4

Challenges and Performance of SLAM Algorithms on Resource-constrained Devices

Calvin Galagain[1, 2], Martyna Poreba[1], and François Goulette[2]

[1]Université Paris-Saclay, CEA-List, France
[2]ENSTA Paris, France

Abstract

Evaluating the performance of Simultaneous Localization and Mapping (SLAM) algorithms is essential for the progress of robotic systems. However, conducting a comprehensive assessment of SLAM systems in the context of recent advancements is challenging due to the wide variety of hardware platforms, algorithm configurations, and datasets available. This study aims to test SLAM algorithms on resource-constrained devices such as the NVIDIA Jetson AGX Orin 64GB. Experiments are conducted with various visual-based localization algorithms that either leverage deep learning models for specific tasks within the SLAM process or are learned end-to-end to estimate camera pose. The evaluation focuses on the following systems: RDS-SLAM and VDO-SLAM, which utilize semantic information to achieve precise motion estimation; TSformer-VO, an end-to-end Transformer-based model designed for monocular visual odometry; and DeepVO, which based on recurrent neural networks. The systems are evaluated using several metrics, including ATE and RPE to assess pose accuracy and rotational drift, respectively, alongside runtime, energy consumption, and resource usage to gauge their efficiency and practicality for real-world applications.

Keywords: SLAM, localization, benchmarking, computational efficiency, system efficiency, performance.

4.1 Introduction and Background

Simultaneous Localization and Mapping (SLAM) is a crucial technology that enables autonomous systems, such as robots, drones, and AR/VR devices, to navigate and understand their environments without relying on external reference systems. SLAM algorithms operate by simultaneously constructing a map of an unknown environment while tracking the system's position within it. They typically utilise a combination of sensors, including cameras, LiDAR, and inertial measurement units (IMUs), to gather data about the surrounding area. Integrating these sensors improves the accuracy and robustness of SLAM but also introduces a higher computational burden, posing a substantial challenge to achieving real-time performance. To maintain operational efficiency, optimizations such as simplifying algorithms, reducing the number of processed features, and leveraging parallel processing capabilities are essential. When deployed on embedded systems, SLAM encounters unique challenges due to resource constraints, including limited processing power, memory, and energy consumption. These limitations necessitate the development of highly efficient algorithms capable of performing complex tasks in real time, such as image processing, sensor fusion, loop closure detection, and optimization. Despite advances in the field, numerous challenges persist. Ensuring robustness against sensor noise, managing varying environmental conditions, and scaling algorithms to accommodate different map sizes and complexities are critical areas of ongoing research. Furthermore, the demand for lightweight implementations that do not compromise performance underscores the continuous evolution of SLAM technologies.

The primary aim of this paper is to benchmark specific SLAM methods on the NVIDIA Jetson AGX Orin 64GB, a robust embedded platform tailored for real-time processing in autonomous systems. Specifically, we perform a comprehensive performance analysis, comparing RDS-SLAM [1] and VDO-SLAM [2], two semantic SLAM techniques, against the Visual Odometry (VO) performance of CNN-based methods trained in an end-to-end manner. This benchmarking aims to assess the performance of these SLAM algorithms under constrained resource conditions, with a specific focus on two key aspects:

- **Performance Comparison:** Evaluating the ability of each algorithm to accurately localize in dynamic environments.
- **Resource Utilisation Assessment:** Analysing how each SLAM method leverages the resources of the Jetson AGX Orin, specifically regarding CPU and GPU performance, memory usage, and energy consumption

Finally, the study provides recommendations for optimizing SLAM algorithms on embedded platforms and identifies key areas for future research and development.

4.2 Related Work

SLAM has attracted considerable attention over the past few decades, leading to the development of numerous approaches [1-7]. A thorough review of the existing literature on SLAM methods reveals a diverse array of algorithms tailored for different applications and hardware platforms. For example, ORB-SLAM [3] has been widely recognized for its efficiency and robustness on conventional CPUs, demonstrating good performance across various environments. Similarly, VINS-Mono [4] and VINS-RGBD [5] are noted for their effectiveness in combining visual and inertial data. Recent studies have highlighted the potential of DNN-based SLAM systems, such as RDS-SLAM [1], VDO-SLAM [2], DF-SLAM [6], Dyna-SLAM [7], and DeepFactors [8], which leverage deep learning techniques to improve feature extraction, pose estimation, or environment understanding. End-to-end deep learning methods like DeepVO [9], TS-Former [10], and DROID-SLAM [11] also mark a significant shift in visual localization systems by using neural networks to directly learn the entire process from raw sensor data to pose estimation and map generation, bypassing traditional hand-crafted feature extraction and geometric modeling. However, these methods frequently encounter limitations compared to traditional SLAM techniques, such as lower accuracy and significant dependence on large training datasets. Neural Radiance Fields (NeRF)- based SLAM [12–16] offers a novel solution by incorporating NeRF models into SLAM systems to enhance the representation of 3D environments. In contrast to traditional SLAM methods that rely on discrete points or sparse features, NeRF-enhanced approaches produce continuous volumetric fields to create highly detailed and realistic 3D reconstructions. Through the use of neural networks, these methods can model entire scenes and facilitate photorealistic rendering by understanding the interactions of

light with surfaces. However, NeRF-based SLAM methods face challenges on embedded systems due to their high computational requirements, energy consumption, complex integration, and limited generalisation, underscoring the need for more efficient solutions.

A growing number of embedded computing platforms now feature NPU/GPU units, enabling lightweight deep learning networks to function in real time. Many researchers have worked to modify SLAM algorithms for low-power embedded platforms, reengineering them to ensure compatibility with these devices. Despite these advancements in SLAM technology, implementing these algorithms on resource constrained platforms still faces considerable challenges, largely determined by the distinct characteristics of both the algorithms and the underlying embedded architectures. Although some researchers [17, 18] have explored VO systems resulting in a decreased accuracy, others [19–22] have successfully developed keyframe-based SLAM systems. Various approaches have been explored to optimize computation and power overhead in visual-inertial odometry (VIO), particularly through hardware acceleration using FPGAs [23–25]. Although advancements have been made, keyframe-based SLAM systems still face challenges in achieving an optimal balance between efficiency and accuracy for mobile robot applications. One of the most recent developments, Dynamic-VINS [26], an enhanced iteration of VINS-Mono and VINS-RGBD, showcases remarkable performance on resource-constrained platforms such as the HUAWEI Atlas200 DK and NVIDIA Jetson AGX Xavier. Finally, a hardware-software co-design approach is proposed to optimize latency, power consumption, and tracking speed in VIO systems by incorporating the Optical Flow (OF) estimation on the sensor [27]. In the VINS-Mono pipeline, feature tracking was substituted with an OF camera that employs an ASIC-based accelerator, while the other components of the VIO pipeline operate on the main processor of a Raspberry Pi Compute Module4.

Assessing the performance of SLAM algorithms is essential for both researchers and users of robotic systems. Comprehensive benchmarking enables in-depth evaluations, helping to identify the most effective SLAM algorithms, and laying the groundwork for future enhancements and innovations in the field. The wide variety of hardware configurations, algorithm settings, and datasets complicates thorough comparisons across the state-of-the-art. There is a notable scarcity of research focused on benchmarking SLAM algorithms on embedded systems, highlighting significant gaps in the existing literature that this study aims to address. For example, the research

presented in [28] assesses power consumption, accuracy metrics, and processing frame rates for ORB-SLAM and OpenVSLAM [29] on NVIDIA Jetson embedded systems, particularly the Jetson Nano, Jetson TX2, and Jetson Xavier. The SLAM Hive Benchmarking Suite [30] has recently addressed this challenge by offering a scalable solution that leverages container technology and cloud deployment to analyze thousands of SLAM executions.

4.3 Methodology

For this comparison study, a diverse range of approaches have been selected, including semantic geometric SLAM methods as well as VO techniques derived from deep learning models trained in an end-to-end manner. This enables a comprehensive evaluation of various strategies to identify those most suitable for embedded systems, focusing on balancing performance and computational efficiency. The systems are assessed based on several metrics such as pose accuracy and rotational drift, along with an analysis of runtime, energy consumption, and GPU and CPU usage to determine their efficiency and suitability for real-world applications.

4.3.1 Selected systems

We examine the following systems: RDS-SLAM, which enhances the localization process with semantic segmentation; VDO-SLAM, which utilizes semantic information for accurate motion estimation and tracking of dynamic rigid objects; TSformer-VO, an end-to-end Transformer-based model for monocular visual odometry that learns motion estimation directly from raw images; and DeepVO, which leverages recurrent networks to capture temporal dependencies.

Unlike traditional SLAM algorithms, which assume a static scene, RDS-SLAM (see Figure 4.1) detects and excludes dynamic objects to enhance the robustness of tracking and mapping. The algorithm extends the base framework of ORB-SLAM3 by introducing two parallel threads: a semantic segmentation thread and a semantic-based optimization thread. These threads enable the segmentation of images into static and dynamic objects using methods such as Mask R-CNN [31] or SegNet [32], while optimizing the tracking data in real time without blocking the process. The semantic thread is executed selectively to keyframes, rather than every frame. The semantic information is then propagated across the global map, where each map

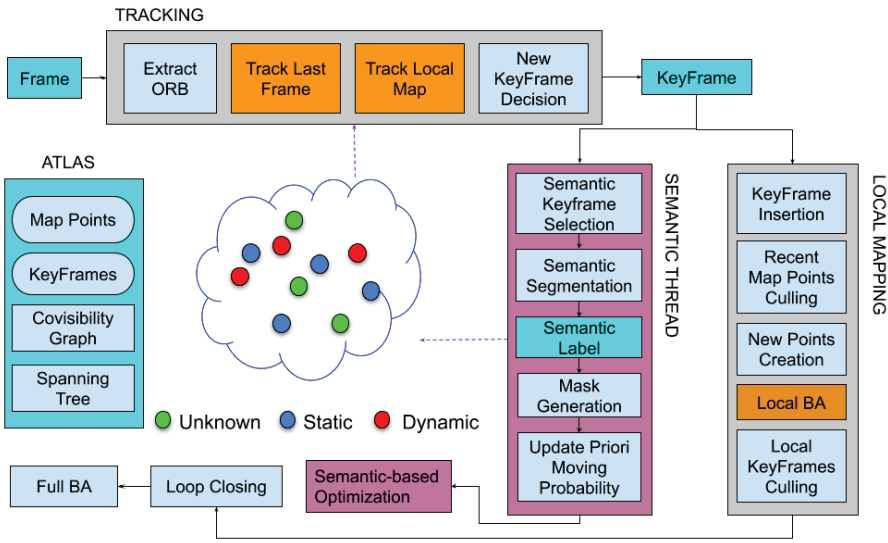

Figure 4.1 Overview of RDS-SLAM [1].

point is assigned a moving probability. This probability is updated as new keyframes are processed and is used to classify map points as dynamic, static, or unknown. The algorithm identifies dynamic objects using segmentation masks from semantic models, assuming that classes such as people and vehicles are likely dynamic. Points classified as dynamic are excluded from the tracking process to avoid introducing errors in the camera pose estimation. In contrast, static points are used to improve the accuracy of the tracking. This approach allows RDS-SLAM to achieve precise tracking and robust mapping, even in environments with dynamic objects that typically pose challenges for traditional SLAM algorithms.

The VDO-SLAM system (Figure 4.2) is also designed to handle dynamic environments. Before the execution of the SLAM algorithm, two crucial pre-processing steps are applied. First, Mask R-CNN is used for instance-level semantic segmentation, which allows for the identification of both static and dynamic objects, such as vehicles and pedestrians, by generating object masks. Second, PWC-Net [33], a state-of-the-art optical flow network, is applied to estimate the dense pixel motion between consecutive frames. Using these data, VDO-SLAM can estimate the full SE(3) motion of dynamic objects, including both their linear velocity and rotational movement, while also refining its own camera pose. This integration allows

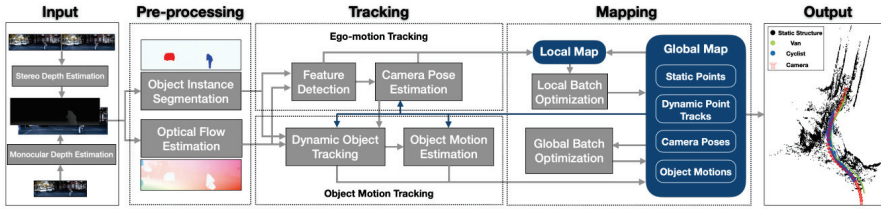

Figure 4.2 Overview of VDO-SLAM [2].

for robust navigation in dynamic environments where traditional SLAM methods, which assume static surroundings, would fail.

Visual odometry, a key component of the tracking phase in SLAM, can also be accomplished using methods based on Convolutional Neural Networks (CNNs). The goal is to enable the system to directly learn motion estimation from raw image inputs, eliminating the need for traditional feature extraction and matching techniques. DeepVO combines Convolutional Neural Networks (CNNs) to extract visual features from images with Long Short-Term Memory networks (LSTMs) to capture and model temporal relationships between consecutive images, enabling the prediction of camera movements.

On the other hand, TSformer-VO takes a distinct approach by utilizing transformers to extract spatio-temporal features from video sequences. Unlike DeepVO, which relies on recurrent mechanisms, TSformer-VO utilizes spatio-temporal attention to capture interactions between images across both spatial and temporal dimensions. This allows for a more holistic understanding of the scene, leading to enhanced precision in estimating the camera's 6-DoF poses. By processing long-range dependencies within the video data, TSformer-VO reduces pose drift and improves robustness in dynamic environments. Additionally, the model's end-to-end learning framework enables it to adaptively optimize feature representations, making it a powerful alternative to traditional visual odometry methods.

4.3.2 Selected systems

For the benchmarking setup, we used the NVIDIA Jetson AGX Orin 64GB, a high-performance platform specifically designed for real-time AI and embedded applications. The Jetson AGX Orin features a 2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores and a 12-core ARM Cortex-A78AE CPU, running at 2.2 GHz. The system is equipped with 64GB of LPDDR5 memory and provides 275 TOPS (INT8) of AI performance.

The Jetson platform allows for efficient execution of SLAM algorithms, balancing high computational power and energy efficiency, making it ideal for real-time applications in resource-constrained embedded environments. On the software side, all algorithms were deployed using Docker to guarantee consistent and efficient testing across various approaches and datasets. This containerization enables performance and results to be compared under the same conditions, thus streamlining the testing process.

4.3.3 Evaluation metrics

The evaluation of SLAM algorithms is based on a set of well-defined metrics designed to assess both accuracy and computational efficiency. These metrics provide quantitative insight into the global alignment and local consistency of the estimated trajectories. Specifically, we utilize Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) to measure the discrepancy between the estimated and ground truth trajectories, capturing both overall accuracy and the drift over time. Additionally, Frames Per Second (FPS) is employed to evaluate the real-time performance of the system. To further assess computational efficiency, we monitor system resource usage, including CPU and GPU utilization, memory footprint, and power consumption. Detailed definitions and the methodology for computing these metrics can be found in the Appendix.

4.3.4 Dataset

For evaluation, we used the TUM RGB-D dataset [34], a widely recognized benchmark for SLAM systems. This dataset provides various indoor sequences captured using RGB-D cameras, which include both static and dynamic scenes. These sequences are ideal for testing the performance of SLAM algorithms in diverse and challenging real-world scenarios. To ensure diversity in our evaluation, we selected a specific subset of sequences that represent a wide range of environments, motions, and dynamics. The chosen sequences are as follows:

- **freiburg1_desk:** This sequence captures normal movements within a static office environment, making it suitable for evaluating the performance of SLAM algorithms in controlled, steady indoor settings.
- **freiburg1_xyz:** In this sequence, the camera undergoes translations along all three axes (X, Y, Z) while the environment remains static.

It evaluates the algorithm's ability to handle structured translational movements.

- **freiburg2_xyz:** This sequence involves rapid translations in a static indoor setting, challenging SLAM systems with faster motions, and requiring precise tracking in environments with minimal changes.
- **freiburg2_rpi:** The camera performs quick rotations around the roll, pitch, and yaw axes within an indoor space. This sequence stresses the system's ability to manage sudden rotational movements.
- **freiburg3_long_office_household:** A longer sequence set in a domestic environment with various objects and changing lighting conditions. This provides a complex, real-world scenario with longer-term tracking requirements and varying conditions.
- **freiburg3_walking_xyz:** This sequence captures rapid movements with significant translations and human dynamics, simulating more unpredictable and dynamic real-world conditions.
- **freiburg3_walking_static:** Featuring fast movements within a static environment, this sequence tests the robustness of SLAM algorithms when confronted with high-speed camera motion while the scene remains unchanged.

4.4 Experimentation

The experiments were conducted to analyze the performance of the selected SLAM algorithms under realistic conditions, with a focus on their accuracy, efficiency, and suitability for embedded platforms.

4.4.1 Performance evaluation

Our study begins with a visual comparison of the accuracy in trajectory estimation across the selected localization systems, using sequences from the TUM RGB-D dataset. Figure 4.3 shows the trajectories for the *freiburg3_structure_texture_far*. For TSformer, three configurations are used, considering 1, 2, or 3 images to predict the camera motion.

Table 4.1 & Table 4.2 summarize the average performance on selected sequences. Inference for models, whether trained end-to-end or integrated as a semantic thread, is performed using PyTorch (FP32), without utilizing TensorRT for performance optimization on NVIDIA hardware. VDO-SLAM demonstrates a balanced approach, achieving the best performance among

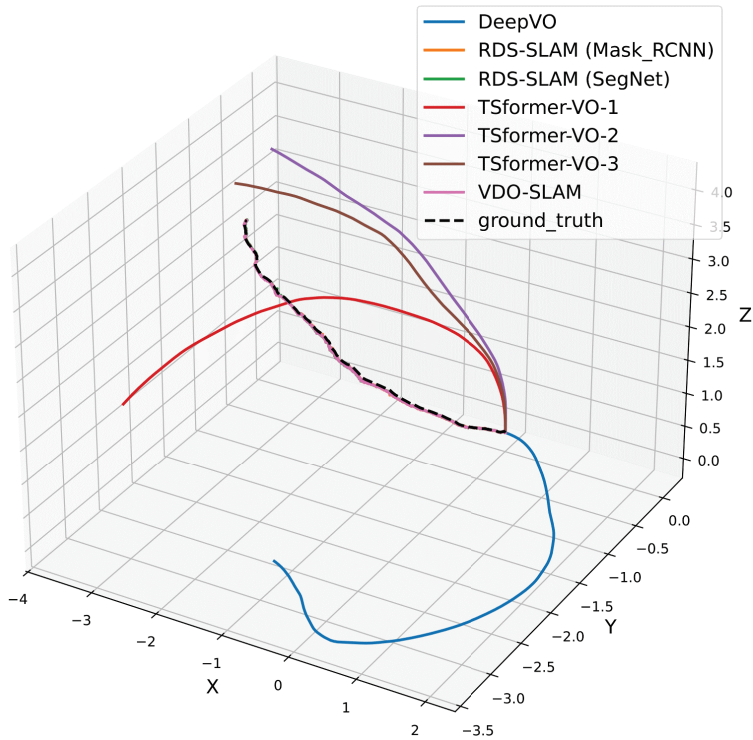

Figure 4.3 Trajectory predictions: each color denotes a different tested system.

Table 4.1 Performance Metrics: overall localization accuracy (ATE), Error between successive poses (RPE) and Inference Time

Methods	ATE (cm)	RPE (cm)	FPS
DeepVO	402.5	1.42	9.6
TSformer-VO-1	135.2	0.95	7.1
TSformer-VO-2	172.9	0.91	5.0
TSformer-VO-3	152.2	0.92	3.8
RDS SLAM (Mask RCNN)	3.4	0.96	3.2
RDS SLAM (SegNet)	3.3	1.00	7.5
VDO SLAM	3.4	1.00	8.1

the two SLAM systems tested, with an FPS of 8.1, reasonable energy consumption (∼10.85W), and a competitive ATE of 3.4 cm. However, a key factor driving VDO-SLAM's efficiency is that semantic segmentation is handled in a pre-processing step (0% in the SLAM pipeline itself). This approach allows the system to focus the bulk of its computational

Table 4.2 Resource Usage

Methods	CPU (%)	GPU (%)	RAM (%)	Power (mW)
DeepVO	20.99	46.45	15.93	14017.10
TSformer-VO-1	12.11	95.19	17.90	16881.53
TSformer-VO-2	10.65	96.77	17.98	16767.64
TSformer-VO-3	11.07	96.96	18.04	16637.78
RDS SLAM (Mask RCNN)	18.08	70.51	14.78	16000.26
RDS SLAM (SegNet)	16.26	60.20	13.49	14535.83
VDO SLAM	21.20	30.96	10.34	10852.83

resources on tracking (62.06%), while tasks such as mask generation (13. 47%) and map optimization (12.79%) consume significantly fewer resources (see Figure 4.4). Although this preprocessing strategy improves real-time performance during the execution of SLAM, it presents certain limitations. From a design perspective, offloading semantic segmentation and motion estimation to an earlier stage outside the main pipeline is not an ideal long-term solution. Ideally, these tasks should be integrated directly into the SLAM architecture to enable a more adaptive and responsive system that can adjust to new scenes in real time. By separating these processes from the SLAM pipeline, VDO-SLAM sacrifices flexibility, which could be a drawback in environments where on-the-fly processing of dynamic objects and scene changes is essential.

In contrast, the performance of RDS-SLAM is highly dependent on the choice of semantic segmentation model. When using Mask R-CNN, RDS-SLAM dedicates 51.64% of its computational resources to semantic segmentation, achieving high accuracy (ATE of 3.4 cm) but with a significant trade-off in real-time performance (3.2 FPS) and resource consumption (16W). This makes it less practical for power-sensitive applications. However, when SegNet is used, the performance of RDS-SLAM improves substantially, with an FPS of 7.5 and lower energy consumption (14.5W), making it comparable to VDO-SLAM. SegNet offers a lighter alternative that better balances the trade-off between accuracy and efficiency. Despite these improvements, both methods still lag behind true real-time performance, underscoring the challenge of optimizing SLAM systems for dynamic environments without excessive resource demands.

End-to-end deep learning-based approaches, such as DeepVO and TSformer-VO, exhibit significant limitations. Although this architecture promises to simplify the visual odometry pipeline, the results show that these approaches struggle in terms of accuracy and resource efficiency. For example, DeepVO achieves a poor ATE of 402.5 cm, reflecting significant drift

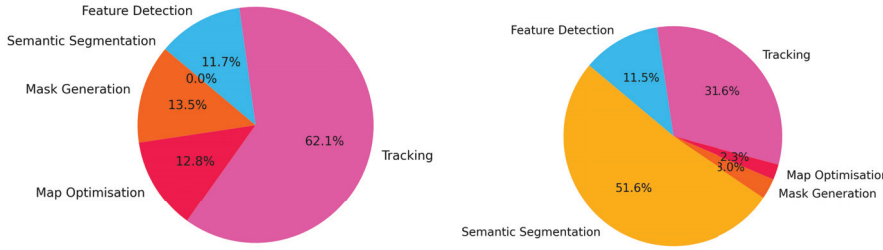

Figure 4.4 SLAM Block Execution Time Breakdown. Left: VDO-SLAM; Right: RDS-SLAM.

over time, particularly in environments that differ from the training datasets. Similarly, TSformer-VO, although slightly better in terms of accuracy, still performs poorly compared to geometric SLAM methods. More importantly, these end-to-end models suffer from extremely high GPU usage, with TSformer-VO consuming more than 95% of the available GPU resources. This excessive consumption of computational resources, coupled with limited accuracy, makes end-to-end approaches inefficient for real-world applications on embedded systems. Although they deliver higher FPS, they remain inadequate for real-time performance. Furthermore, their imprecision and inefficient resource utilization make them unsuitable for tasks that demand reliable localization.

4.4.2 TensorRT optimization for SLAM algorithms

Exporting the model to TensorRT is essential for running it on NVIDIA Jetson devices, as TensorRT is specifically designed to maximize the performance of deep learning models on NVIDIA GPUs. It achieves this by implementing optimizations such as precision calibration, kernel fusion, and layer fusion, which collectively improve inference speed and reduce memory usage. In our investigation into improving the efficiency of SLAM algorithms, we explored the possibility of optimizing the DeepVO PyTorch model using TensorRT. To accomplish this, the model was initially exported to the ONNX format, which serves as an intermediary representation that facilitates compatibility with TensorRT. Then, we evaluated the performance of the model using various precision formats, including FP32, FP16, and INT8, along with the quantization technique utilizing PQT (Post-Training Quantization). This allowed us to assess the trade-offs between model accuracy

and computational efficiency across different precision levels. The results of these optimizations, in terms of speedup compared to the original PyTorch implementation, are presented in Table 4.3. These findings indicate that the application of TensorRT to a deep learning model for VO can result in significant speed enhancements while maintaining minimal variations in prediction accuracy. Depending on the chosen precision level, it is possible to effectively balance speed and accuracy, allowing tailored performance based on specific application needs.

Table 4.3 Performance Comparison of DeepVO-pytorch Optimized with TensorRT

Precision Format	Speedup over PyTorch	ATE (cm)
FP32	4.36x	405.5
FP16	3.80x	411.9
INT8	4.23x	468.5

4.5 Conclusion

This work evaluated the performance of various localization systems, such as SLAM and VO, on resource-constrained devices, specifically the NVIDIA Jetson AGX Orin 64GB. By benchmarking systems such as RDS-SLAM, VDO-SLAM, DeepVO, and TSformer-VO, the study provided valuable insights into the efficiency and accuracy of these methods under constraints of computational power, memory, and energy resources. The results highlighted the trade-offs between real-time performance and accuracy. Although RDS-SLAM achieved higher accuracy, it incurred greater computational and energy demands. VDO-SLAM performs at a similar level in terms of performance; however, it is important to note that the CNN-based component, semantic extraction, and optical flow processing are conducted upstream on pre-known datasets. Despite this pre-processing step, which limits its adaptability to dynamic environments or new scenarios where the models have not been previously trained, VDO-SLAM still fails to achieve efficient real-time performance. On the other hand, deep learning-based models like DeepVO and TSformer-VO showed limitations in accuracy and high resource consumption, making them unsuitable for real-time applications.

The study emphasized the need for optimization, particularly in integrating semantic segmentation directly into the SLAM pipeline for greater flexibility. The findings suggest that there is significant potential for further optimizing deep learning-based approaches to improve their viability on resource-constrained platforms. Future research could focus on co-designing,

refining these algorithms, and exploring ways to balance accuracy and efficiency, while further tailoring them to the constraints of embedded hardware.

Acknowledgements

This work was conducted as part of a PhD thesis within the AI optimization roadmap at the LIAE laboratory of the DSCIN department at CEA-LIST.

References

[1] Y. Liu and J. Miura, "RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods," IEEE Access, vol. 9, pp. 23772–23785, 2021, doi: https://doi.org/10.1109/access.2021.3050617.

[2] J. Zhang, M. Henein, R. Mahony, and V. Ila, "VDO-SLAM: A Visual Dynamic Object-aware SLAM System," arXiv.org, 2020. https://arxiv.org/abs/2005.11052 (accessed Jan. 24, 2025).

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, doi: https://doi.org/10.1109/tro.2015.2463671.

[4] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018, doi: https://doi.org/10.1109/tro.2018.2853729.

[5] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A Research Platform for Visual-Inertial Estimation," May 2020, doi: https://doi.org/10.1109/icra40945.2020.9196524.

[6] R. Kang, J. Shi, X. Li, Y. Liu, and X. Liu, "DF-SLAM: A Deep-Learning Enhanced Visual SLAM System based on Deep Local Features," *arXiv.org*, 2019. https://www.semanticscholar.org/paper/DF-SLAM%3A-A-Deep-Learning-Enhanced-Visual-SLAM-based-Kang-Shi/6dd357e65acc5faae4e36c506553f51ec71777d2 (accessed Jan. 24, 2025).

[7] B. Bescos, C. Campos, J. D. Tardós, and J. Neira, "DynaSLAM II: Tightly-Coupled Multi-Object Tracking and SLAM," *arXiv.org*, 2020. https://arxiv.org/abs/2010.07820 (accessed Jan. 24, 2025).

[8] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison, "DeepFactors: Real-Time Probabilistic Dense Monocular SLAM," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 721–728, Apr. 2020, doi: https://doi.org/10.1109/lra.2020.2965415.

[9] S. Wang, R. Clark, H. Wen, and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks," *IEEE Xplore*, May 01, 2017. https://ieeexplore.ieee.org/document/7989236

[10] Françani, André O and M. Marcos, "Transformer-based model for monocular visual odometry: a video understanding approach," *arXiv.org*, 2023. https://arxiv.org/abs/2305.06121

[11] Z. Teed and J. Deng, "DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras," *arXiv:2108.10869 [cs]*, Feb. 2022, Available: https://arxiv.org/abs/2108.10869

[12] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit Mapping and Positioning in Real-Time," *arXiv.org*, 2021. https://arxiv.org/abs/2103.12352 (accessed Sep. 14, 2024).

[13] Z. Zhu *et al.*, "NICE-SLAM: Neural Implicit Scalable Encoding for SLAM," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, doi: https://doi.org/10.1109/cvpr52688.2022.01245.

[14] A. Rosinol, J. J. Leonard, and L. Carlone, "NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields," *arXiv.org*, Oct. 24, 2022. https://arxiv.org/abs/2210.13641

[15] D. Liao and W. Ai, "VI-NeRF-SLAM: a real-time visual–inertial SLAM with NeRF mapping," *Journal of Real-Time Image Processing*, vol. 21, no. 2, Feb. 2024, doi: https://doi.org/10.1007/s11554-023-01412-6.

[16] G. Li, Q. Chen, Y. Yan, and J. Pu, "EC-SLAM: Effectively Constrained Neural RGB-D SLAM with Sparse TSDF Encoding and Global Bundle Adjustment," *arXiv.org*, 2024. https://arxiv.org/abs/2404.13346 (accessed Jan. 24, 2025).

[17] Z. Zakaryaie Nejad and A. Hosseininaveh Ahmadabadian, "ARM-VO: an efficient monocular visual odometry for ground vehicles on ARM CPUs," *Machine Vision and Applications*, vol. 30, no. 6, pp. 1061–1070, May 2019, doi: https://doi.org/10.1007/s00138-019-01037-5.

[18] S. Bahnam, S. Pfeiffer, and G. C. H. E. de Croon, "Stereo Visual Inertial Odometry for Robots with Limited Computational Resources," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*

(IROS), pp. 9154–9159, Sep. 2021, doi: https://doi.org/10.1109/iros51168.2021.9636807.

[19] L. Xiao, J. Wang, X. Qiu, Z. Rong, and X. Zou, "Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment," *Robotics and Autonomous Systems*, vol. 117, pp. 1–16, Jul. 2019, doi: https://doi.org/10.1016/j.robot.2019.03.012.

[20] T. Ji, C. Wang, and L. Xie, "Towards Real-time Semantic RGB-D SLAM in Dynamic Environments," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11175–11181, May 2021, doi: https://doi.org/10.1109/ICRA48506.2021.9561743.

[21] HiIAmTzeKean, "GitHub - HiIAmTzeKean/Jetson-Nano-SLAM: A Nanyang Technological University research project. Multi-USB-Cam implementation on Jetson Nano.," *GitHub*, 2022. https://github.com/HiIAmTzeKean/Jetson-Nano-SLAM

[22] M. Ali, M. Erfani, and J. Mesbah, "Same App, Different App Stores: A Comparative Study," doi: https://doi.org/10.1145/1235.

[23] G. Lentaris, Ioannis Stamoulias, Dimitrios Soudris, and Manolis, "HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 8, pp. 1563–1577, Aug. 2016, doi: https://doi.org/10.1109/tcsvt.2015.2452781.

[24] D. Törtei Tertei, J. Piat, and M. Devy, "FPGA design of EKF block accelerator for 3D visual SLAM," *Computers & Electrical Engineering*, vol. 55, pp. 123–137, Oct. 2016, doi: https://doi.org/10.1016/j.compeleceng.2016.05.003.

[25] Z. Zhang, A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach," *DSpace@MIT (Massachusetts Institute of Technology)*, Jul. 2017, doi: https://doi.org/10.15607/rss.2017.xiii.028.

[26] J. Liu, X. Li, Y. Liu, and H. Chen, "RGB-D Inertial Odometry for a Resource-Restricted Robot in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9573–9580, Oct. 2022, doi: https://doi.org/10.1109/lra.2022.3191193.

[27] J. Kühne, M. Magno, and L. Benini, "Low Latency Visual Inertial Odometry with On-Sensor Accelerated Optical Flow for Resource-Constrained UAVs," *IEEE Sensors Journal*, pp. 1–1, 2024, doi: https://doi.org/10.1109/jsen.2024.3406948.

[28] T. Peng, D. Zhang, D. L. N. Hettiarachchi, and J. Loomis, "An Evaluation of Embedded GPU Systems for Visual SLAM Algorithms," *Electronic Imaging*, vol. 2020, no. 6, pp. 325–1325–6, Jan. 2020, doi: https://doi.org/10.2352/issn.2470-1173.2020.6.iriacv-074.

[29] S. Sumikura, M. Shibuya, and K. Sakurada, "OpenVSLAM: A Versatile Visual SLAM Framework," doi: https://doi.org/10.1145/3343031.3350539.

[30] X. Liu, Y. Yang, B. Xu, and S. Schwertfeger, "Benchmarking SLAM Algorithms in the Cloud: The SLAM Hive System," *arXiv.org*, 2024. https://arxiv.org/abs/2406.17586.

[31] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv.org*, 2017. https://arxiv.org/abs/1703.06870.

[32] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *arXiv:1511.00561 [cs]*, Oct. 2016, Available: https://arxiv.org/abs/1511.00561

[33] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, doi: https://doi.org/10.1109/cvpr.2018.00931.

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, doi: https://doi.org/10.1109/iros.2012.6385773.

[35] Z. Shan, R. Li, and S. Schwertfeger, "RGBD-Inertial Trajectory Estimation and Mapping for Ground Robots," *Sensors*, vol. 19, no. 10, p. 2251, May 2019, doi: https://doi.org/10.3390/s19102251.

4.6 Appendix

4.6.1 Calculation of the metrics used in evaluation

In order to evaluate the accuracy and performance of SLAM algorithms, we employ several key metrics that assess both the global alignment and local consistency of the estimated trajectories. The following metrics provide a comprehensive understanding of the system's behavior, including Absolute Trajectory Error (ATE), Relative Pose Error (RPE), and Frames Per Second (FPS). In addition, we monitor system resource usage using hardware

statistics on CPU, GPU, memory, and power consumption to evaluate the efficiency on embedded platforms.

4.6.1.1 Absolute Trajectory Error (ATE)

The Absolute Trajectory Error (ATE) is used to measure the global accuracy of the SLAM system by evaluating the difference between the predicted trajectory and the ground-truth trajectory after aligning them. Let the ground-truth poses be represented as a series of homogeneous transformation matrices $\mathbf{P}_i^{gt} \in SE(3)$ where *i* indexes the sequence frames, and let the estimated poses be $\mathbf{P}_i^{est}$. The ATE is computed as the root mean square error (RMSE) between the positions of the estimated trajectory and the ground truth after aligning them through a rigid body transformation.

The ATE is mathematically defined as:

$$\mathrm{ATE}_{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \| \mathbf{t}_i^{gt} - \mathbf{t}_i^{est} \|^2}, \tag{4.1}$$

where $\mathbf{t}_i^{gt}$ and $\mathbf{t}_i^{est}$ represent the translation vectors (positions) extracted from the ground-truth and estimated poses $\mathbf{P}_i^{gt}$ and $\mathbf{P}_i^{est}$, respectively. The RMSE measures the Euclidean distance between corresponding positions in both trajectories, giving a global measure of accuracy.

4.6.1.2 Relative Pose Error (RPE)

The Relative Pose Error (RPE) evaluates the local consistency of the trajectory by comparing the relative motion between consecutive poses in the estimated trajectory to that in the ground-truth trajectory. This metric is essential for assessing the short-term accuracy of the system in tracking small movements, which is critical in dynamic environments.

Given two consecutive ground-truth poses $\mathbf{P}_i^{gt}$ and $\mathbf{P}_{i+1}^{gt}$, the relative transformation between these poses is:

$$\mathbf{T}_i^{gt} = \left(\mathbf{P}_i^{gt}\right)^{-1} \mathbf{P}_{i+1}^{gt}. \tag{4.2}$$

Similarly, the relative transformation between consecutive estimated poses is:

$$\mathbf{T}_i^{est} = \left(\mathbf{P}_i^{est}\right)^{-1} \mathbf{P}_{i+1}^{est}. \tag{4.3}$$

The RPE is then computed as the difference between the relative transformations of the ground-truth and estimated poses. The translational RPE is

defined as the root mean square error (RMSE) of the translation vectors from the relative transformations:

$$\mathrm{RPE}_{trans} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \| \left(\mathbf{t}_i^{gt} - \mathbf{t}_{i+\triangle}^{gt}\right) - \left(\mathbf{t}_i^{est} - \mathbf{t}_{i+\triangle}^{est}\right) \|^2}, \tag{4.4}$$

where $\mathbf{t}_i^{gt}$ and $\mathbf{t}_i^{est}$ are the translation components of the relative transformations $\mathbf{T}_i^{gt}$ and $\mathbf{T}_i^{est}$ and $\triangle$ is the time interval.

For the rotational RPE, we measure the angular difference between the relative rotation matrices $\mathbf{R}_i^{gt}$ and $\mathbf{R}_i^{est}$, which can be quantified using the angle-axis representation. The rotational error is defined as:

$$\mathrm{RPE}_{rot} = \frac{1}{n}\sum\nolimits_{i=1}^{n} \arccos\left(\frac{\mathrm{tr}\left(\left(\mathbf{R}_i^{gt} - \mathbf{R}_{i+\triangle}^{gt\ \mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{R}_i^{est} - \mathbf{R}_{i+\triangle}^{est\ \mathrm{T}}\right)\right) - 1}{2}\right), \tag{4.5}$$

where $\mathbf{R}_i^{gt}$ and $\mathbf{R}_i^{est}$ represent the rotation matrices of the ground-truth and estimated poses, tr denotes the trace of a matrix, and $\triangle$ is the time interval. This metric provides the average rotational error over the trajectory.

4.6.1.3 Frames Per Second (FPS)

In real-time applications, the Frames Per Second (FPS) is a critical metric that measures the number of frames processed by the SLAM system per second. High FPS is essential for ensuring that the SLAM algorithm operates efficiently and in real time, especially in embedded or constrained environments where computational resources are limited. FPS can be computed as:

$$\mathrm{FPS} = \frac{\text{number of frames}}{\text{total processing time}}. \tag{4.6}$$

This metric helps assess the speed and responsiveness of the SLAM system.

4.6.2 Alignment methods

In this section, we describe four different trajectory alignment methods commonly used to compare predicted poses against ground-truth data in odometry and SLAM systems: scale, 6DOF, 7DOF, and 7DOF with scale. Each method focuses on optimizing different parameters (scale, rotation, and translation) to minimize the alignment error.

4.6.2.1 Scale alignment (scale)

The **scale** alignment adjusts only the global scale factor c between the predicted trajectory X and the ground-truth trajectory Y, without modifying the rotations or translations.

Method:

- The objective is to minimize the distance between X and Y by adjusting only the scale.
- The optimal scale factor c is computed as:

$$c = \frac{\sum (X \,.\, Y)}{\sum X^2}. \tag{4.7}$$

- The aligned trajectory is then scaled as:

$$X_{aligned} = \; c \,.\, X. \tag{4.8}$$

This method is particularly useful for monocular systems, where the scale is typically unknown and must be estimated post-facto.

4.6.2.2 6 Degress of Freedom (6DOF)

The **6DOF** alignment adjusts both the rotation and the translation between the predicted trajectory X and the ground-truth Y, but keeps the scale fixed. This allows the correction of orientation and position errors.

Method:

- The six degrees of freedom include three for rotation and three for translation.
- The alignment is based on the Singular Value Decomposition (*SVD*) of the covariance matrix between X and.
- The covariance matrix cov_{xy} is computed as:

$$\text{cov}_{xy} = \; \frac{1}{n} \sum_{i=1}^{n} (y_i - \text{mean}_y)(x_i - \text{mean}_x)^{\text{T}}. \tag{4.9}$$

- The SVD of cov_{xy} gives:

$$\text{cov}_{xy} = \; UDV^T. \tag{4.10}$$

  where U and V are orthogonal matrices representing the principal directions of Y and X, respectively, and D is a diagonal matrix of singular values.

- The rotation matrix *R* is computed as:

$$\mathrm{R} = U \sum V^T. \tag{4.11}$$

  where Σ is a diagonal matrix ensuring a proper right-handed coordinate system.
- The translation vector *t* is calculated as:

$$\mathrm{t} = \mathrm{mean}_y - R.\mathrm{mean}_x. \tag{4.12}$$

- The final aligned trajectory is:

$$X_{\mathrm{aligned}} = \mathrm{R} \,.\, \mathrm{X} + \mathrm{t}. \tag{4.13}$$

This method is ideal for LiDAR or stereo systems where scale is fixed but orientation and position errors need correction.

4.6.2.3 7 Degress of Freedom (7DOF)

The **7DOF** alignment extends the 6DOF method by also adjusting the scale factor *c*, in addition to the rotation and translation. This allows for simultaneous correction of scale, rotation, and translation errors.

Method:

- In addition to the rotation *R* and translation *t*, the scale factor *c* is computed to minimize the alignment error.
- The scale factor *c* is given by:

$$\mathrm{c} = \frac{1}{\sigma_x} \,.\, tr(D\Sigma), \tag{4.14}$$

  where σ_x is the variance of the points in *X*, and $tr(D\Sigma)$ is the trace of the product of the singular values and the diagonal matrix Σ.
- The transformed trajectory becomes:

$$X_{\mathrm{aligned}} = \mathrm{c} \,.\, \mathrm{R} \,.\, \mathrm{X} + \mathrm{t}. \tag{4.15}$$

This approach is beneficial in systems where the scale might not be exactly known, such as stereo or some lidar systems.

4.6.2.4 Scale + 7 Degrees of Freedom

The **scale_7DOF** alignment combines the benefits of the 7DOF alignment with an additional scale optimization step. After applying the rotation and

translation, the scale factor is optimized separately to further minimize the final trajectory error.

Method:

- First, perform a 6DOF alignment to adjust the rotation R and translation t.
- Then, optimize the scale factor c independently using the formula:

$$\mathrm{c} = \frac{\sum (X_{aligned} \cdot Y)}{\sum ({X_{aligned}}^2)}. \tag{4.16}$$

- The final trajectory becomes:

$$X_{final} = c \cdot X_{aligned}. \tag{4.17}$$

This method provides a finer correction of scale after the rotational and translational alignment, making it useful when significant scale variations exist between predicted and ground-truth trajectories.