, where bounding box coordinates are scaled to a range between 0 and 1 (Ultralytics, 2024). These adjusted coordinates are expressed mathematically as:

$$label = (x, y, w, h) \in \mathbb{R}^4 \tag{2}$$

where

- **x**: Denotes the x-coordinate of bounding box centre (x_{center}).
- **y**: Denotes the y-coordinate of the bounding box centre (y_{center}).
- **w**: Indicates the normalised width of the bounding box.
- **h**: Indicates the normalised height of the bounding box.

With these coordinate definitions, the computation for each component proceeds as follows:

- **Center Coordinates:** The values for x_{center} and y_{center} are derived by taking the average of the respective minimum and maximum x and y values, followed by normalisation relative to the image's total width and height. This can be formulated as:

$$x_{\text{center}} = \frac{x_{\min} + x_{\max}}{2 \times \text{width}}, \quad y_{\text{center}} = \frac{y_{\min} + y_{\max}}{2 \times \text{height}}$$

- **The bounding box's dimensions:** The width and height are established by calculating the span between the maximum and minimum coordinates, then normalising these differences against the image dimensions. The expressions are:

$$\text{width} = \frac{x_{\max} - x_{\min}}{\text{width}}, \quad \text{height} = \frac{y_{\max} - y_{\min}}{\text{height}}$$

- **Class ID:** Lastly, the class identifier is obtained by consulting a predetermined mapping of object categories to numerical values.

Table 1: Class Mappings for Object Types in BDD100K and KITTI

(a) BDD100K

Object Type	**Class ID**
car	0
lane	1
traffic sign	2
traffic light	3
drivable area	4
person	5
truck	6
bus	7
bike	8
rider	9
motor	10
train	11

(b) KITTI

Object Type	**Class ID**
car	0
van	6
truck	6
pedestrian	5
person sitting	5
cyclist	8
tram	11

1.3: Cross-Dataset Evaluation Strategy

Our approach deliberately separated training (BDD100K) and testing (KITTI) datasets to assess real-world generalisation capability. Given our limited environmental constraint, we decided to adopt the deep learning-based object detection model YOLOv8, while the latest version at the moment is YOLO26 (released in September 2025) (Ultralytics, 2025). In specific, we conduct two experiments utilising the nano (YOLOv8n) and largest (YOLOv8x) versions:

- **YOLOv8n:**
  - **Experiment 1 (In-Domain):** Trained on BDD100K, evaluated on BDD100K.
  - **Experiment 2 (Cross-Dataset Generalisation):** Trained on BDD100K, evaluated on KITTI.
- **YOLOv8x:**
  - **Experiment 1 (In-Domain):** Trained on BDD100K, evaluated on BDD100K.
  - **Experiment 2 (Cross-Dataset Generalisation):** Trained on BDD100K, evaluated on KITTI.

As a result, after implementing both experiments to two YOLOv8 variants, we acquire empirical data necessary for a comprehensive analysis of performance and generalisation across different model scale. This evaluation covers the models' performance (mAP50 metrics) and computational overheads (GFLOPs and time). A detailed summary of the key performance and efficiency metrics for both the in-domain (Exp1) and cross-dataset generalisation (Exp2) trials is concisely presented in Table 2.

Table 2: Performance Comparison Between YOLOv8n and YOLOv8x

Experiment	Model	Params (M)	GFLOPs	Dataset (Train/Eval)	mAP50 (%)	Time (h)
Exp1	YOLOv8n	3.2	8.1	**BDD100K** ↓ **BDD100K**	35	1.775
	YOLOv8x	68.2	257.4	**BDD100K** ↓ **BDD100K**	45	8.301
Exp2	YOLOv8n	3.2	8.1	**BDD100K** ↓ **KITTI**	25	1.260
	YOLOv8x	68.2	257.4	**BDD100K** ↓ **KITTI**	40	6.800

Our empirical findings emphasise the typical Accuracy-Efficiency trade-offs concept faced by deep learning models. It is clear from the Table 2 that the largest version (YOLOv8x)

results in superior accuracy compared to its lower version (YOLOv8n); yet it requires considerable computational resources. In this work, we employ the optimal weight from Exp2's YOLOv8x for the continuing pipeline despite the lower `mAP50` score in Exp2 (**40%**) in comparison with Exp1 (**45%**). The reason for our selection lies in the cross-dataset evaluation framework, which provides opportunity for the model to be generalised across diverse urban environments. Moreover, Exp2's weights ensure more reliable object detection in subsequent pipeline stages, such as 3D cuboid generation from KITTI sequences, fostering robust VSLAM performance in heterogeneous urban settings.

Stage 2: Multi-Object Tracking and 3D Cuboid Generation

2.1: 2D Bounding Boxes Extraction

In this subsection, we outline the extraction of 2D bounding boxes from KITTI odometry sequences to serve as the input for subsequent tracking and 3D reconstruction. As mentioned earlier, we utilise the best training epoch weights from Experiment 2 (as shown in Table 2) to perform inference on the KITTI image frames. In particular, we choose the KITTI sequence 08 as our primary sequence throughout our pipelines due to its dynamic urban driving scenarios with diversified obstacles.
For each frame, inference is run with a confidence threshold of `0.6` and an Intersection over Union (IoU) threshold of `0.45`. After getting detection results as per evaluation, the bounding boxes are output into `x_min, y_min, x_max, y_max` format, along with associated class labels and confidence scores. As a result, the extracted data is serialised and saved as `.json` files, with each file named corresponding to the image frame (e.g., `000001.json`). Consequently, we have a total of 4071 `.json` files corresponding to 4071 frames of KITTI sequence 08.

2.2: ByteTrack Multi-Object Tracking

In terms of frame-by-frame detection, we recognise the importance of the data association and Multi-Object Tracking (MOT) problem in SLAM applications. The issue is that each frame produces independent detections and the objects are not being tracked or associated across frames. To maintain the object identity across frames, we integrate ByteTrack (a lightweight, SOTA Multi-Object Tracking algorithm) between the detection and 3D reconstruction stages.
Following the acquisition of 2D bounding box detections, a critical issue emerged, which was recognised as a result of our preliminary experimentation. Without applying any tracking mechanism, the system treats detections in every frame as new, independent object observations, regardless of whether it represents the same physical characteristic from the previous frame.
Unlike the full ByteTrack algorithm, which incorporates a Kalman-filter-based motion model (Zhang et al., 2022), our approach uses a simplified, non-parametric tracking strategy. In our pipeline, we apply ByteTrack's two-stage matching (high-conf, low-conf):

- High-confidence detections (above $\tau_h = 0.6$) are matched to existing tracks using the Hungarian algorithm with an IoU-based cost matrix.
- Unmatched tracks then receive a second association attempt with low-confidence detections (between $\tau_l = 0.1$ and τ_h), which often represent partially occluded or visually degraded objects.

For data association, the Intersection over Union (IoU) between a detection $\mathcal{B}_j$ and an existing track's most recent bounding box $\mathcal{B}_i$ is computed as:

$$\text{IoU}(\mathcal{B}_i, \mathcal{B}_j) = \frac{\text{Area}(\mathcal{B}_i \cap \mathcal{B}_j)}{\text{Area}(\mathcal{B}_i \cup \mathcal{B}_j)} \tag{3}$$

The outcome is a set of stable 2D tracklets, each associated with a persistent ID. These tracklets form the foundation for subsequent 3D cuboid reconstruction and long-term SLAM data association.

2.3: 3D Cuboids Generation

To reconstruct full 3D cuboids for detected objects in KITTI sequence 08, we combine camera calibration, object size priors, and SLAM pose data. Our approach aligns with **CubeSLAM** techniques, which first generate 3D cuboid proposals from 2D detections, then refine them using multi-view constraints. We estimate each object's depth (distance) from a monocular image using its 2D bounding box height (assuming average real-world object height), and estimate the object's orientation by fitting the projected cuboid to the 2D box width. With known camera poses from SLAM, we transform each cuboid to world coordinates for integration into ORB-SLAM3. This yields a 6-DoF pose (3D position + rotation) and dimensions for each object, consistent with ORB-SLAM3's world frame.

CubeSLAM Algorithm Overview

The CubeSLAM framework (Yang and Scherer, 2019) consists of two main components: (1) single-image 3D cuboid proposal generation using vanishing points, and (2) multi-view bundle adjustment that jointly optimises camera poses, object poses, and feature points. Below we summarise the key algorithms that form the backbone of this approach.

3D Cuboid Representation A 3D cuboid object $\mathcal{O}$ is represented by 9 degrees of freedom:

$$\mathcal{O} = \{\mathbf{T}_o, \mathbf{d}\} = \{[\mathbf{R} \mid \mathbf{t}], [d_x, d_y, d_z]^T\} \tag{4}$$

where $\mathbf{T}_o \in SE(3)$ is the 6-DoF pose (rotation and translation), and $\mathbf{d} \in \mathbb{R}^3$ represents the cuboid dimensions.

Proposal Scoring Function Each cuboid proposal is scored using:

$$E(\mathcal{O}|I) = \phi_{\text{dist}}(\mathcal{O}, I) + w_1\phi_{\text{angle}}(\mathcal{O}, I) + w_2\phi_{\text{shape}}(\mathcal{O}) \tag{5}$$

where ϕ_{dist} measures alignment with image edges via Chamfer distance, ϕ_{angle} evaluates consistency between line segments and vanishing points:

$$\phi_{\text{angle}} = \sum_{i=1}^{3} \left(\|\langle l_i^{ms}, l_i^{mt} \rangle - \langle VP_i, l_i^{mt} \rangle\| + \|\langle l_i^{ns}, l_i^{nt} \rangle - \langle VP_i, l_i^{nt} \rangle\| \right) \tag{6}$$

and ϕ_{shape} penalises unrealistic aspect ratios.

Inspired by the high-level idea of CubeSLAM, in our Python codebase, we implement a simplified geometric approximation of its 3D cuboid reasoning module. Our approach applies pinhole projection and class-specific object priors, and yaw angle to recover a plausible 3D pose and shape for each detected object. Finally, the 2D detections are transformed into 3D cuboids for each frame, stored as `.json` files, ready to be injected into ORB-SLAM3 system.

Algorithm 1 3D Cuboid Proposal Generation

Require: 2D bounding box B, image I, camera intrinsics K, vanishing points VP_i $(i = 1, 2, 3)$, ground plane normal $\mathbf{n}$ and distance m (if ground object)
Ensure: Set of 3D cuboid proposals $\mathcal{O} = \{[\mathbf{R}, \mathbf{t}, \mathbf{d}]\}$
Detect 2D corners of B: $p_1, \ldots, p_8$
for each VP_i **do**
    Compute 2D cuboid corners using line intersections
    $p_2 = (VP_1, p_1) \times (B, C)$, $p_4 = \ldots$, $p_3 = (VP_1, p_4) \times (VP_2, p_2)$, etc.
end for
if object has arbitrary pose **then**
    Use PnP solver to estimate $\mathbf{t}, \mathbf{d}$ up to scale
    $p_i = \pi(\mathbf{R}[\pm d_x, \pm d_y, \pm d_z]^T/2 + \mathbf{t})$
    Solve for $\mathbf{t}, \mathbf{d}$ using four adjacent corners (e.g., 1,2,4,7)
end if
if object is on ground **then**
    Back-project corners to 3D ground plane
    $\mathbf{P}_5 = \frac{-m}{\mathbf{n}^T(\mathbf{K}^{-1}p_5)}(\mathbf{K}^{-1}p_5)$
    Compute other vertical corners and determine scale from camera height
end if
Sample rotation $\mathbf{R}$ (or yaw for ground objects) to compute $VP_i = K\mathbf{R}_{\text{col}(i)}$
Generate proposals by varying $\mathbf{R}$, compute 2D projections, retain those fitting B
Score proposals using edge alignment and shape constraints
Select top-ranked proposals $\mathcal{O}$

Figure 5

Algorithm 2 Multi-View Bundle Adjustment with Objects

Require: Camera poses $\mathcal{C} = \{\mathbf{C}_i\}$, cuboids $\mathcal{O} = \{\mathcal{O}_j\}$, points $\mathcal{P} = \{\mathbf{P}_k\}$, measurements $e(c, o), e(c, p), e(o, p)$
Ensure: Optimised $\mathcal{C}^*, \mathcal{O}^*, \mathcal{P}^*$
Formulate as nonlinear least squares
$\min_{\mathcal{C},\mathcal{O},\mathcal{P}} \sum_{i,j,k} \left(\|e(\mathbf{c}_i, \mathbf{o}_j)\|^2_{\Sigma_{ij}} + \|e(\mathbf{c}_i, \mathbf{p}_k)\|^2_{\Sigma_{ik}} + \|e(\mathbf{o}_j, \mathbf{p}_k)\|^2_{\Sigma_{jk}} \right)$
Define errors
$e^{3D}_{co} = [\log((T_c^{-1}T_o)T_{om}^{-1})^{\mathfrak{se}(3)}_{\vee}; se_3(d - d_m)]$
$e^{2D}_{co} = [c, s] - [c_m, s_m]$ where c, s are projected box center/size
$e_{op} = \max(|T_o^{-1}P| - d_m, \mathbf{0})$
$e_{cp} = \pi(T_c^{-1}P) - z_m$
Solve using Gauss-Newton or Levenberg-Marquardt (e.g., g2o)
Apply Huber robust cost to all errors
Weight measurements by detection probability p and distance d

Figure 6

Stage 3: ORB-SLAM3 Integration

At this stage, we already have clean, non-overlapping 3D cuboid representations available from Stage 2. Our next goal is to feed these 3D coordinates into the ORB-SLAM3 framework running in monocular mode on KITTI sequence 08. In particular, we divide Stage 3 into distinct substages.

3.1: ORB-SLAM3 - Background and Architecture

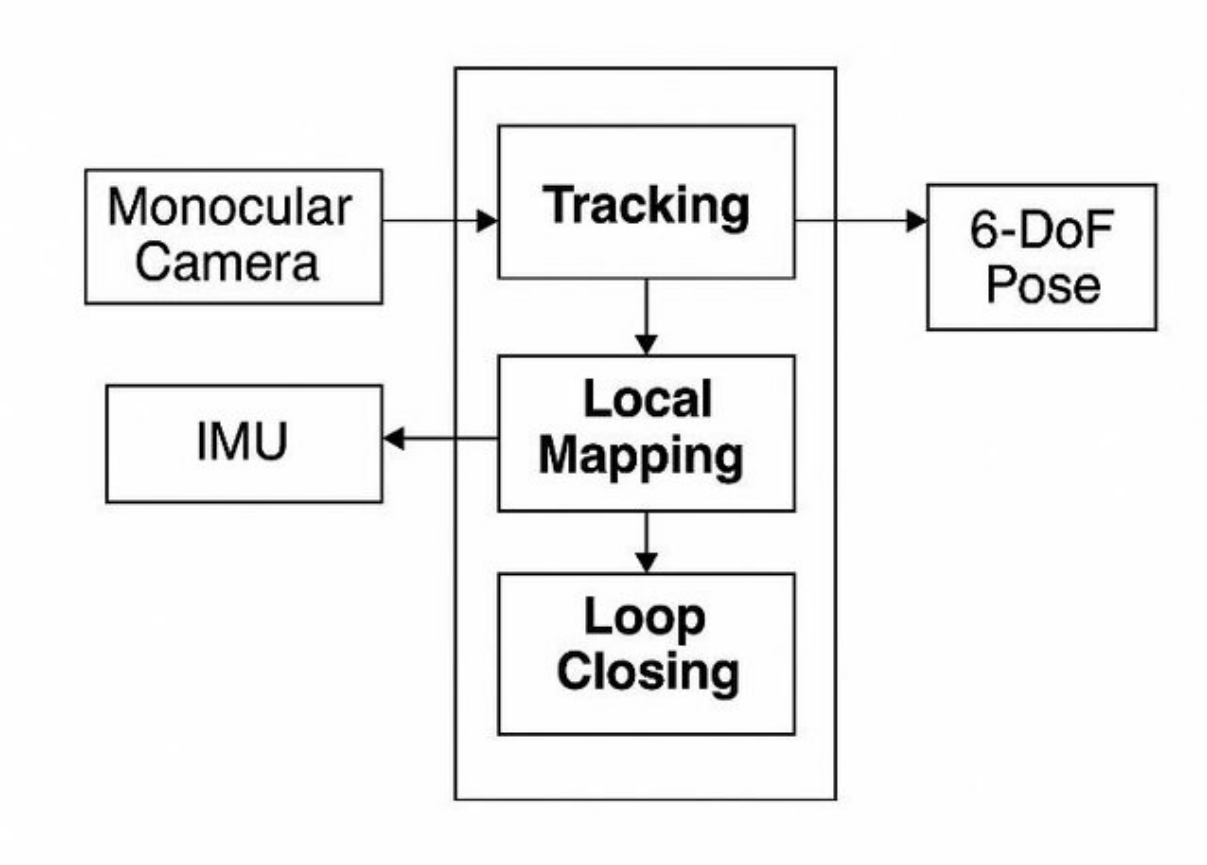


Figure 7: ORB-SLAM3 System Architecture

ORB-SLAM3 is currently recognised as the latest official version of the ORB-SLAM series. Introduced in the work of Campos et al. (2021), ORB-SLAM3 has shown remarkable improvements in computational efficiency while maintaining accuracy across visual, visual-inertial, and multi-map SLAM with monocular, stereo, and RGB-D cameras. In this work, we select ORB-SLAM3's monocular configuration for our urban tram application.
As illustrated in Fig. 7, the ORB-SLAM3 architecture comprises three core parallel threads: tracking, local mapping, and loop closing. The tracking thread processes each frame to localise the camera by finding feature correspondences with the local map. The local mapping thread manages keyframes and performs local bundle adjustment to refine the map structure. Finally, the loop closing thread detects revisited areas and corrects accumulated drift through pose graph optimisation (Mur-Artal and Tardós, 2017). In summary, the Algorithm in Fig. 8 covers the primary components of ORB-SLAM3 system.

Algorithm 3 ORB-SLAM3 Multi-Threaded Architecture

Require: Input frame I_t
Ensure: Camera pose T_t, updated map $\mathcal{M}$
Thread 1 - Tracking:
  Feature extraction → Feature matching → Pose estimation (PnP)
  Motion-only bundle adjustment → Decide if keyframe needed
Thread 2 - Local Mapping:
  Process new keyframes → Create new map points (triangulation)
  Local BA optimisation → Keyframe/point culling
Thread 3 - Loop Closing:
  Loop detection (BoW) → Loop validation (Sim(3))
  Pose graph optimisation → Global map correction
return T_t, $\mathcal{M}$

Figure 8

3.2: Docker-Based Development Environment

Rationale for Containerisation. Our working environment for the entire pipeline is based on an M1 MacBook Pro with 16 Gigabytes of RAM memory. During our initial attempts to configure ORB-SLAM3 natively on macOS, we faced a critical issue regarding the compatibility of the Pangolin visualisation library. We discovered that Pangolin often fails (`NSInternalInconsistencyException` error in Fig. 9) because macOS requires all UI calls on the main thread (Apple Inc., 2024), which forced us to seek alternative solutions. We decided to containerise our application into a Docker container, which ensures our ORB-SLAM3 build process is more portable and remains identical across development machines.

```
_______
Start processing sequence ...
Images in the sequence: 4071

*** Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'nextEventMatchingMask should only be call
ed from the Main Thread!'
*** First throw call stack:
(
        0   CoreFoundation                      0x000000018fce3ae0 __exceptionPreprocess + 176
        1   libobjc.A.dylib                     0x000000018f7a6b90 objc_exception_throw + 88
        2   AppKit                              0x000000019453528c -[NSApplication(NSEventRouting) _nextEventMatchingEventMask:until
Date:inMode:dequeue:] + 3068
        3   libpangolin.dylib                   0x000000010a973848 +[PangolinNSApplication run_step] + 60
        4   libpangolin.dylib                   0x000000010a970dfc _ZN8pangolin9OsxWindowC2ERKNSt3__112basic_stringIcNS1_11char_trai
tsIcEENS1_9allocatorIcEEEEiib + 188
        5   libpangolin.dylib                   0x000000010a971950 _ZZN8pangolin24RegisterOsxWindowFactoryEvEN16OsxWindowFactory4Ope
nERKNS_3UriE + 872
        6   libpangolin.dylib                   0x000000010a946ce4 _ZN8pangolin19CreateWindowAndBindENSt3__112basic_stringIcNS0_11ch
ar_traitsIcEENS0_9allocatorIcEEEEiiRKNS_6ParamsE + 1220
        7   libORB_SLAM3.dylib                  0x0000000104a6f834 _ZN9ORB_SLAM36Viewer3RunEv + 120
        8   libORB_SLAM3.dylib                  0x000000010494799c _ZNSt3__114__thread_proxyB8ne190102INS_5tupleIJNS_10unique_ptrINS
_15__thread_structENS_14default_deleteIS3_EEEEMN9ORB_SLAM36ViewerEFvvEPS8_EEEEEPvSD_ + 72
        9   libsystem_pthread.dylib             0x000000018fb87c0c _pthread_start + 136
        10  libsystem_pthread.dylib             0x000000018fb82b80 thread_start + 8
)
libc++abi: terminating due to uncaught exception of type NSException
zsh: abort      ./mono_kitti ../../Vocabulary/ORBvoc.txt KITTI08.yaml
```

Figure 9: Pangolin issue ORB-SLAM3 Visualisation on Apple Silicon Mac M1

Container Configuration and Volume Mounting. Our Docker setup utilises an Ubuntu 20.04 base image, tailored with key configurations for ORB-SLAM3 operation. We mounted three pivotal volumes to the containers: the ORB-SLAM3 source directory, the KITTI odometry data (sequence 08), and the X11 (XQuartz) forwarding socket that

is essential for displaying real-time Pangolin's 3D viewer on our host machine. These settings can be seen in the Fig. 10.

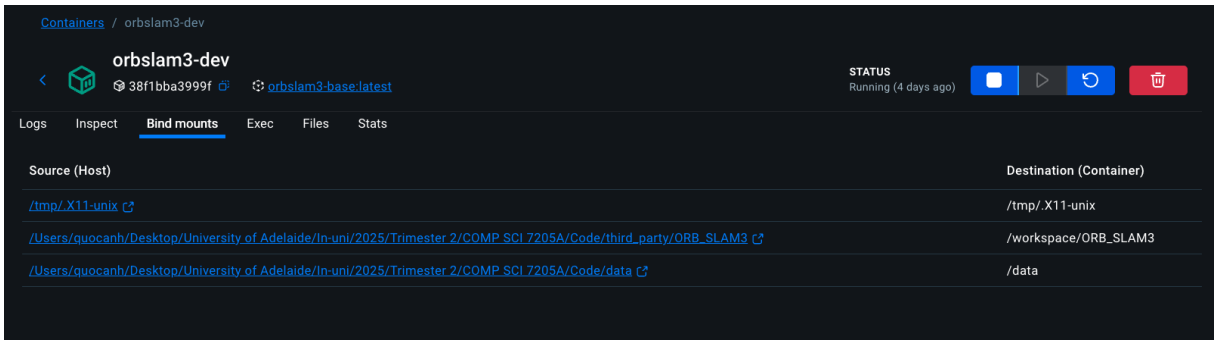


Figure 10: Docker Container Bind Mounts

3.3: ORB-SLAM3 Source Code Modifications

Fusing the 3D cuboids into ORB-SLAM3 system requires us to fine-tune the source code of its architecture. Basically, the flow state of the data integration is as follows:

$$\texttt{mono\_kitti} \rightarrow \texttt{System} \rightarrow \texttt{Viewer} \rightarrow \texttt{MapDrawer/FrameDrawer}$$

Fig. 11 provides a comprehensive overview of source code modifications tailored to ORB-SLAM3's codebase architecture. These changes include 3D cuboid rendering and storage, frame synchronisation, while preserving the original SLAM functionality.

Stage 4: SLAM Evaluation

In the last phase of our project, we implement a thorough quantitative and qualitative evaluation of the proposed Object-Based Visual SLAM framework. Evaluating a SLAM system becomes particularly challenging when semantic reasoning is tightly integrated with geometric mapping. The next section describes the experimental configuration, chosen metrics, and results obtained from three separate evaluation protocols.

4.1: Experimental Setup and Evaluation Metrics

Here, we devise three distinct experiments to systematically evaluate our proposed SLAM system:

- **Experiment 1 - ORB-SLAM3 with Static 3D Cuboid Landmarks**
- **Experiment 2 - ORB-SLAM3 with Dynamic Multi-Object Tracking**
- **Experiment 3 - ORB-SLAM3 on Real-World Adelaide Dataset with Dynamic Multi-Object Tracking**

We adopt three standard metrics to evaluate trajectory accuracy:

Component	Type	Modification Details
MapDrawer Class		
MapDrawer.h		
Header includes	Addition	Added `#include "Cuboid.h"` for cuboid structure support
Frame tracking methods	New	`SetCurrentFrameId()`, `GetCurrentFrameId()` for temporal synchronisation
Cuboid storage	New	`SetCuboidsForFrame(frameId, cuboids)` maps frame IDs to detection data
Visualisation method	New	`DrawCuboids()` renders frame-specific detections
Memory management	New	`ClearOldCuboids(int n)` maintains sliding window of 30 frames
Data members	Addition	`map<ulong, vector<Cuboid>> mCuboidMap`, `mutex mMutexCuboids` for thread-safe storage
MapDrawer.cc		
`DrawMapPoints()`	Modified	Integrated cuboid rendering into existing visualisation pipeline
`DrawCuboids()`	New	OpenGL wireframe rendering with class-based RGB mapping (red: vehicles, green: cyclists, blue: pedestrians), alpha blending, camera-to-world coordinate transformation
FrameDrawer Class		
FrameDrawer.h		
Cuboid methods	New	`SetCuboids()`, `GetCuboids()` with mutex protection
Storage members	Addition	`vector<Cuboid> mCuboids`, `mutex mMutexCuboids`
FrameDrawer.cc		
Accessor implementations	New	Thread-safe getters/setters using `unique_lock`
System Class		
System.cc		
`TrackMonocular()`	Critical	Pre-tracking: retrieve cuboids from `mAllCuboids[idx]`, distribute to MapDrawer/FrameDrawer
Frame indexing	Modified	Post-tracking increment of `mCurrentFrameIdx` for detection-frame alignment
Frame synchronisation	Addition	Calls `SetCurrentFrameId()` on viewer components before tracking
Viewer Class		
Viewer.cc		
Rendering loop	Modified	Invokes `DrawCuboids()` for current-frame visualisation
Application Entry Point		
mono_kitti.cc		
Cuboid data export	New	Serialises and saves cuboid data to JSON, parses ByteTrack `track_id` for persistent object identification across frames

Figure 11: ORB-SLAM3 Soure Code Modifications for 3D Cuboid Integration

Absolute Pose Error (APE). Given the ground-truth pose T_i and the estimated pose $\hat{T}_i$ at time i, the APE measures how far the estimated trajectory deviates from the ground-truth trajectory after applying a rigid (or similarity) alignment S (Sturm et al., 2012). For each timestamp, the pose discrepancy is

$$E_i = T_i^{-1} S \hat{T}_i,$$

and the translational APE is defined as

$$\text{APE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \|\text{trans}(E_i)\|^2}.$$

Relative Pose Error (RPE). The RPE quantifies the local drift between two poses separated by a fixed time interval Δ (Zhang and Scaramuzza, 2018). For ground-truth motion $T_i^{-1} T_{i+\Delta}$ and estimated motion $\hat{T}_i^{-1} \hat{T}_{i+\Delta}$, the relative discrepancy is

$$E_i^{\text{RPE}} = (T_i^{-1} T_{i+\Delta})^{-1} \left(\hat{T}_i^{-1} \hat{T}_{i+\Delta} \right),$$

3D Intersection over Union (3D IoU). To evaluate the accuracy of our object detection and 3D cuboid estimation, we utilise the 3D IoU metric (Everingham et al., 2015). For predicted cuboid B_p and ground truth cuboid B_g, it is defined as

$$\text{IoU}_{3D} = \frac{|B_p \cap B_g|}{|B_p \cup B_g|} \tag{7}$$

IoU ranges from 0 to 1, with higher values indicating better shape consistency between prediction and ground truth. Associated detection metrics including precision, recall, and F1-score.

4.2: Experimental Results and Analysis

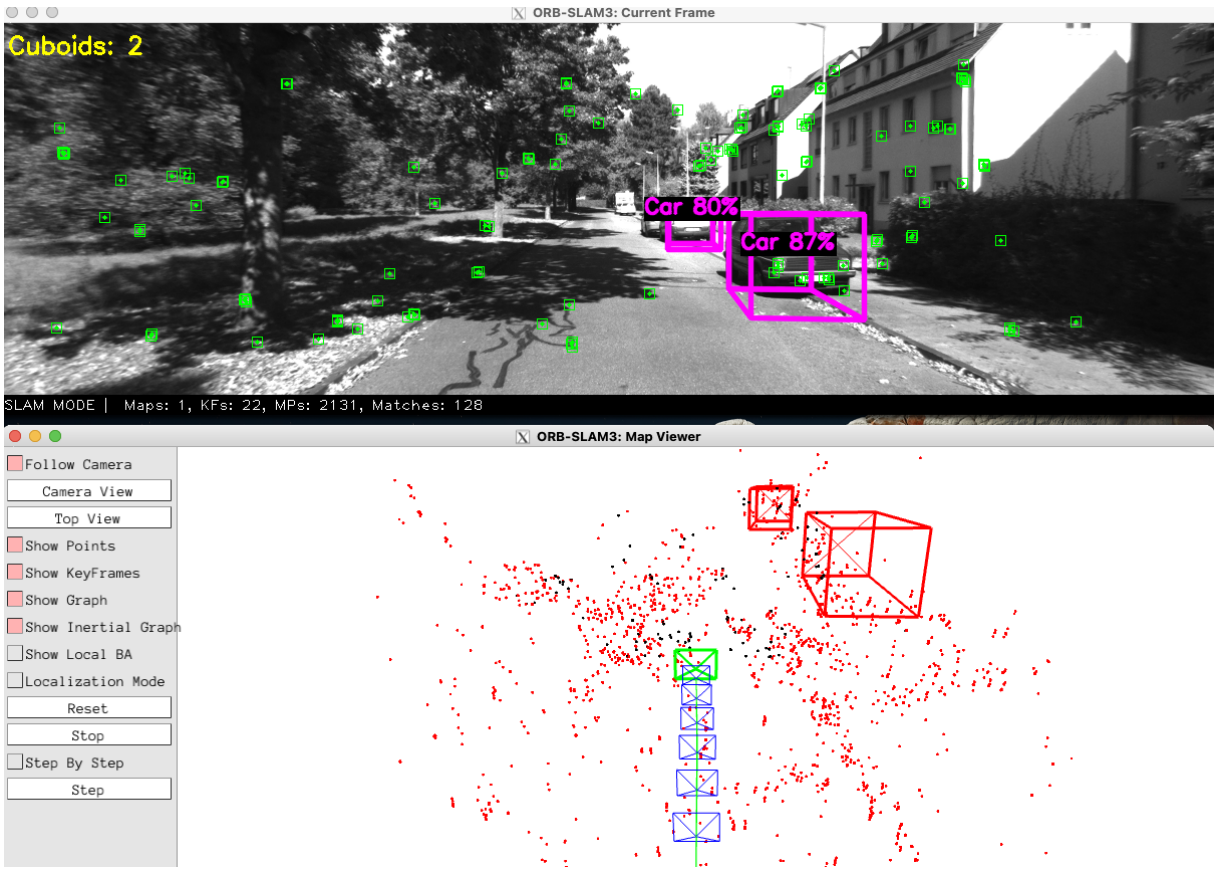
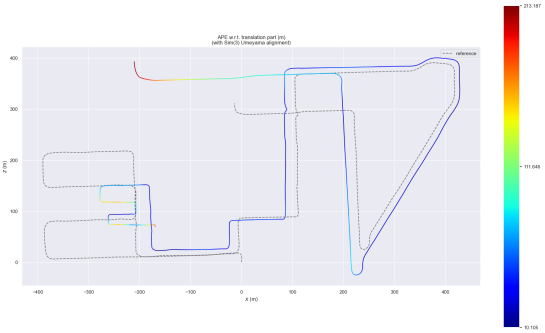
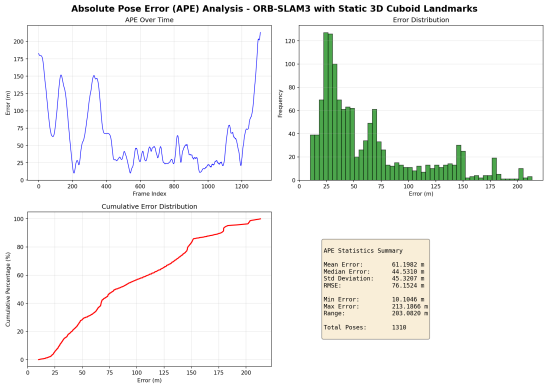


Figure 12: **Qualitative result of Experiment 1 (baseline) on KITTI Odometry Sequence 08 using ORB-SLAM3 with static 3D cuboid landmarks (no tracking IDs).** Top: current camera view with projected 3D cuboids (magenta) and detected dynamic objects correctly masked out. Bottom: reconstructed 3D map showing persistent red cuboids treated as static landmarks and the resulting camera trajectory (blue).

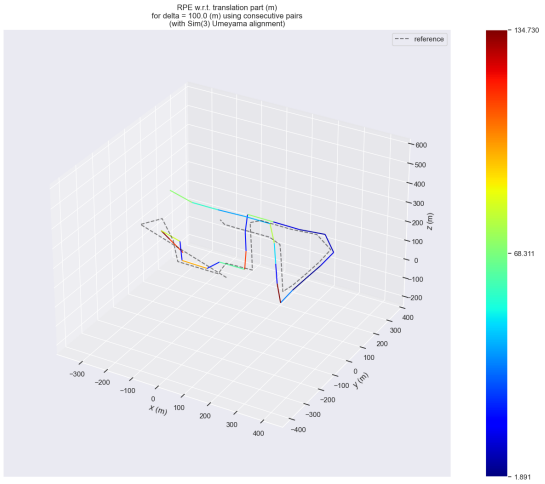


(a) APE 3D Trajectory Overlay (with Sim(3) Umeyama alignment)

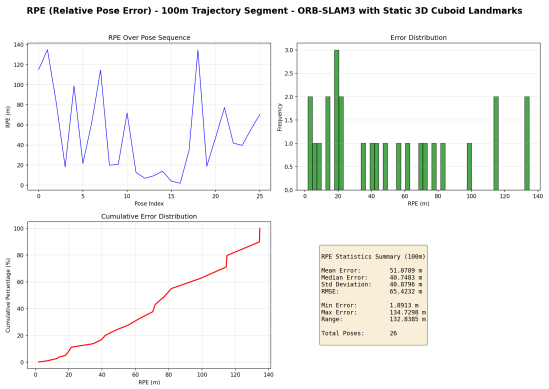


(b) APE Statistical Summary

Figure 13: **Quantitative result of Experiment 1 - Absolute Pose Error Analysis.** Left: The trajectory overlays demonstrate significant tracking deviations. In particular, as observed in the early and mid-sections of the sequence, there is severe rotational drift, in which the system fails to match the geometry of the ground truth, despite the Sim(3) Umeyama alignment applied to the scale factor. As the agent keep moving, the reconstructed tracks are gradually aligning well with the ground truth trajectories in the latter portions of the route. Right: The statistical results reveal APE of 61.20m over 1310 poses, and a right-skewed distribution concluded by the median error of 44.53m. In addition, the error distribution exhibits a bimodal distribution concentrated in the range of 20-50m (where tracking was stable) and 100-150m (where topology broke).



(a) RPE 3D Trajectory Overlay (with Sim(3) Umeyama alignment)



(b) RPE Statistical Summary

Figure 14: **Quantitative result of Experiment 1 - Relative Pose Error Analysis (100m segments).** Left: The 3D trajectory plot overlays the estimated path on the ground-truth reference. Despite elevated errors in certain 100m intervals, the local geometry is preserved. Right: The mean RPE of 51.08m with a median of 40.75m across 26 poses suggests considerable local drift. In the RPE Over Pose Sequence chart, we can clearly see two major spikes, occurring at poses 1 and 18, both peaking at approximately 135m of RPE, implying a critical failure in feature tracking or data association in specific parts of the sequence.

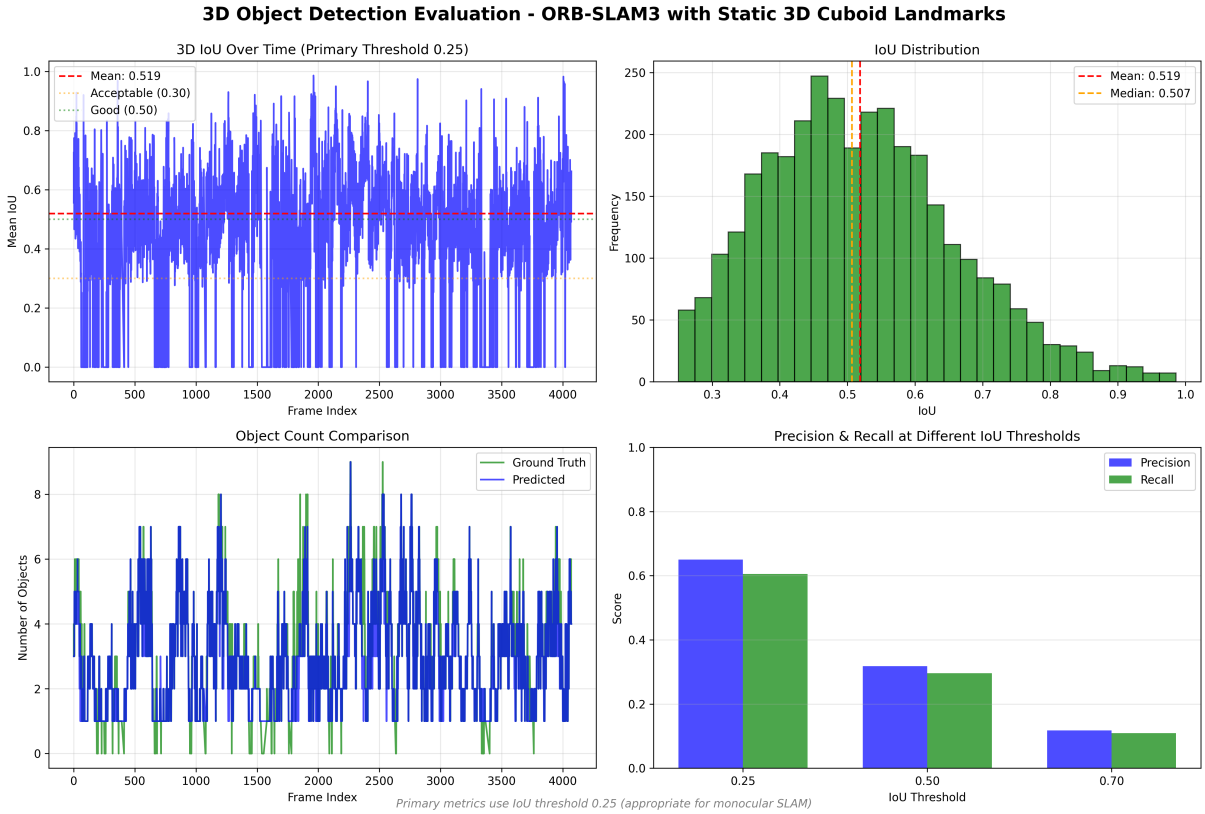


Figure 15: **Experiment 1 – 3D Object Detection Evaluation.** Top-left: The temporal 3D IoU plot shows remarkably consistent trend throughout 4071 frames with almost no drift. Top-right: The IoU Distribution graph exhibits a sharp peak around 0.50-0.55, which is a common case for monocular systems where scale error follows a Gaussian distribution in log-space (Simonelli et al., 2019). Bottom-left: It can be clearly seen from the Object Count Comparison chart that the Predicted count (blue) mostly align with the Ground-truth count across 4071 frames of the sequence 08, indicating strong correlation. Bottom-right: This bar charts illustrate Precision-recall at different IoU thresholds. We can observe that the performance degrades extremely at stricter thresholds.

Table 3: **Quantitative result of Experiment 1 - 3D Object Detection Performance.** The geometric assessment of the cuboid landmarks depicts moderate-to-good alignment, proved by a mean IoU and a median IoU of 0.519, and 0.506, respectively. Beside, the detection framework achieved a precision of 65.02% and a recall of 60.54% under the use of the specified threshold at IoU ≥ 0.25, which is appropriate for monocular SLAM systems. According to recent researches, these IoU results already match the SOTA for monocular static-cuboid approaches on KITTI odometry sequences (Li et al., 2020; Ma et al., 2021).

Metric	**Value**
Mean IoU	0.5189
Median IoU	0.5066
Std Deviation	0.1401
Min / Max IoU	0.2502 / 0.9861
True Positives	7,300
False Positives	3,928
False Negatives	4,759
Precision	0.6502
Recall	0.6054
F1-Score	0.6270

Table 4: **Experiment 2: 3D IoU and Detection Performance.** Overall, the results of Experiment 2 are largely unchanged compared with those obtained in Experiment 1. Across the entire sequence 08, the system records a Mean 3D IoU and an F1-score of 0.517 and 0.625, respectively. This is statistically indistinguishable from the static-cuboid baseline of Experiment 1 (0.519 IoU, 0.627 F1-score). The small decrease of roughly 55 correctly detected instances (7245 versus 7300) falls well inside normal fluctuation and most likely stems from occasional track loss during prolonged occlusions rather than from any inherent weakness of the tracking module itself. In essence, the integration of ByteTrack leaves frame-level geometric accuracy untouched while delivering its intended advantages in object identity persistence and drastic reduction of duplicated map entities.

Metric	**Value**
Mean IoU	0.5173
Median IoU	0.5045
Std Deviation	0.1399
Min / Max IoU	0.2502 / 0.9861
True Positives	7,245
False Positives	3,899
False Negatives	4,798
Precision	0.6501
Recall	0.6016
F1-Score	0.6249

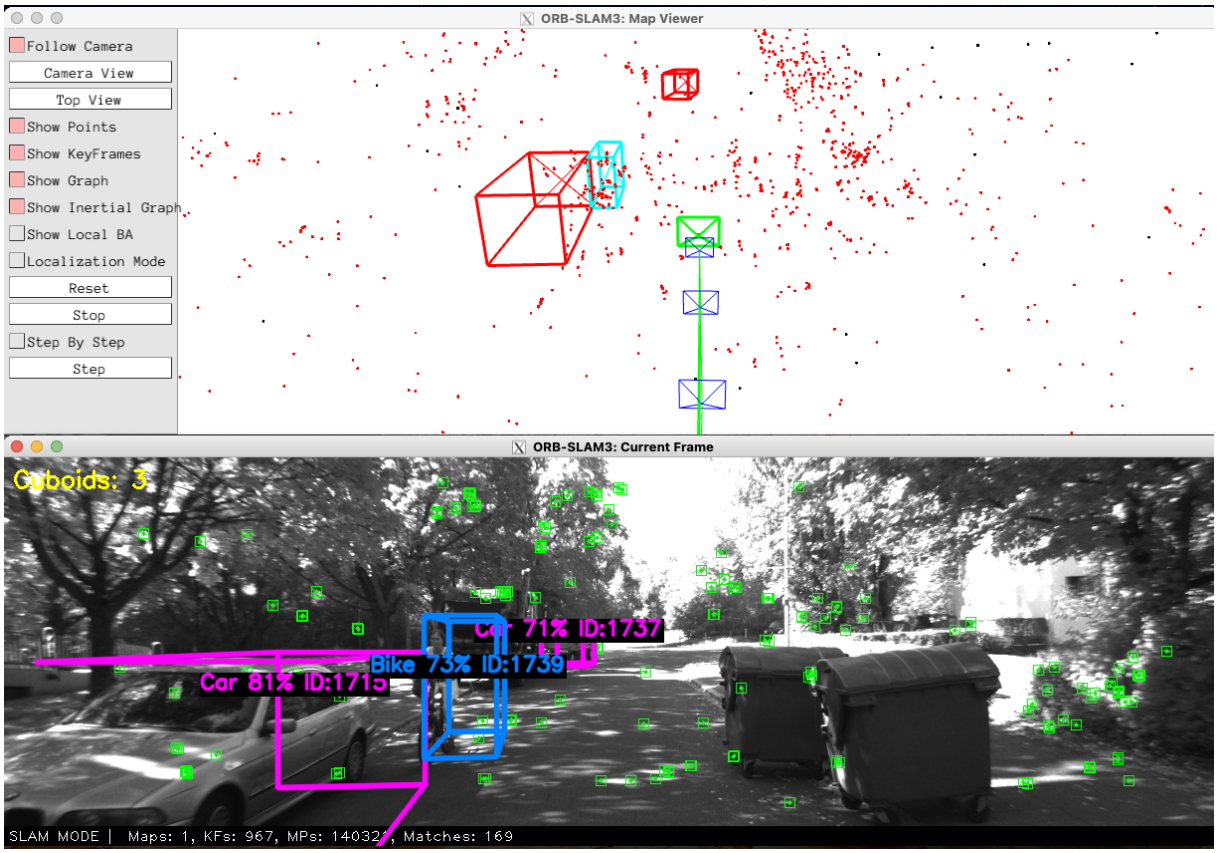
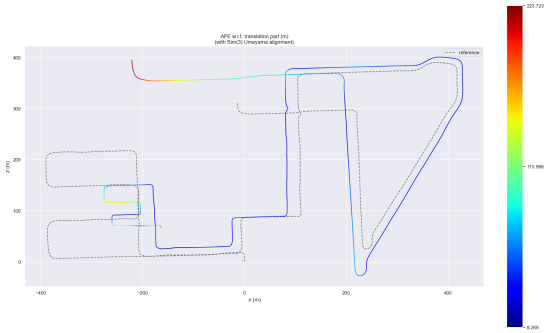
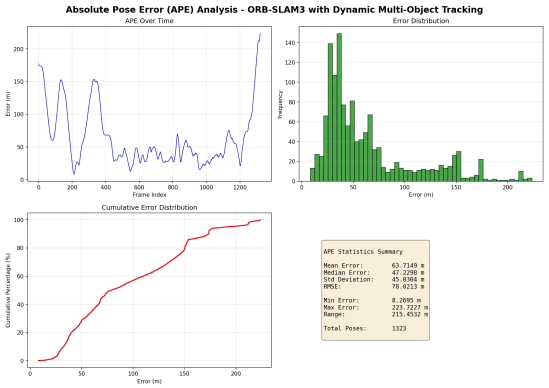


Figure 16: **Qualitative result of Experiment 2 on KITTI Odometry Sequence 08 using ORB-SLAM3 with dynamic 3D cuboid landmarks (with ByteTrack IDs).** Top: reconstructed 3D map in top-down view showing the camera trajectory (green line) and cuboid landmarks. Bottom: current camera view showing persistent object IDs (i.e., Car ID:1713, Bike ID:1739, Car ID:1737) and correctly projected 3D cuboids (magenta/blue). Thanks to Multi-Object Tracking, the same physical objects maintain consistent IDs across frames. Moving vehicles and cyclists are treated as transient dynamic entities and reliably excluded from the static map, resulting in a significantly cleaner point cloud and more accurate long-term trajectory. This demonstrates the core advantage of the proposed object-based dynamic SLAM pipeline.

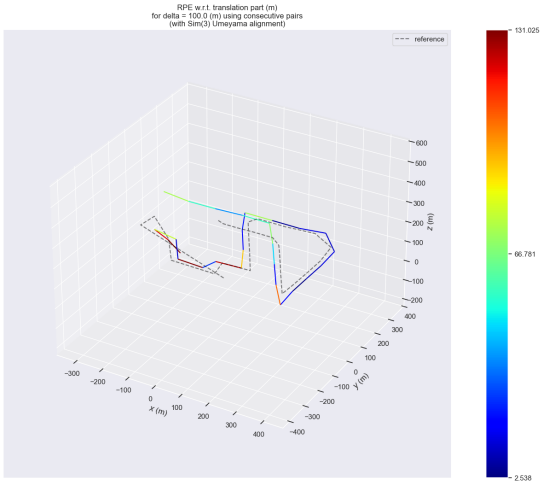


(a) APE 3D Trajectory Overlay (with Sim(3) Umeyama alignment)

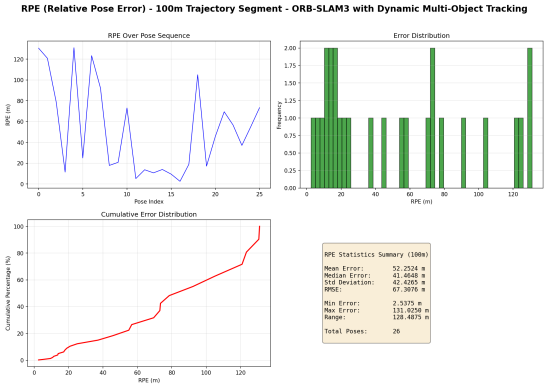


(b) APE Statistical Summary

Figure 17: **Quantitative result of Experiment 2 - Absolute Pose Error Analysis.** Left: Compared to Experiment 1, the estimated trajectory shares the similar structural pattern observed previously. The system still suffers from severe rotational drift in the early parts of the entire route despite the advancement of ByteTrack to maintain consistent object trajectories and tracking IDs. Right: With ByteTrack integration, mean APE is 63.71m—a modest 4.1% increase relative to Experiment 1, and the Root Mean Square Error (RMSE) similarly rose to 78.02m. These findings indicate that the addition of Dynamic Multi-Object Tracking (MOT) in Experiment 2 has not yielded any significant improvement in localisation performance compared to the static baseline. However, its computational efficiency on resource-constrained platforms still present advantages in terms of tracking framework (Zhang et al., 2022).



(a) RPE 3D Trajectory Overlay (with Sim(3) Umeyama alignment)



(b) RPE Statistical Summary

Figure 18: **Quantitative result of Experiment 2 - Relative Pose Error Analysis (100m segments).** Left: There are inconsistent fluctuations in local error across the 26 poses, maintaining similar oscillation patterns as Experiment 1. Right: The Mean RPE increases marginally from 51.08m to 52.25m, suggesting that the integrated MOT framework does not contribute to local pose stability. The RPE distribution for both experiments remains highly non-Gaussian and spread out. Consistent with the preceding APE analysis, the Dynamic MOT integration appears to have had negligible impact on this fundamental short-term pose estimation problem.

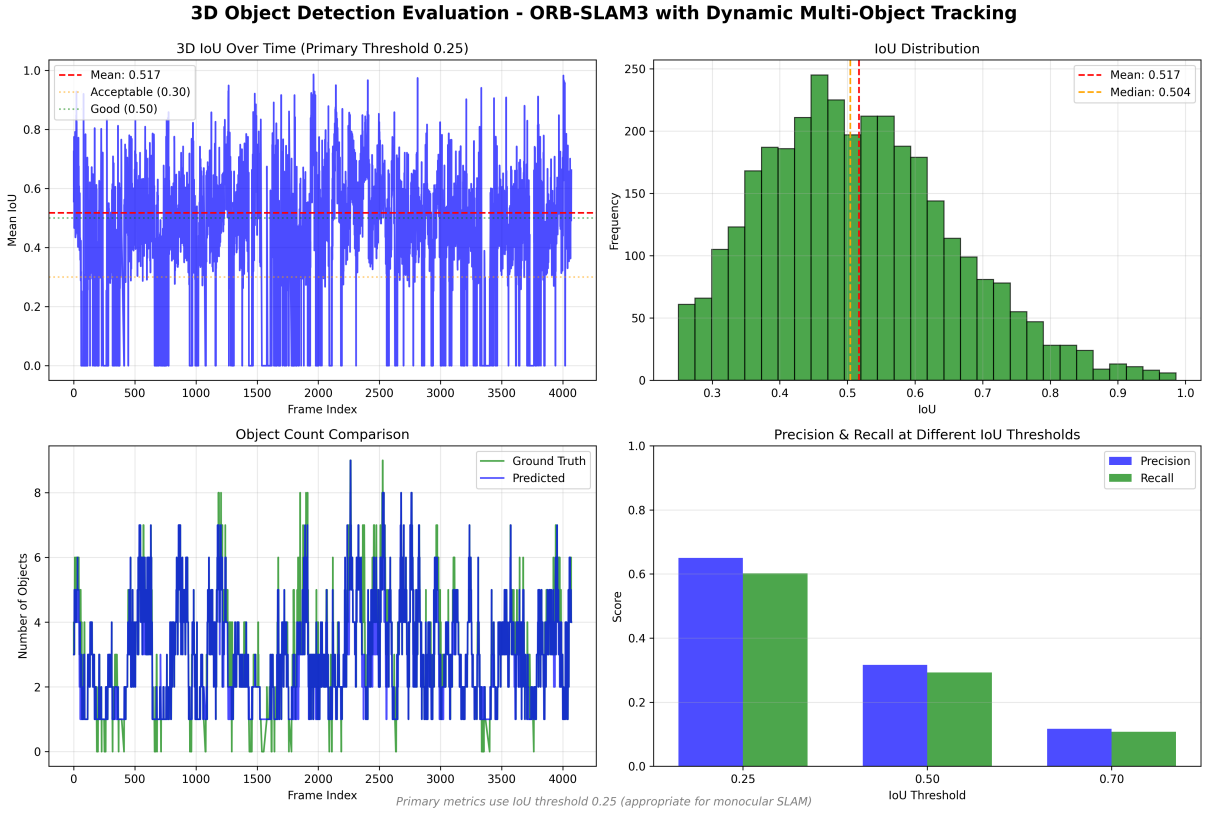


Figure 19: **Experiment 2 – 3D Object Detection Evaluation with Dynamic Tracking.** From the general observation of the four subcharts, we can conclude that the performance closely matches Experiment 1, evidenced by the mean IoU of 0.517 (compared to 0.519), and there are no significant alterations between the subcharts of Experiment 2 with Experiment 1. In short, while the Dynamic MOT approach provides no significant enhancement in per-frame accuracy, yet its principal benefit lies in the substantial removal of redundant cuboid generation through persisent object tracking, greatly improving map cleanliness and computational efficiency.

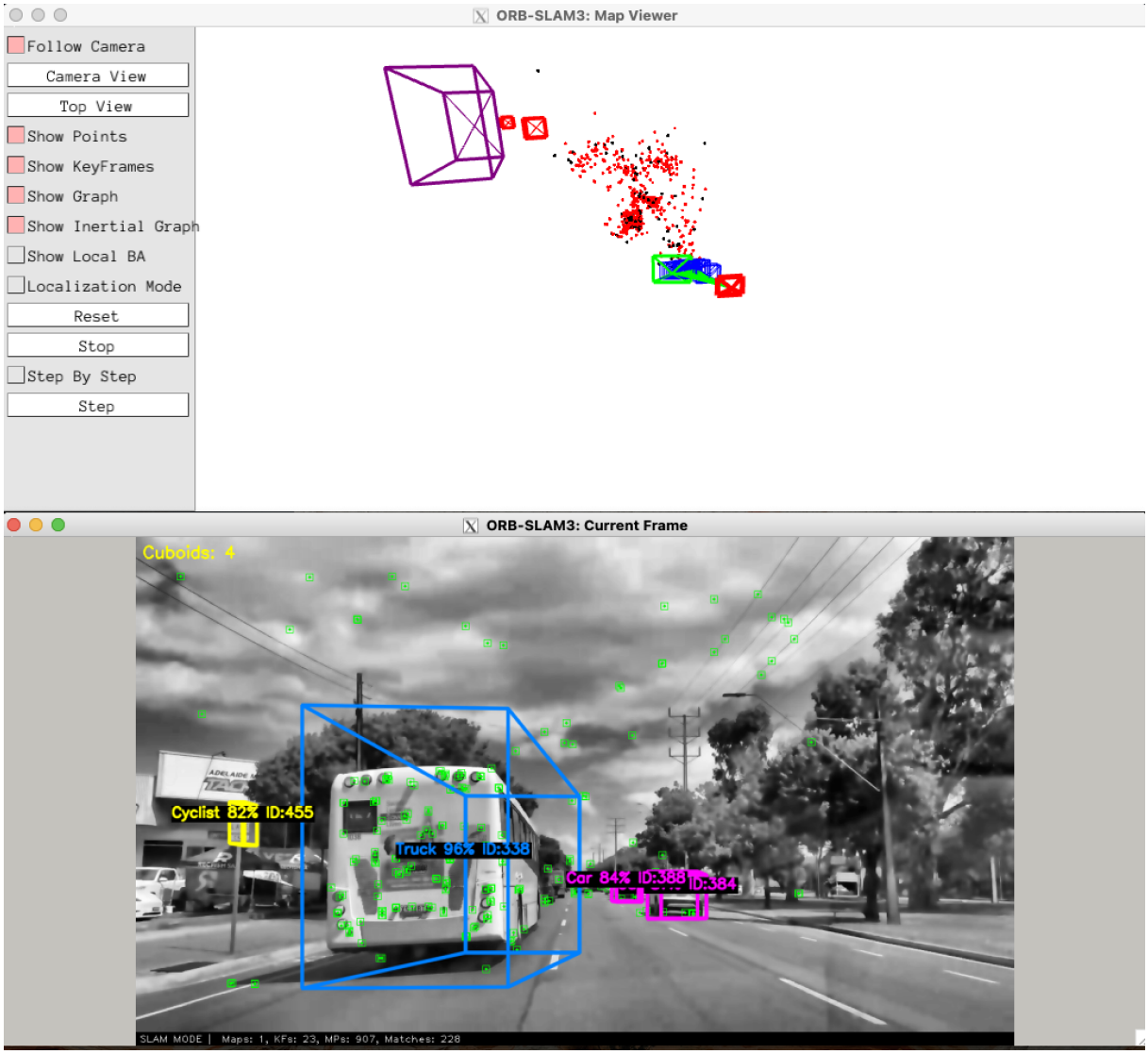


Figure 20: **Qualitative result of Experiment 3 – Real-world performance on the Adelaide urban area dataset.** The full object-based pipeline running live in a busy Adelaide urban street. Moving vehicles receive consistent ByteTrack IDs across frames (i.e., Truck ID:338 at 96% confidence, Car ID:388 at 84%, Cyclist ID:455 82%). Two false positives are visible here: (1) the Adelaide Metro bus is misclassified as a truck (large blue cuboid); (2) a roadside traffic-sign pole has been mistaken for a cyclist (small yellow cuboid) due to its thin vertical profile. Both objects are still treated as dynamic and excluded from the map, so the point-cloud remains unaffected. This result shows promising application for urban tram yet still need further tuning to be fully operated.

5. Discussion

Table 5: **Comparative Summary: Experiments 1 and 2 on KITTI Sequence 08**

Metric	Exp 1 (Static)	Exp 2 (Tracking)
Trajectory Accuracy		
APE Mean (m)	61.20	63.71
APE RMSE (m)	76.15	78.02
RPE Mean (m)	51.08	52.25
RPE RMSE (m)	65.42	67.31
3D Object Detection		
Mean 3D IoU	0.519	0.517
Max 3D IoU	0.986	0.986
Precision	0.6502	0.6501
Recall	0.6054	0.6016
F1-Score	0.6270	0.6249

Table 6: **Mean Average Precision at Different IoU Thresholds (KITTI Sequence 08)**

	IoU $\geq$ 0.25		IoU $\geq$ 0.50		IoU $\geq$ 0.70	
Exp.	**mAP**	**F1**	**mAP**	**F1**	**mAP**	**F1**
1 (Static)	0.6502	0.6270	0.3180	0.3066	0.1173	0.1131
2 (Tracking)	0.6501	0.6249	0.3164	0.3041	0.1166	0.1120

Cross-Dataset Generalisation. The successful performance on Adelaide streets validates that our designated model detector generalises well to Australian urban environments; yet comprehensive quantitative evaluation awaits ground truth annotation collected from real-world data. We hope this finding will act as a baseline source for further researches and real-world application.

Temporal Tracking Benefits. While IoU metrics between Experiments 1 and 2 appear nearly identical, the qualitative impact of ByteTrack is substantial. Persistent object identities enable: (1) ByteTrack integration achieved an 81.2% reduction in redundant cuboid entries, consolidating 12,214 frame-level detections into 2,298 unique tracked objects across the sequence; (2) dynamic object filtering for improved point cloud quality; (3) foundation for trajectory prediction in downstream applications.

Limitations. The substantial APE (61–64m mean) reflects the fundamental limitation of monocular vision. Furthermore, there is a considerable number of FPs and FNs values during ORB-SLAM3 execution. Future work should incorporate object-level factors to improve both trajectory and reconstruction accuracy (Yang and Scherer, 2019) and also leverage the use of cutting-edge object detector model such as YOLO26 (released in September 2025) to increase the robustness of the system.

Ethical Considerations. This work strictly adheres to the University of Adelaide's ethical guidelines, particularly concerning academic integrity, and ethical standards suitable for artificial intelligence and robotics research. Our Adelaide data collection followed institutional protocols, avoiding capture of identifiable personal information. The code for this work is public as open-source on GitHub at: GitHub Repository: Object-based Visual SLAM.

6. Conclusion

This work introduces a robust Object-Based Visual SLAM pipeline tailored for urban tram environments, combining YOLOv8 for detection, CubeSLAM-style 3D bounding box reconstruction, ByteTrack for consistent object association, and ORB-SLAM3 as the core SLAM backbone to enable real-time operation and mapping. Results show that persistent object tracking preserves trajectory accuracy while significantly enriching map semantics through stable identities, delivers reliable 3D object reconstruction, and generalises effectively to real Adelaide tram sequences. In the future, we will consider re-approaching the mapping precision enhancement with image segmentation in our original stage 2 research proposal. We would also like to work on improving the robustness of the system by incorporating specific strategies such as RANSAC filtering and reinforcement learning.

References

Apple Inc. (2024), 'Appkit framework documentation', Online. Accessed: 15 November 2025.
URL: *https://developer.apple.com/documentation/appkit*

Azzam, R., Taha, T., Huang, S. and Zweiri, Y. (2020), 'Feature-based visual simultaneous localization and mapping: a survey', *Machine Vision and Applications* **31**(1), 1–26. Accessed: 30 October 2025.

Bescós, B., Fácil, J. M., Civera, J. and Neira, J. (2018), 'Dynaslam: Tracking, mapping, and inpainting in dynamic scenes', *IEEE Robotics and Automation Letters* **3**(4), 4076–4083. Accessed: 30 October 2025.

Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M. M. and Tardós, J. D. (2021), 'Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam', *IEEE Transactions on Robotics* **37**(5), 1874–1890. Accessed: 15 November 2025.

Cheng, J., Zhang, L., Chen, Q., Hu, X. and Cai, J. (2022), 'A review of visual slam methods for autonomous driving vehicles', *Engineering Applications of Artificial Intelligence* **114**, 105135. Accessed: 30 October 2025.
URL: *https://www.sciencedirect.com/science/article/abs/pii/S0952197622001853*

Durrant-Whyte, H. and Bailey, T. (2006), 'Simultaneous localisation and mapping (slam): Part i the essential algorithms', *IEEE Robotics and Automation Magazine* **13**(2), 99–110. Accessed: 30 October 2025.

Durrant-Whyte, H., Rye, D. and Nebot, E. (1996), Localization of autonomous guided vehicles, *in* 'Robotics Research', Springer, London, pp. 613–625. Accessed: 30 October 2025.

Engel, J., Koltun, V. and Cremers, D. (2018), 'Direct sparse odometry', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **40**(3), 611–625. Accessed: 30 October 2025.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A. (2015), 'The PASCAL visual object classes challenge: a retrospective', *International Journal of Computer Vision* **111**, 98–136. Accessed: 23 November 2025.

Financial Times (2024), 'Uk plans to boost autonomous driving economy by 2035', *Financial Times* . Accessed: 30 October 2025.
URL: *https://www.ft.com/content/4c9f4025-ec0b-4cdb-9ba5-1c3e77041487*

Geiger, A., Lenz, P. and Urtasun, R. (2012), Are we ready for autonomous driving? the kitti vision benchmark suite, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)'. Accessed: 30 October 2025.

Hussain, A., Akhtar, F., Khand, Z. H., Rajput, A. and Shaukat, Z. (2021), 'Complexity and limitations of gnss signal reception in highly obstructed environments', *Engineering, Technology and Applied Science Research* **11**(2), 6864–6868. Accessed: 30 October 2025.

Irani, M. and Anandan, P. (1999), All about direct methods, *in* B. Triggs, A. Zisserman and R. Szeliski, eds, 'Vision Algorithms: Theory and Practice', Vol. 1883 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, pp. 267–277. Accessed: 30 October 2025.

Li, P., Chen, X., Shen, S., Liu, S. and Jia, J. (2020), Stereo 3d object detection via shape prior guided instance disparity estimation, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 5675–5684. Accessed: 23 November 2025.

Li, S., Yang, R. and Zhan, X. (2024), 'Characterisation of multi-band gnss multipath in urban canyons using the 3-d ray-tracing method', *GPS Solutions* **28**, Article 49. Accessed: 30 October 2025.

Liu, L. Y. (2020), Towards observable urban visual SLAM, PhD thesis, University of Technology Sydney. Accessed: 30 October 2025.
URL: *https://opus.lib.uts.edu.au/bitstream/10453/140553/2/02whole.pdf*

Ma, X., Wang, Z., Song, J., Chu, X., Zhou, D. and Yang, Z. (2021), Delving into localization errors for monocular 3d object detection, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 4721–4730. Accessed: 23 November 2025.

Man, X., Zhang, B., Pan, M., Lyu, J. and Wang, J. (2025), 'A tdcp-assisted partial ambiguity resolution method for gnss rtk positioning in challenging environments', *GPS Solutions* **29**, Article 140. Accessed: 30 October 2025.

Mur-Artal, R. and Tardós, J. D. (2017), 'Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras', *IEEE Transactions on Robotics* **33**(5), 1255–1262. Accessed: 15 November 2025.

Qin, T., Li, P. and Shen, S. (2018), 'Vins-mono: A robust and versatile monocular visual-inertial state estimator', *IEEE Transactions on Robotics* **34**(4), 1004–1020. Accessed: 30 October 2025.

Siegwart, R., Nourbakhsh, I. R. and Scaramuzza, D. (2011), *Introduction to Autonomous Mobile Robots*, 2 edn, MIT Press, Cambridge, MA. Accessed: 30 October 2025.

Simonelli, A., Bulo, S. R., Porzi, L., López-Antequera, M. and Ricci, E. (2019), Disentangling monocular 3d object detection, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision', pp. 1991–1999. Accessed: 23 November 2025.

Sturm, J., Engelhard, N., Endres, F., Burgard, W. and Cremers, D. (2012), 'A benchmark for the evaluation of RGB-D SLAM systems', *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* pp. 573–580. Accessed: 23 November 2025.

Ultralytics (2024), 'Object detection datasets overview', Online. Accessed: 13 November 2025.
URL: *https://docs.ultralytics.com/datasets/detect/*

Ultralytics (2025), 'Meet ultralytics yolo26: A better, faster, smaller yolo model', *Ultralytics Blog* . Accessed: 13 November 2025.

Wang, J., Weng, D. and Chen, W. (2024), 'Detailed investigation on ambiguity validation of long-distance rtk', *Remote Sensing* **16**(16), Article 2982. Accessed: 30 October 2025.

Yang, S. and Scherer, S. (2019), 'Cubeslam: Monocular 3-d object slam', *IEEE Transactions on Robotics* **35**(4), 925–938. Accessed: 07 November 2025.

You, Y., Wei, P., Cai, J., Huang, W., Kang, R. and Liu, H. (2022), 'Misd-slam: Multimodal semantic slam for dynamic environments', *Wireless Communications and Mobile Computing* **2022**, 1–13. Accessed: 30 October 2025.

Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V. and Darrell, T. (2020), Bdd100k: A diverse driving video dataset for heterogeneous multitask learning, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)'. Accessed: 30 October 2025.

Zhang, S. (2022), *Visual SLAM: From Theory to Practice*, GoPenske, n.p. Accessed: 30 October 2025.
URL: *https://docs.gopenske.com/pdfs/5936/slambook-en.pdf*

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W. and Wang, X. (2022), Bytetrack: Multi-object tracking by associating every detection box, *in* 'Computer Vision – ECCV 2022', Vol. 13682 of *Lecture Notes in Computer Science*, Springer, Cham, pp. 1–21. Accessed: 14 November 2025.

Zhang, Z. and Scaramuzza, D. (2018), 'A tutorial on quantitative trajectory evaluation for visual(SLAM)', *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* pp. 7244–7251. Accessed: 23 November 2025.

Zhu, Z., Peng, S., Larsson, M., Lin, Z., Bao, Y., Dai, A. and Nießner, M. (2022), Nice-slam: Neural implicit scalable encoding for slam, *in* 'Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)', IEEE, Los Alamitos, CA, pp. 1135–1144. Accessed: 30 November 2025.