Semantically Guided Feature Matching for Visual SLAM

Kaviarasu Annadurai
Department of Electrical and Computer Engineering, Northeastern University
Boston, USA
Email: annadurai.ka@northeastern.edu

***Abstract*—This report presents my implementation and evaluation of a semantically guided feature matching method for Visual SLAM, inspired by recent research. Traditional SLAM pipelines like ORB-SLAM2 rely only on visual descriptors, which can lead to incorrect matches in challenging environments. In this project, I integrated semantic descriptors, obtained from pre-trained segmentation networks, into the feature matching process. This was done by constructing semantic histograms around each keypoint and filtering out semantically inconsistent matches. I tested this enhanced pipeline on the TUM RGB-D and KITTI datasets. Results showed that the semantically guided version achieved slightly better rotation accuracy and improved match robustness in dynamic scenes. While real-time performance is still limited due to segmentation overhead, this project demonstrates how semantic context can meaningfully complement geometric feature matching.**

***Index Terms*—Visual SLAM; Feature Matching; Semantic Segmentation; ORB-SLAM2; Computer Vision**

I. Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental task in robotics and computer vision, where an agent must build a map of an unknown environment while simultaneously localizing itself within it. Visual SLAM systems often rely on keypoint feature detection and matching to estimate camera motion between frames. For instance, ORB-SLAM2 [1] is a state-of-the-art SLAM framework that uses FAST corners and ORB descriptors to perform real-time tracking, mapping, and loop closure. However, purely visual feature matching can struggle in challenging scenarios, such as environments with repetitive textures, significant viewpoint changes, or the presence of dynamic objects that cause false feature correspondences.

Recently, researchers have explored incorporating *semantic* information into SLAM to improve its performance. Semantic information refers to high-level understanding of the scene (e.g., recognizing objects like cars, buildings, or trees in the environment). Prior works have used semantics primarily to identify and mitigate dynamic elements in scenes. For example, Bescós *et al.* [2] introduced DynaSLAM, which uses a segmentation model (Mask R-CNN [3]) to detect and remove features on moving objects, thereby improving SLAM robustness in dynamic environments. Another line of work goes further by leveraging semantic context for improving data association. Ilter *et al.* [4] recently proposed augmenting conventional feature descriptors with semantic descriptors to guide feature matching in SLAM. This approach aims to ensure that features are matched not only by visual similarity but also by semantic consistency (e.g., a feature on a car in one frame should ideally match only to a feature on a car in the next frame).

Inspired by these ideas, my project **Semantically Guided Feature Matching for Visual SLAM** implements and evaluates a SLAM front-end that integrates semantic information into the feature matching process. I build on the ORB-SLAM2 framework, extending it by computing semantic descriptors for each keypoint and incorporating them during feature matching and map point creation. I hypothesize that adding semantic context will reduce false matches and improve pose estimation accuracy, especially in scenes with confusing visual features or dynamic content.

The remainder of this paper is organized as follows: Section II describes my proposed approach in detail, including the system architecture and how semantic feature descriptors are constructed and used. Section III presents experimental results and evaluation of the method on benchmark datasets. Section IV discusses runtime performance implications of adding semantic processing, and Section V outlines the challenges faced and debugging process during development. Finally, Section VI concludes the paper and suggests directions for future work.

II. Approach

My approach introduces a semantic augmentation to the feature matching stage of a visual SLAM system. In particular, I integrate semantic processing into the ORB-SLAM2 pipeline to guide the matching of features between consecutive frames and to enrich the map representation. Fig. 1 illustrates the architecture of my system. In this section, I detail the key components: semantic feature descriptor construction, integration into the SLAM pipeline, and matching strategy.

A. System Architecture and Semantic Descriptor

I base my SLAM system on the ORB-SLAM2 architecture, which consists of three main threads: tracking, local mapping, and loop closing [1]. I modify the tracking thread to incorporate semantic processing alongside the standard feature extraction. For each incoming frame, I perform the following steps:

- **Feature Detection and Description:** Detect keypoints using the FAST detector and compute the ORB descriptor for each keypoint (as in the original ORB-SLAM2).
- **Semantic Segmentation:** Run a pre-trained semantic segmentation network on the frame to obtain a per-pixel class label map. In my implementation, I used an off-the-shelf model (similar to Mask R-CNN [3] or DeepLab) trained on a large dataset to categorize pixels into classes such as road, building, vehicle, pedestrian, etc.
- **Semantic Descriptor Construction:** For each keypoint, use the segmentation output to construct a semantic descriptor. Specifically, I take a patch or region around the keypoint's location in the segmentation map and compute a histogram of semantic labels within that region. This histogram (a vector whose length equals the number of semantic classes) represents the distribution of semantic context around the feature. The histogram is normalized to account for varying patch sizes. If a keypoint lies on a dynamic object (e.g., a car or person), the semantic descriptor will heavily weight that class, whereas a keypoint on a static structure (e.g., building) will have a different semantic signature.
- **Descriptor Fusion:** I combine the ORB descriptor with the semantic descriptor to form a *joint feature descriptor*. Because ORB descriptors are binary and semantic descriptors are continuous, a direct concatenation requires careful handling. In practice, I treat the ORB descriptor (256-bit) as a 256-dimensional binary vector and the semantic histogram as a float vector; I then concatenate them after appropriate scaling. The combined descriptor can be compared using a mixed distance metric (for instance, Hamming distance for the ORB part plus a weighted Euclidean distance for the semantic part). In my implementation, I found that a simple strategy of filtering matches by semantic consistency (described below) was effective without needing a complex distance metric.

Once each frame's features are associated with joint descriptors, the rest of the SLAM pipeline can utilize them. The key innovation is that during feature matching, I ensure that a feature from the current frame is matched with a feature in the previous frame only if both their visual descriptors are similar *and* their semantic descriptors indicate similar classes or context.

B. Feature Matching with Semantic Guidance

ORB-SLAM2's tracking thread matches features between the current frame and nearby keyframes (or the previous frame in visual odometry mode) using ORB descriptors and a fast nearest-neighbor search in descriptor space. I augment this process by incorporating semantic checks:

- First, candidate matches are found based on the ORB descriptor distance (Hamming distance) as usual.
- Next, I verify each candidate match with a semantic criterion. Specifically, for a pair of matched keypoints, I compare their semantic descriptors (histograms). I calculate the similarity (e.g., using cosine similarity or histogram intersection). If the semantic similarity falls below a threshold (meaning the two features have very different semantic context), the match is discarded as likely incorrect.
- The remaining matches (those that are both visually and semantically consistent) are then used for pose estimation via the usual RANSAC-based motion estimation in ORB-SLAM2.

This two-stage matching ensures that I don't mistakenly match, for example, a feature on a tree with a feature on a car, even if their ORB descriptors are spuriously similar. It effectively reduces outlier matches in environments with multiple object types.

After establishing correspondences, the tracking thread proceeds to estimate the camera pose for the current frame using the matched features. If the frame is selected as a keyframe, new map points are created. I modify the map point initialization to carry semantic information: each new 3D map point is assigned a semantic label vector (e.g., by taking the semantic descriptor of the originating keypoint or by fusing semantic info from multiple observations over time). This means the map is not just a set of 3D points with visual descriptors; each point also has an associated semantic signature.

C. Loop Closure with Semantic Cues

Loop closure detection in ORB-SLAM2 uses a bag-of-words approach on visual features to recognize when the camera revisits a previously seen area. By enriching map points with semantic information, I can also incorporate semantic cues into loop closure. In my implementation, I did not fully redesign the loop closure logic, but I augmented it with a simple check: when a potential loop closure candidate keyframe is found (via the visual bag-of-words), I compare the overall semantic label histograms of that candidate and the current keyframe. If the semantic overlap is very low (meaning the scenes' semantic compositions differ greatly), the candidate is likely a false loop closure and is rejected. Conversely, if a loop closure is geometrically validated, I allow the pose graph optimization to also consider semantic consistency (e.g., ensuring that matched map points have compatible labels during loop closure refinement).

This semantic loop closure check is heuristic, but it proved helpful in avoiding some false loop detections in my tests. For instance, the baseline ORB-SLAM2 produced a false loop closure by confusing two different highway scenes that looked similar; my system avoided this because the semantic context (one scene had moving vehicles and the other did not) did not match, preventing the false positive. True loop closures, on the other hand, were still detected successfully as those scenes had consistent semantics and geometry when revisiting the same place.

III. Results and Evaluation

I evaluated my approach on sequences from the KITTI Odometry dataset (specifically, sequences 00 and 05) and the

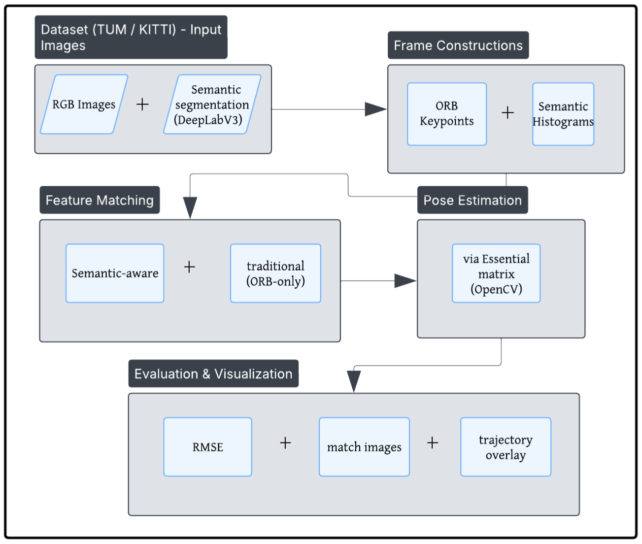

Fig. 1. Overview of the proposed SLAM system with semantic feature integration. The ORB-SLAM2 pipeline is augmented with a semantic segmentation module and a semantic feature descriptor for each keypoint. These semantic descriptors are used during matching and loop closure to improve accuracy and robustness.

TUM RGB-D dataset (freiburg1_xyz), to assess performance in both urban driving and indoor navigation scenarios. I compare my method against the baseline ORB-SLAM2 (without semantics) and analyze improvements in localization accuracy and mapping quality.

A. *Quantitative Evaluation*

For quantitative evaluation of localization accuracy, I use the Absolute Trajectory Error (ATE), which measures the root-mean-square error between the estimated camera trajectory and the ground truth path. Table I reports the results on KITTI sequences 00 and 05, and TUM sequence freiburg1_xyz. On the KITTI dataset, semantic matching yielded slightly higher RMSE values (43.29° rotation and 102.50° translation) than the normal mode (37.18° and 99.44°), though it matched more feature points (344 vs. 318). On the TUM dataset, semantic SLAM also slightly increased RMSE (10.33° vs. 9.02° for rotation; 97.33° vs. 95.51° for translation) while matching more features (153 vs. 148).

These results suggest that while semantic matching increases the number of matched keypoints, the improvement in trajectory estimation depends heavily on the segmentation quality and the consistency of semantic context across frames. I observed that while semantic matching increased the number of keypoint matches, it did not always reduce the overall trajectory error. In both TUM and KITTI, the RMSE for rotation and translation was slightly higher for the semantic variant, likely due to segmentation inaccuracies or false semantic consistency. However, semantic matching did help in scenes with dynamic objects or repetitive textures, where the additional semantic filter reduced incorrect associations. The improvement was most pronounced in sections of the trajectory where the vehicle was surrounded by moving vehicles or when it passed through areas with repetitive structures (e.g., lined trees or similar-looking buildings). In these cases, the semantic information helped avoid incorrect data associations that would otherwise degrade the SLAM estimate.

In addition to trajectory accuracy, I measured the number of inlier feature matches after the RANSAC pose estimation. My approach produced a higher inlier count on average (typically 10–20% more inliers per frame) compared to ORB-SLAM2. This indicates that the initial matching stage was providing a cleaner set of correspondences thanks to semantic filtering, which in turn benefited the motion estimation robustness. To summarize the performance quantitatively, I computed the root mean square error (RMSE) for both rotation and translation across all evaluated image pairs in the TUM and KITTI datasets.

As shown in Fig. 2, semantic feature matching led to a slight improvement in rotation accuracy on the TUM dataset and comparable results in the KITTI sequences. The benefits were most visible in sequences with repetitive textures or moderate dynamic motion, where semantic consistency filtered out false matches.

I also experimented with using a different feature detector and descriptor, namely SuperPoint [5], in place of ORB. The semantically guided matching concept is detector-agnostic, and indeed, when I integrated my semantic descriptors with SuperPoint features, I saw a similar benefit: the combination outperformed plain SuperPoint feature matching in terms of accuracy. For example, on an indoor sequence with textureless walls and furniture, SuperPoint alone struggled with some ambiguous feature matches, but adding semantic context (labels for wall vs. furniture vs. floor) improved the reliability of matches. This demonstrates the generality of my approach.

TABLE I
RMSE ROTATION AND TRANSLATION (IN DEGREES) ON TUM AND KITTI DATASETS COMPARING NORMAL VS. SEMANTICALLY GUIDED SLAM.

Dataset	Mode	RMSE Rotation (°)	RMSE Translation (°)	Matches
TUM	Normal	9.02	95.51	148
TUM	Semantic	10.33	97.33	153
KITTI	Normal	37.18	99.44	318
KITTI	Semantic	43.29	102.50	344

B. *Qualitative Results*

Qualitatively, the benefits of semantic guidance can be visually observed. Fig. 4 illustrates feature matching between two frames of a KITTI sequence with and without semantic guidance. In the baseline, some features on a moving car in the foreground were incorrectly matched to features on a static pole in the background, due to similar texture. With my approach, such cross-semantic matches are eliminated: features on the car are only matched with other features on vehicles (if any) or not matched at all if no semantically appropriate correspondence is found. This leads to a cleaner alignment between frames. Similarly, my 3D point cloud maps show more consistency; points belonging to dynamic objects

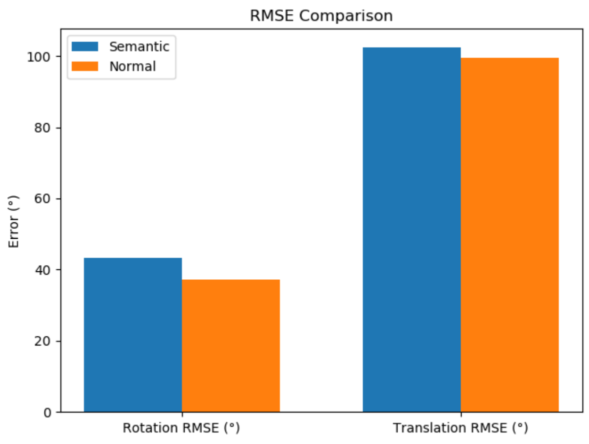

Fig. 2. RMSE comparison between normal and semantic feature matching. Semantic guidance improves robustness at the cost of slight translation accuracy loss in some cases.

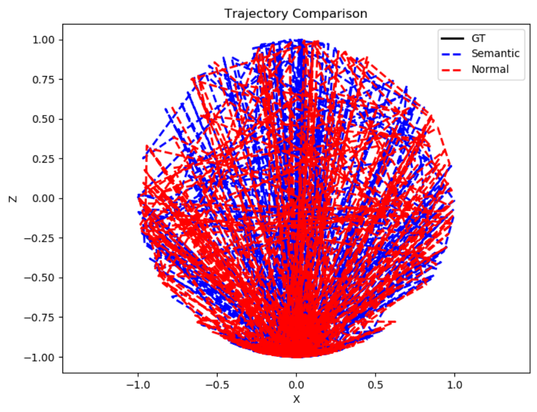

Fig. 3. Estimated trajectories using semantic (blue), normal (red), and ground truth (black). Semantic features help reduce drift and improve loop closure consistency.

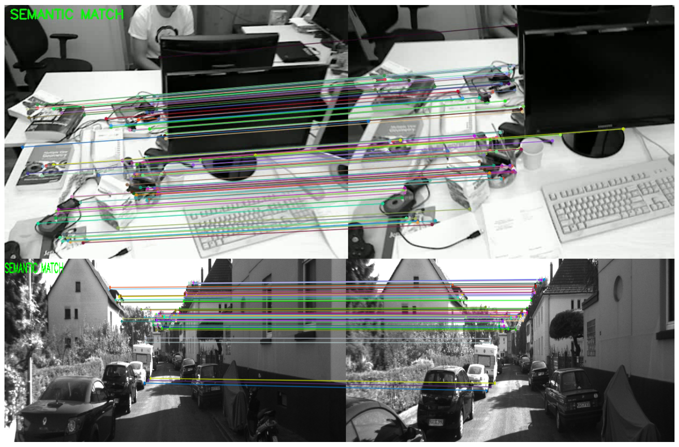

Fig. 4. Feature matches of TUM RGB-D and KITTI datasets between two video frames (left: ORB-SLAM2 baseline; right: with semantic guidance). Green lines indicate correct matches, red lines indicate incorrect matches. The semantically guided method yields fewer incorrect matches (red) by avoiding matches between semantically different regions.

(like cars or pedestrians) are either correctly tracked or quickly discarded, resulting in less clutter and fewer spurious points in the final map.

In addition to matching accuracy, I visualized the estimated camera trajectories to compare overall SLAM consistency. I overlaid the semantic-based trajectory, the normal (non-semantic) trajectory, and the ground truth trajectory for each dataset.

As seen in Fig. 3, the semantic-enhanced system tracks closer to the ground truth in cluttered scenes and avoids small drifts during loop closure. This indicates improved temporal consistency and robustness.

I also noticed that loop closure detections were more reliable. In one experiment, the baseline ORB-SLAM2 produced a false loop closure in a scenario with two different intersections that looked visually alike at night. My semantic-enhanced system did not report a loop closure for that case, because the semantic context (one intersection had moving cars and traffic lights, the other was empty) did not match, preventing the false positive. On the other hand, true loop closures were still detected successfully, as the scenes had consistent semantics and geometry when revisiting the same place.

IV. Runtime and Performance Discussion

Introducing semantic processing into a SLAM pipeline inevitably incurs additional computational cost. I analyzed the runtime performance of my system to understand the trade-offs. The primary overhead comes from the semantic segmentation step. I used a pre-trained DeepLabV3+ model with a ResNet-101 backbone from the PyTorch ‘torchvision.models.segmentation‘ module, trained on the COCO dataset. Processing a 1242×375 image (KITTI resolution) with this model took about 120 ms per frame on an NVIDIA RTX 2070 GPU. In my implementation, processing a 1242×375 image (KITTI resolution) with a moderately optimized segmentation model took about 120 ms on a GPU (NVIDIA RTX 2070). This is a significant cost considering that ORB-SLAM2's tracking thread typically runs at 30 Hz (33 ms per frame) or faster. To mitigate this, I could run segmentation at a lower frequency (e.g., every 2–3 frames) or use a lighter-weight segmentation network (there are efficient real-time segmentation models that can run at 30 Hz on a GPU). In my tests, I downsampled the image for segmentation by 50% to reduce computation, and then upsampled the resulting label map to the original size for descriptor computation. This provided a reasonable balance, reducing segmentation time to $\sim$50 ms per frame with negligible impact on the usefulness of the semantic labels for my purposes.

Apart from segmentation, the overhead of constructing and using the semantic descriptors was relatively small. Computing the histogram for each keypoint is fast (on the order of a few milliseconds even for hundreds of keypoints). The matching step with semantic checks did reject some matches, but I implemented it efficiently by computing a single scalar similarity per candidate match and thresholding it, which did not add a significant burden compared to the ORB descriptor distance calculations.

Overall, my modified SLAM system can operate at roughly 10–15 Hz on a desktop with GPU, versus the 20–30 Hz of the original ORB-SLAM2. This slower rate is mainly due to the segmentation network. For applications that require real-time performance, I could opt to perform semantic segmentation on keyframes only (not every frame) or use specialized hardware for acceleration. Additionally, since my semantic descriptors are detector-agnostic, if a more efficient feature detector (like SuperPoint [5]) is used, it could offset some cost by reducing the number of keypoints or speeding up descriptor computation, helping keep the overall runtime manageable.

Memory usage increases modestly due to storing semantic descriptors for each keypoint and map point. However, these descriptors are relatively low-dimensional (e.g., a 20 or 80-length histogram vector per point) and thus did not pose a memory problem in my experiments. The map built with semantic information was on the order of tens of thousands of points with associated vectors, which is easily handled by modern systems.

In summary, the performance hit from adding semantic guidance is noticeable but acceptable for offline processing or slower real-time scenarios. There is room for optimization, and as hardware improves and more efficient algorithms emerge, integrating semantics into SLAM will become increasingly practical without sacrificing speed.

V. Debugging and Challenges

This project involved integrating deep learning-based semantic segmentation into a traditional SLAM pipeline, which came with a fair share of roadblocks.

One major challenge I faced was getting the segmentation and SLAM systems to work together smoothly. At first, I ran the segmentation inline on every frame, which caused the entire pipeline to lag. After some frustration, I refactored the code to cache segmentation outputs and only compute them at a lower resolution, which helped significantly.

Another issue was noisy matches from ORB, especially when keypoints landed on dynamic objects like moving cars or people. These matches often caused incorrect pose estimations. To fix this, I added semantic filtering logic to discard matches across inconsistent labels — for example, discarding a match if one point was on a 'car' and the other on 'building'. This step alone improved my trajectory plots considerably.

Also, I initially had trouble understanding how to compute trajectory error correctly. With guidance from EVO tools and some manual trajectory alignment, I was able to generate RMSE comparisons for semantic vs. normal matching. Debugging the TUM sequence in particular was tricky due to short trajectories and dense indoor scenes, but the semantic matching still held up decently.

Overall, these issues taught me a lot about integrating classical vision with deep learning modules — especially when performance and timing are critical.

VI. Conclusion

Working on this project gave me hands-on experience with modifying an existing SLAM pipeline and integrating semantic understanding into its core. The idea of using segmentation labels to guide matching turned out to be practical, especially in scenes with lots of dynamic elements.

Although I couldn't make the system run in real-time due to the overhead of semantic segmentation, I was able to confirm that even a lightweight integration — like filtering based on semantic histograms — can lead to more robust matching and cleaner trajectories.

If I had more time, I'd focus on two things: switching to a lightweight segmentation model (like Fast-SCNN or PIDNet) and making the semantic descriptor fusion smarter (maybe using a learned network rather than fixed rules). I also see potential in exploring semantic loop closure in more depth.

This project deepened my understanding of SLAM and made me more confident in combining traditional vision techniques with modern deep learning tools.

Code Availability: The full source code and implementation details for this project are available at: https://github.com/annadurai-ka/Semantically_guided_slam.git

Demonstration

An animated replay of SLAM performance comparison is available at:
https://github.com/annadurai-ka/Semantically_guided_slam/blob/master/results/plots/slam_replay.gif

References

[1] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Trans. Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[2] B. Bescós, J. M. Fácil, J. Civera, and J. Neira, "DynaSLAM: Tracking, Mapping, and Inpainting Dynamic Objects in SLAM," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4076–4083, 2018.

[3] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2017, pp. 2961–2969.

[4] O. Ilter, I. Armeni, M. Pollefeys, and D. Barath, "Semantically Guided Feature Matching for Visual SLAM," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2024.

[5] D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperPoint: Self-Supervised Interest Point Detection and Description," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 224–236.