LM-Droid SLAM: Modifying Droid SLAM with a Levenberg-Marquardt Solver

Parth Mahajan
mahajan.parth@northeastern.edu
NUID: 002301579

Utkarsh Rai
rai.ut@northeastern.edu
NUID: 002312222

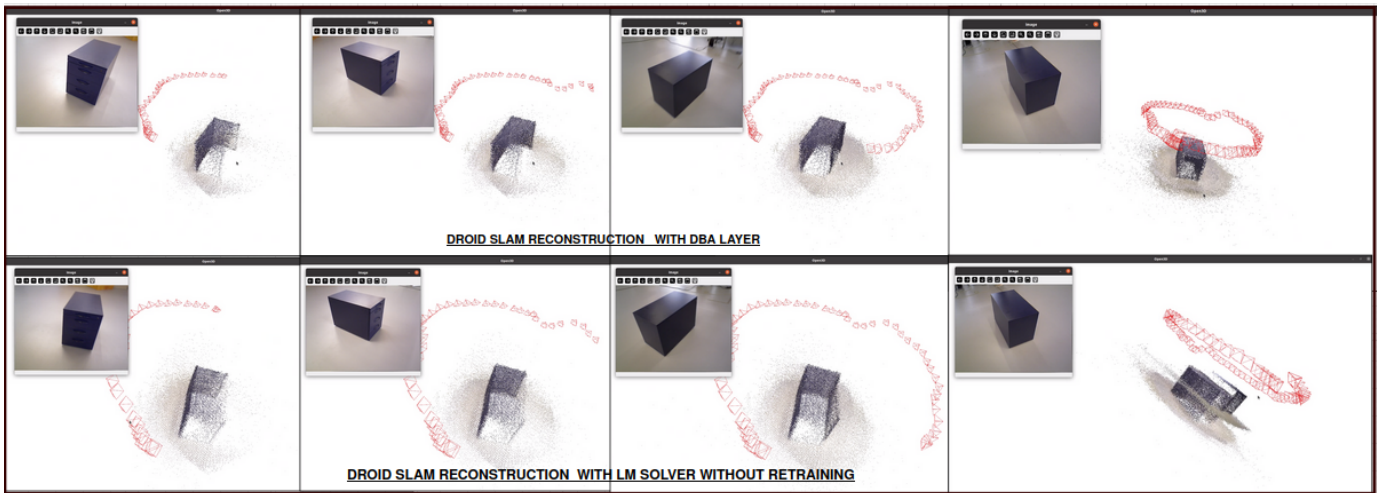

DROID SLAM RECONSTRUCTION WITH DBA LAYER
DROID SLAM RECONSTRUCTION WITH LM SOLVER WITHOUT RETRAINING

Figure 1: DROID-SLAM Structure from Motion(SfM) reconstruction of the same scene using two different approaches - with DBA layers(top), and with an LM solver(bottom).

Abstract

This project investigates the modification of DROID-SLAM, a cutting-edge visual SLAM system, by integrating a Levenberg-Marquardt (LM) solver in it. DROID-SLAM employs dense bundle adjustment (DBA) layers for camera pose and depth refinement. Replacing DBA layers with LM solver, known for robust handling of non-linear least squares problems, aims to enhance accuracy and stability while retaining the system's differentiable end-to-end architecture. The modified system is evaluated on TartanAir dataset, with performance metrics focusing on Absolute Trajectory Error (ATE) and profiling computational efficiency through detailed timing analysis. This study seeks to provide insights into overcoming challenges, addressing bottlenecks and advancing the capabilities of Droid SLAM systems.

1 Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental element for robotic systems, enabling them to navigate and build representations of their environments. SLAM plays a critical role in applications ranging from autonomous driving and drone navigation to augmented reality. As robots operate in increasingly complex and dynamic environments, the demand for robust, accurate, and computationally efficient SLAM solutions has grown significantly. Recent advancements in SLAM have shifted from traditional sparse feature-based methods to dense, learning-based approaches that leverage deep neural networks and advanced optimization techniques. Among these, DROID-SLAM [1] has emerged as a state-of-the-art framework, offering exceptional robustness and precision. By incorporating dense bundle adjustment (DBA) layers, DROID-SLAM optimizes camera poses and inverse depth estimates using dense image information. This dense optimization allows DROID-SLAM to outperform traditional sparse methods in challenging scenarios such as low-texture environments and dynamic scenes. Moreover, its differentiable architecture enables end-to-end learning, allowing the entire pipeline to adapt and improve through data-driven training. These innovations have resulted in significant improvements over prior methods across multiple datasets and modalities, including benchmark datasets like TartanAir[2].

Despite these advancements, current methods like DROID-SLAM face notable challenges. While dense optimization approaches improve accuracy, they also introduce computational bottlenecks. In particular, the learned DBA layers and context update operations are resource-intensive, significantly impacting processing time during both inference and training. This limits their scalability, particularly in scenarios with constrained computational resources. Furthermore, although end-to-end differentiable pipelines excel in learning context-specific features, they may lack the robustness of classical optimization algorithms for non-linear least squares problems, which are critical in refining pose and depth estimates. In this work, we propose integrating a Levenberg-Marquardt (LM) solver into the DROID-SLAM pipeline as a replacement for the learned DBA layers. The LM algorithm, widely regarded for its effectiveness in solving non-linear least squares problems, offers a promising solution to the limitations of current approaches. By incorporating LM into the pipeline, we aim to refine pose and inverse depth estimates more effectively. Through this project, we present the following analysis:

1) We evaluate the effect of substituting the DBA layer with an LM-based optimization technique, analyzing its impact on the Absolute Trajectory Error (ATE) to assess accuracy trade-offs.

2) Through detailed timing evaluations, we explore the computational efficiency of the system, highlighting critical bottlenecks and potential areas for optimization.

2 Literature Survey

Visual Simultaneous Localization and Mapping (SLAM) has seen significant advancements in recent years, with approaches evolving from classical geometry-based methods to deep learning-based techniques. Classical SLAM algorithms like MonoSLAM[3], PTAM[4], and ORB-SLAM[5] relied on hand-crafted features and geometric optimization to estimate camera poses and reconstruct 3D maps. These methods demonstrated robust performance in various environments but often struggled in challenging scenarios with low texture or dynamic objects. The introduction of deep learning to SLAM has opened new possibilities for improving robustness and accuracy. Early deep learning approaches like DeepVO[6] and UnDeepVO[7] used convolutional neural networks (CNNs) to estimate camera poses directly from image sequences. These methods showed promise in learning complex motion patterns but initially struggled to match the accuracy of classical approaches. More recent deep SLAM systems have sought to combine the strengths of both paradigms. DeepTAM[8] and D2VO[9], for instance, use learned components for feature extraction and matching while retaining geometric optimization for pose estimation. This hybrid approach has shown improved performance in challenging scenarios while maintaining the interpretability of classical methods. DROID-SLAM[10] represents a significant step forward, using a fully differentiable architecture that allows end-to-end training of the entire SLAM pipeline. By incorporating a learned dense bundle adjustment layer, DROID-SLAM achieves state-of-the-art accuracy across monocular, stereo, and RGB-D settings while demonstrating improved robustness to challenging motions and environments. Ongoing research focuses on developing more efficient architectures, improving the interpretability of learned components, and leveraging semantic information to enhance mapping and localization capabilities.

3 Methodology

3.1 Dense Bundle Adjustment Layer

The Dense Bundle Adjustment Layer (DBA) is a crucial component in Droid SLAM , designed to optimize camera poses and pixelwise depth estimates. It operates by mapping flow revisions to pose and depth updates across the entire frame graph. The DBA layer minimizes a cost function defined as:

$$E(G', d') = \sum_{(i,j)\in E} \left| p^* i - \Pi_c(G' j \circ \Pi^{-1} c(p_i, d' i)) \right|^2 \Sigma ij \tag{1}$$

where $\Sigma_{ij} = \text{diag}(w_{ij})$, and $|\cdot|_\Sigma$ denotes the Mahalanobis distance. This function aims to align reprojected points with revised correspondences predicted by the update operator. To solve this optimization problem, one can employ local parameterization for linearization and apply the Gauss-Newton algorithm. The system's structure allows for efficient solution using the Schur complement method:

$$\begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E}^T & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{\xi} \\ \Delta\mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} \tag{2}$$

$$\Delta\boldsymbol{\xi} = [\mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^T]^{-1}(\mathbf{v} - \mathbf{E}\mathbf{C}^{-1}\mathbf{w}) \tag{3}$$

$$\Delta\mathbf{d} = \mathbf{C}^{-1}(\mathbf{w} - \mathbf{E}^T\Delta\boldsymbol{\xi}) \tag{4}$$

The DBA layer is integrated into the computation graph, enabling backpropagation during training. This approach allows for end-to-end optimization of the SLAM system, enhancing its accuracy and robustness in various environments.

3.2 LM Solver Implementation

We implement a Levenberg-Marquardt algorithm, to remove the above dense bundle adjustment layer. The solver is designed to optimize both camera poses and inverse depth estimates simultaneously. Our approach involves formulating the objective function based on the same reprojection errors and computing the Jacobian matrix using automatic differentiation. We also use a damping factor adjustment mechanism and an iterative optimization process with backpropagation support.

3.3 Potential Strengths of the Proposed Approach

The DBA layer proposed in DROID-SLAM uses Schur's Complent and Gauss-Newton based update steps for geometrical grounding and optimization. There are several benefits that we anticipated for our choice of an LM solver over the current DBA layer:
One of LM's primary strengths is its robustness and ability to converge even from poor initial guesses. Unlike GN, which relies heavily on good starting points to achieve convergence, LM combines the robustness of gradient descent with the efficiency of the Gauss-Newton method. This hybrid approach ensures stable convergence across a wider range of scenarios. Additionally, LM prevents the issue of parameter evaporation, a phenomenon where parameters diverge toward infinity, further enhancing its reliability.

Another key advantage of LM is its adaptive behavior. The algorithm dynamically switches between gradient descent and Gauss-Newton methods depending on the local characteristics of the optimization problem. In regions with complex curvature, LM emphasizes stability by adopting a gradient descent approach, reducing the risk of overshooting or instability. Conversely, in well-behaved regions of the optimization landscape, it transitions to the Gauss-Newton method to achieve faster convergence. This adaptive mechanism allows LM to efficiently navigate diverse optimization landscapes, balancing stability and speed as needed.
Stability is another area where LM outperforms GN. The introduction of a damping factor (μ) into the Hessian matrix approximation addresses the ill-conditioning problems often faced by GN. This ensures that the matrix remains invertible throughout the optimization process, making LM more robust in handling poorly conditioned models. The damping mechanism also enhances numerical stability, allowing LM to perform effectively even in scenarios where the optimization problem involves challenging or unstable conditions.

3.4 Integration with DROID-SLAM Architecture

The LM solver is integrated into the DROID-SLAM pipeline by modifying the update operator architecture to ensure compatibility with back propagation, feature extraction, and context correlation modules. To isolate its impact, only the update operator was altered, leaving the ConvGRU block unchanged.

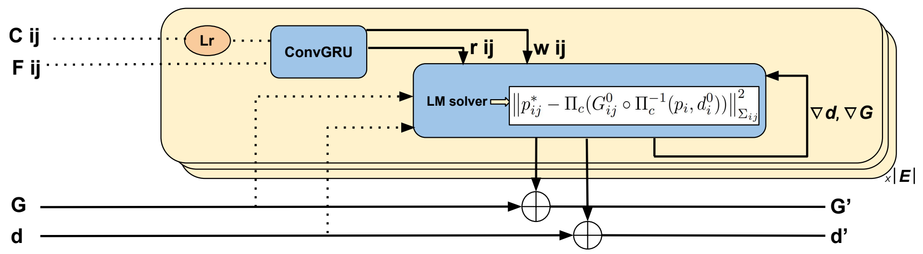

C ij
F ij
Lr
ConvGRU
r ij
w ij
LM solver
$\left\|p_{ij}^{*} - \Pi_c(G_{ij}^{0} \circ \Pi_c^{-1}(p_i, d_i^{0}))\right\|_{\Sigma_{ij}}^{2}$
$\nabla d, \nabla G$
x|E|
G
d
G'
d'

Figure 2: Our change to the DROID-SLAM [10] architecture by swapping the DBA layer with an LM solver. For our inference task, we perform a single Conv-GRU based update operation, and run the LM solver N times

3.5 Evaluation Framework

We develop a comprehensive evaluation framework to assess the performance of our modified DROID-SLAM system using TartanAir datasets: . The TartanAir dataset offers diverse synthetic environments with challenging conditions. Our evaluation framework includes benchmarking against the original DROID-SLAM and other state-of-the-art SLAM systems. We evaluate accuracy using standard metrics such as Absolute Trajectory Error (ATE).

4 Experiments

To evaluate the performance of our proposed modifications to DROID-SLAM, we conducted a series of experiments comparing the original DROID-SLAM implementation with our pytorch LM-solver based approach. Our evaluation was majority focused on two factors: Absolute Trajectory Error (ATE) and Timing analysis of various computational blocks of the DROID-SLAM pipeline.

4.1 Datasets

We utilized TartanAir dataset[2] for our experiments. The TartanAir dataset is a large-scale synthetic dataset specifically designed for visual SLAM, and we selected the `MH001, MH002` sequence, which is similar to the original DROID-SLAM benchmarks, for our tests. This sequence provides challenging synthetic environments characterized by varying lighting conditions and complex structures.

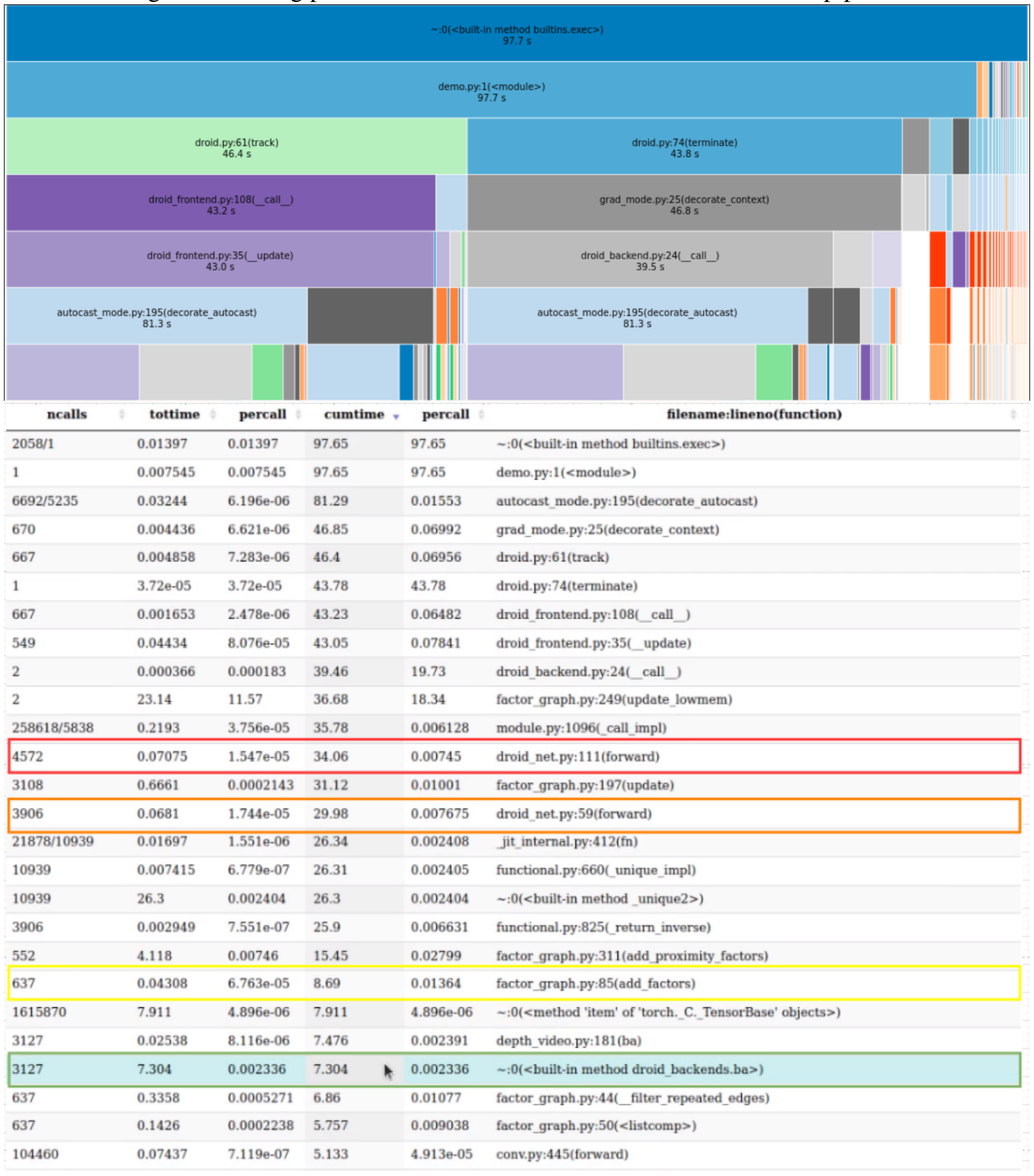
4.2 Absolute Trajectory Error (ATE) Comparison

To evaluate the accuracy of our configurations, we compared the Absolute Trajectory Error (ATE) for two setups: DROID-SLAM with DBA and DROID-SLAM with our LM-based layer, using a damping factor of $\lambda = 1e^{-3}$, the key frame threshold was chosen as 2.5m for both of the cases. For each configuration, the selected sequences were processed, and the ATE was computed against the ground truth trajectories. The results, summarized in Table 1, present ATE values for both subsets across all configurations. Additionally, we attempted to integrate Theseus as a differentiable LM solver into our model and retrain the entire pipeline—including the ConvGRU, feature network, and context network—end-to-end. However, this effort encountered significant resource constraints, as Theseus with gradient computation and optimization demanded over 90 GB of VRAM. This limitation, possibly due to a bug or misconfiguration, required extensive debugging and optimization efforts beyond the current timeline. We believe that resolving this issue and successfully retraining the pipeline with Theseus would yield improved results, providing a promising direction for future work.

Table 1: Absolute Trajectory Error (ATE) Comparison

Configuration	**TartanAir MHOO1 ATE (m)**	**TartanAir MHOO2 ATE (m)**
With DBA	0.05	0.04
With LM solver, without retraining	0.27	0.15

Figure 3: Timing profile of the relevant calls in the DROID-SLAM pipeline


~:0(<built-in method builtins.exec>)
97.7 s
demo.py:1(<module>)
97.7 s
droid.py:61(track)
46.4 s
droid.py:74(terminate)
43.8 s
droid_frontend.py:108(__call__)
43.2 s
grad_mode.py:25(decorate_context)
46.8 s
droid_frontend.py:35(__update)
43.0 s
droid_backend.py:24(__call__)
39.5 s
autocast_mode.py:195(decorate_autocast)
81.3 s
autocast_mode.py:195(decorate_autocast)
81.3 s

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
2058/1	0.01397	0.01397	97.65	97.65	~:0(<built-in method builtins.exec>)
1	0.007545	0.007545	97.65	97.65	demo.py:1(<module>)
6692/5235	0.03244	6.196e-06	81.29	0.01553	autocast_mode.py:195(decorate_autocast)
670	0.004436	6.621e-06	46.85	0.06992	grad_mode.py:25(decorate_context)
667	0.004858	7.283e-06	46.4	0.06956	droid.py:61(track)
1	3.72e-05	3.72e-05	43.78	43.78	droid.py:74(terminate)
667	0.001653	2.478e-06	43.23	0.06482	droid_frontend.py:108(__call__)
549	0.04434	8.076e-05	43.05	0.07841	droid_frontend.py:35(__update)
2	0.000366	0.000183	39.46	19.73	droid_backend.py:24(__call__)
2	23.14	11.57	36.68	18.34	factor_graph.py:249(update_lowmem)
258618/5838	0.2193	3.756e-05	35.78	0.006128	module.py:1096(_call_impl)
4572	0.07075	1.547e-05	34.06	0.00745	droid_net.py:111(forward)
3108	0.6661	0.0002143	31.12	0.01001	factor_graph.py:197(update)
3906	0.0681	1.744e-05	29.98	0.007675	droid_net.py:59(forward)
21878/10939	0.01697	1.551e-06	26.34	0.002408	_jit_internal.py:412(fn)
10939	0.007415	6.779e-07	26.31	0.002405	functional.py:660(_unique_impl)
10939	26.3	0.002404	26.3	0.002404	~:0(<built-in method _unique2>)
3906	0.002949	7.551e-07	25.9	0.006631	functional.py:825(_return_inverse)
552	4.118	0.00746	15.45	0.02799	factor_graph.py:311(add_proximity_factors)
637	0.04308	6.763e-05	8.69	0.01364	factor_graph.py:85(add_factors)
1615870	7.911	4.896e-06	7.911	4.896e-06	~:0(<method 'item' of 'torch._C._TensorBase' objects>)
3127	0.02538	8.116e-06	7.476	0.002391	depth_video.py:181(ba)
3127	7.304	0.002336	7.304	0.002336	~:0(<built-in method droid_backends.ba>)
637	0.3358	0.0005271	6.86	0.01077	factor_graph.py:44(__filter_repeated_edges)
637	0.1426	0.0002238	5.757	0.009038	factor_graph.py:50(<listcomp>)
104460	0.07437	7.119e-07	5.133	4.913e-05	conv.py:445(forward)

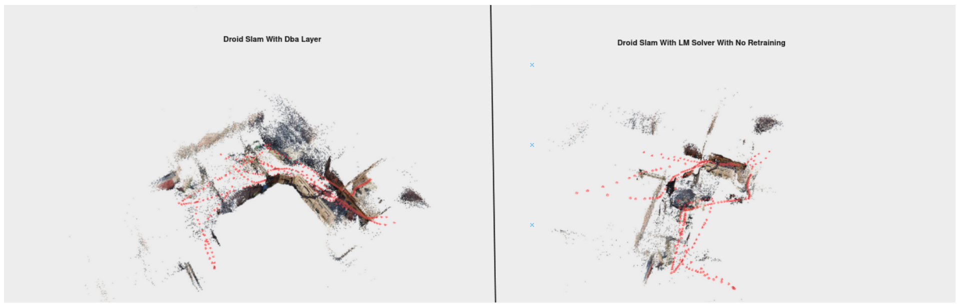

Droid Slam With Dba Layer
Droid Slam With LM Solver With No Retraining

Figure 4: SFM visualization with the DBA layer (left) and the LM solver (right) reveals a noticeable difference in pose estimation quality. The poses generated by the DROID-SLAM pipeline using the LM solver are subpar compared to those produced by the baseline model.

4.3 Timing Analysis

In addition to evaluating accuracy, we conducted a detailed timing analysis of the call stack using cProfiler on an abandoned factory scene containing 3,815 poses and approximately 550 key frames. The results of this analysis are presented in Fig. 3.

Our findings reveal that the bundle adjustment layer (highlighted in green in the figure) represents a notable performance bottleneck, even with utilizing CUDA-optimized code. (The Python implementation, in comparison, would exhibit even higher latency.) Cumulatively, the bundle adjustment layer requires 7.3 seconds, accounting for approximately 7.5% of the total processing time (97.65 seconds) in the pipeline for the scene.

Interestingly, the bundle adjustment layer is not the primary bottleneck. Instead, the graph aggregator method (highlighted in orange) accounts for 30.7% of the cumulative time, while the update operator (highlighted in red), which manages context vector updates through the GRU and local bundle adjustment, contributes nearly 35% of the total runtime.

These observations highlight a critical opportunity for optimization: adopting a more efficient approach to generate equivalent context embeddings while accelerating the update process. Implementing such improvements could significantly enhance the computational efficiency of the entire pipeline, reducing runtime and enabling faster performance without sacrificing accuracy.

Overall, these experiments aim to provide a comprehensive evaluation of our proposed modifications to DROID-SLAM by analyzing both accuracy and efficiency metrics across different datasets.

5 Conclusion

Our experiments lead us to 2 broad conclusions :

- Utilizing the LM solver during inference without retraining the conv-GRU and context and feature nets results in a degradation in performance compared to the baseline model. We hypothesize that training the entire network end-to-end with a differentiable LM layer would generate more robust context vectors and enhance the correlation between matching features. However, due to resource constraints—requiring 4 A100 GPUs and approximately 7 days to train on the Tartan Air dataset—this approach is deferred to future work.

- The graph aggregation network and the update operator are identified as the primary contributors to cumulative runtime during both inference and training. Enhancing the efficiency of the update network operations could be achieved by adopting models that leverage lighter architectures, thereby minimizing computational overhead and improving overall performance.

.

References

[1] Zachary Teed and Jia Deng. Droid-slam: deep visual slam for monocular, stereo, and rgb-d cameras. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2024. Curran Associates Inc.

[2] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian A. Scherer. Tartanair: A dataset to push the limits of visual SLAM. *CoRR*, abs/2003.14338, 2020.

[3] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[4] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pages 83–86. IEEE, 2009.

[5] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[6] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2043–2050. IEEE, 2017.

[7] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7286–7291. IEEE, 2018.

[8] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In *Proceedings of the European conference on computer vision (ECCV)*, pages 822–838, 2018.

[9] Qizeng Jia, Yuechuan Pu, Jingyu Chen, Junda Cheng, Chunyuan Liao, and Xin Yang. D 2 vo: Monocular deep direct visual odometry. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10158–10165. IEEE, 2020.

[10] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 34:16558–16569, 2021.