University of Hertfordshire

School of Engineering and Computer Science

MSc Artificial Intelligence and Robotics

7COM1039-0509-2023-Advanced Computer Science Masters Project

September 1, 2024

Beyond ORB: ALIKED for Enhanced Visual SLAM Performance

Name: Adwaith Kallungal Vrundavanan
Student ID: 22061390
Supervisor: Dr Zoe Jeffrey

MSc Final Project Declaration

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Artificial Intelligence and Robotics 7COM1039-0509-2023-Advanced Computer Science Masters Project at the University of Hertfordshire (UH).

It is my own work except where indicated in the report. I did not use human participants in my MSc Project. I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

Abstract

This research aims to evaluate whether ALIKED, a modern deep learning-based feature extractor, can effectively address the limitations of ORB in visual SLAM applications. As a pioneering study comparing ORB with the novel ALIKED feature extractor in the context of visual SLAM, it provides insights into performance improvements achievable under challenging conditions. Through a comparative analysis, the study assesses the performance of ORB and ALIKED in terms of keypoint replicability, detection consistency, and robustness in complex environments. The findings reveal that ALIKED significantly enhances robustness in low-light conditions compared to ORB, although its practical application in real-time SLAM requires further hardware-specific optimizations to balance speed with enhanced feature extraction performance. This research contributes to the field by demonstrating the comparative advantages of ALIKED over ORB, offering a pathway for future advancements in SLAM technology.

Acknowledgement

I would like to express my deepest gratitude to everyone who has supported me throughout the journey of completing this project.

First and foremost, I want to thank my mother for her unwavering encouragement and emotional support, which gave me the strength to persevere during challenging times. Her belief in me has been a constant source of motivation.

I am also immensely grateful to my friends, whose words of encouragement and companionship helped me stay positive and focused throughout this process. Their presence made the journey more enjoyable and manageable.

Lastly, I extend my sincere thanks to my professor, Dr Zoe Jeffrey, for their invaluable guidance, insightful feedback, and continuous support. Her expertise and mentorship were crucial in shaping this project and ensuring its successful completion.

Thank you all for being an integral part of this achievement.

Table of Contents

Table of Figures

Table of Tables

1 Introduction and Overview

1.1 Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in the field of robotics, where the goal is for a robot to build a map of an unknown environment while simultaneously keeping track of its own location within that environment (Thrun, 2002). This problem is crucial for enabling autonomous systems to navigate and operate effectively in real-world settings, especially in scenarios where GPS signals may be unavailable or unreliable. Over the past decade, SLAM has seen significant advancements, driven by the need for robust and accurate localization and mapping in various applications, ranging from autonomous vehicles to robotic exploration (Cadena et al., 2016).

In the domain of robotic vision, feature-based visual SLAM algorithms have become the dominant approach due to their efficiency, adaptability, and ability to function in diverse environments (Davison et al., 2007). These algorithms rely on identifying and tracking visual features within the environment to estimate both the map and the robot's position. Classic handcrafted visual features like SIFT (Lowe, 2004), ORB (Ethan, 2011), and LIFT (Yi et al., 2016) have been widely used for this purpose. However, these traditional methods often struggle to extract robust features in challenging environments, such as those with dynamic lighting, repetitive textures, or significant occlusions (Mur-Artal and Tardós, 2017).

The limitations of ORB feature extraction, in particular, have been noted by researchers like Mur-Artal and Tardós (2017) and Shi et al. (2020). These limitations can adversely affect the SLAM system's ability to accurately relocalize in scenes with substantial variations, leading to reduced performance in real-world applications.

Recent advancements in deep learning have introduced pixel-wise feature extractors that offer superior robustness under challenging conditions, marking a significant shift from traditional, handcrafted feature extraction methods. Traditional techniques, such as SIFT (Lowe, 2004) and ORB (Ethan, 2011), rely on detecting keypoints based on predefined patterns, which can struggle in environments with dynamic lighting, repetitive textures, or heavy occlusions. To address these limitations, deep learning-based methods have emerged, capable of learning features directly from data, thus providing greater adaptability and robustness.

Notable among these modern approaches is the SuperPoint network proposed by DeTone, Malisiewicz, and Rabinovich (2018). SuperPoint employs a self-supervised training method to detect and describe keypoints, leading to improved performance in a variety of visual tasks. This method demonstrated how deep learning could be applied to produce more reliable feature extraction in real-time SLAM scenarios, especially in environments where traditional methods fall short.

Building on this foundation, Dusmanu et al. (2019) introduced D2-Net, a trainable Convolutional Neural Network (CNN) designed for joint detection and description of local features. D2-Net further demonstrated that deep learning models could outperform traditional feature extractors by learning more discriminative and robust features, even in scenes with complex variations.

Tang et al. (2019) contributed to this advancement by proposing LF-Net, which learns local features from images through a novel end-to-end framework. LF-Net optimizes the detection

and description processes simultaneously, ensuring that the extracted features are not only robust but also computationally efficient, making them suitable for real-time applications.

In this context, the lightweight deep learning network ALIKED, developed by Zhao et al. (2023), has shown particular promise. ALIKED (Zhao et al., 2023) is designed for keypoint detection and descriptor extraction, combining efficiency with effectiveness. Its lightweight architecture allows it to be deployed in resource-constrained environments while still delivering high performance. The network has been proven effective in various visual measurement tasks, suggesting its potential to enhance SLAM systems' feature extraction capabilities by offering a more adaptable and resilient alternative to traditional methods like ORB.

This research explores whether ALIKED (Zhao et al., 2023) can serve as a better feature extractor than ORB in challenging scenarios. By conducting a comparative analysis of the ORB and ALIKED feature extractors, the study aims to provide insights into how well ALIKED might perform in situations where ORB features struggle.

With the advancements and challenges in SLAM technology outlined, the following section details the specific aims of this research, focusing on evaluating modern feature extractors in addressing the limitations of traditional methods like ORB.

1.2 Research Aim

The aim of this research is to evaluate and compare the effectiveness of the ALIKED feature extractor against the traditional ORB feature extractor in the context of visual SLAM. Specifically, this study seeks to:

1. **Compare ORB and ALIKED feature extractors**: Evaluate the performance of ORB and ALIKED in terms of keypoint replicability, detection consistency, and robustness to challenging environments.
2. **Optimize ALIKED for portability and efficiency**: Explore techniques to improve ALIKED's computational efficiency and make it more suitable for deployment on resource-constrained devices.

By achieving these objectives, the research aims to provide valuable insights into the relative strengths of modern deep learning-based feature extractors compared to traditional methods and offer a comprehensive performance evaluation.

1.3 Research Objectives

The objective of the project is to compare the performance of ORB and ALIKED feature extractors. The project aims to achieve the following objectives:

- Conduct a comprehensive literature review on visual SLAM and feature extraction.
- Identify appropriate datasets for evaluating feature extractors in the context of SLAM.
- Evaluate and Compare the performance of ORB and ALIKED feature extractors.
- Optimize ALIKED for portability and efficiency to reduce its computational complexity and make it compatible with a wider range of platforms like tensorflow, huggingface optimum, mathlab, go and C++.

1.4 Project Plan

The project is divided into Six main phases:

1. **Literature Review:** This initial phase involves a comprehensive review of existing literature to understand the current state-of-the-art in Visual SLAM systems and feature extractors. This review will focus on recent advancements and methodologies in these fields to establish a foundation for subsequent analysis.

2. **Dataset Acquisition:** In this phase, relevant datasets for benchmarking the SLAM system will be identified and collected. Specifically, the benchmark datasets provided by Burri et al. (2016) will be utilized for this project. These datasets will serve as a standard for evaluating the performance of the feature extractors and SLAM system.

3. **Analysis of ORB Feature Extractor:** A thorough analysis of the ORB (Ethan, 2011) feature extractor will be conducted to assess its strengths and limitations. Special attention will be given to its performance under challenging conditions, such as low-texture and feature-sparse environments. This phase aims to delineate ORB's efficacy in diverse scenarios.

4. **Setup of ALIKED Feature Extractor:** This phase will focus on the setup and configuration of the ALIKED (Zhao et al., 2023) feature extractor. The goal is to ensure that ALIKED operates correctly and meets the expected functionality. Successful implementation will provide a baseline for the subsequent evaluation of ALIKED's performance.

5. **Analysis of ALIKED Feature Extractor:** In this phase, the performance of ALIKED will be benchmarked against ORB. This comparison will involve evaluating keypoint detection, matching accuracy, and overall reliability, particularly in challenging environments. Additionally, different optimized variants of ALIKED will be assessed to determine their suitability for real-time applications.

6. **Optimization of ALIKED Feature Extractor:** The final phase will concentrate on optimizing the ALIKED feature extractor. This involves improving its inference speed, computational efficiency, and portability while maintaining or enhancing its original performance metrics. The objective is to enhance ALIKED's practicality for real-time use and broader applicability.

A detailed breakdown of the project timeline is provided in the Gantt chart (Appendix A).

1.5 Tools and Techniques

To achieve the objectives outlined in the project plan, it is essential to employ a robust set of tools and techniques tailored for both traditional and deep learning-based SLAM systems. The following section details the specific tools and technologies utilized in this research.

The project will use the following tools and techniques:

- **pytorch** (Paszke et al., 2019): Pytorch is a deeplearning framework focused on usability and speed. The ALIKED model is built on top of the pytorch framework.

- **ONNX** (Bai, Lu, and Zhang, 2019): The ALIKED model was converted to the Open Neural Network Exchange (ONNX) format for interoperability and optimization across different platforms.
- **TensorRT** (NVIDIA, 2021): To accelerate inference, the ONNX model was optimized using NVIDIA TensorRT, a high-performance deep learning inference platform, resulting in significant performance improvements on NVIDIA GPUs.
- **OpenCV** (Bradski and Kaehler, 2000): Leveraged OpenCV for processing the image before being passed into the ALIKED Network and for the implementation of the ORB Feature Extractor.
- **DBoW2** (Gálvez-López and Tardós, 2012): Implemented DBoW2, a library for creating and utilizing bag of words representations, which is used in ORB-SLAM3 for loop detection and relocalization. This helps in recognizing previously seen places to correct drift in the SLAM system.
- **g2o** (Kümmerle et al., 2011): Utilized g2o, a C++ framework for optimizing graph based nonlinear error functions, which is employed for optimizing the pose graph and performing bundle adjustment to refine the SLAM results.
- **Sophus** (Schneider and GAP Team, 2022): This is a C++ library integrated into ORB-SLAM3 for efficient representation and manipulation of 3D rotations and rigid body transformations. This compact representation is crucial for accurate and real-time estimation of camera poses and feature tracking within the SLAM framework.
- **evo** (Grupp, 2017): Python package for the evaluation of odometry and SLAM.
- **Eigen3** (Guennebaud and Jacob, 2010): Used Eigen3, a C++ template library for linear algebra, which provides efficient matrix and vector operations.

1.6 Ethical, Legal, Professional, and Social Issues

This project investigates the performance of ORB and ALIKED feature extractors. While this research holds promise for advancements in SLAM technology, it's crucial to consider the following ethical, legal, social, and professional issues:

1.6.1 Ethical Issue

Privacy Concerns: The use of feature extractors like ORB and ALIKED involves analyzing visual data captured from cameras, which could potentially include environments where individuals are present. To address privacy concerns, this project exclusively utilizes publicly available datasets (Burri et al., 2016) that comply with strict privacy and data handling protocols. These datasets are carefully curated to ensure the exclusion of any personally identifiable information, minimizing privacy risks during experimentation and analysis.

1.6.2 Legal Issues

Intellectual Property: The ORB feature extractor is part of OpenCV and is freely available under a BSD 3-Clause License, allowing for wide use and modification with proper attribution. ALIKED, also utilizing a BSD 3-Clause License, follows similar permissive terms. The project adheres to these licensing requirements, ensuring all modifications and results derived from the use of these feature extractors are properly attributed and compliant with the respective licenses.

Data Privacy Regulations: We are using publicly available datasets which adhere to the Data Privacy regulations of GDPR. These datasets have been used to bench mark various other visual SLAM Algorithms (Campos et al., 2021; Li et al., 2020; Yin et al., 2023).

1.6.3 Social Issues

Job displacement due to automation: Enhancements in feature extraction techniques may contribute to increased automation in areas such as robotics and surveillance, potentially impacting jobs traditionally performed by humans. The project acknowledges these implications and advocates for a balanced approach, including discussions on reskilling and transitioning workers into new roles within the evolving technological landscape.

Public acceptance of Technology: The integration of advanced feature extraction methods in public facing applications might raise concerns about safety, privacy, and social acceptance. To address these concerns, transparent communication regarding the purposes, benefits, and privacy safeguards associated with these technologies is essential in building public trust.

1.6.4 Professional Issues

Standards and Best Practices: This project adheres to established standards and best practices in computer vision and machine learning to ensure the reliability, safety, and interoperability of the research outcomes. By following a structured project plan, including rigorous testing and evaluation protocols, the project maintains a high standard of professional integrity and contributes to the advancement of the field.

2 Literature Review

2.1 Introduction to SLAM and Visual SLAM

Simultaneous Localization and Mapping (SLAM) is a core technology in robotics, enabling an autonomous system to build a map of an unknown environment while simultaneously tracking its location within that map. This capability is essential for a wide range of applications, from autonomous vehicles to drones and robotics, where real-time mapping and localization are critical. The concept of SLAM was first introduced by Smith, Self, and Cheeseman in 1986, laying the foundation for its development in the robotics field (Cheeseman, Smith, and Self, 1987). Over the years, SLAM has evolved, incorporating various sensing modalities to enhance accuracy and robustness in different environments.

Visual SLAM extends the traditional SLAM paradigm by using visual information from cameras to enhance both the mapping and localization processes. Unlike other SLAM systems that rely on range sensors like LiDAR, Visual SLAM utilizes image data to provide rich environmental details. This approach is often more cost-effective and provides a higher level of detail, which is beneficial for tasks requiring fine-grained environmental understanding. Visual SLAM systems are generally categorized into monocular, stereo, and RGB-D systems, each offering unique advantages depending on the application.

2.2 Evolution of Visual SLAM Systems

Visual SLAM has undergone significant evolution since its inception, driven by advancements in both computer vision and robotics. Visual SLAM systems enable robots and devices to navigate and map environments using visual inputs from cameras. This capability is fundamental for applications ranging from autonomous vehicles to augmented reality. This section provides an overview of key developments and milestones in the evolution of Visual SLAM systems.

2.2.1 Extended Kalman Filter Based Method

2.2.1.1 MonoSLAM

The origins of Visual SLAM can be traced back to probabilistic methods that relied on extracting distinct features from images to estimate camera motion and build maps. One of the earliest and most influential feature-based SLAM systems, MonoSLAM was introduced by Davison et al. (2007), who demonstrated real-time SLAM using a single camera in small-scale environments.

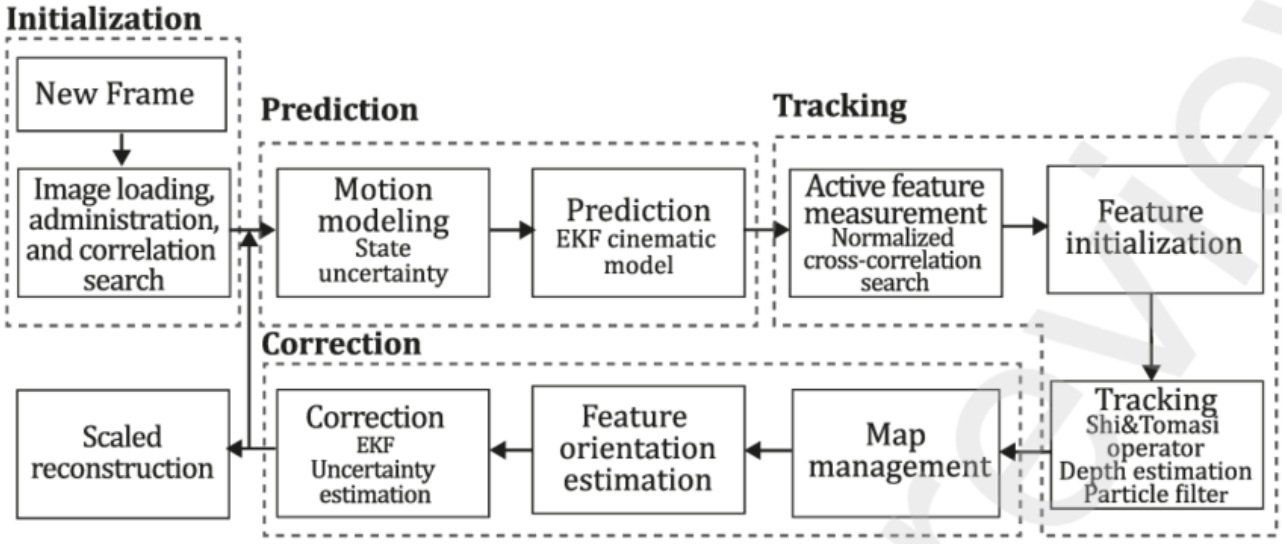

Figure 2.1: System Architecture of MONOSLAM (Herrera-Granda, Torres-Cantero, and Peluffo-Ordoñez, 2023)

As shown in Figure 2.1, MonoSLAM's key components include feature extraction, feature matching, pose estimation, map update, and the use of an Extended Kalman Filter (Kalman,

1960) to maintain a probabilistic representation of the camera pose and the 3D map. The FAST (Viswanathan, 2009) and ORB (Ethan, 2011) algorithms are employed for feature detection and description, while a brute-force matching approach is used to identify corresponding features between consecutive frames. This work laid the foundation for the development of monocular SLAM systems, emphasizing the importance of robust feature detection and tracking.

While MonoSLAM has made significant contributions, it also has limitations. It is unable to determine the absolute scale of the reconstructed map without additional information and may struggle in challenging environments. Despite these limitations, MonoSLAM remains a valuable benchmark and has influenced the development of more advanced monocular SLAM algorithms.

2.2.1.2 MSCKF

To address the limitations of MonoSLAM, Klein and Murray (2007) explored alternative approaches that incorporate additional information or constraints to improve accuracy and robustness. One such approach is the use of a multi-state constraint Kalman filter (MSCKF) for vision-aided inertial navigation.

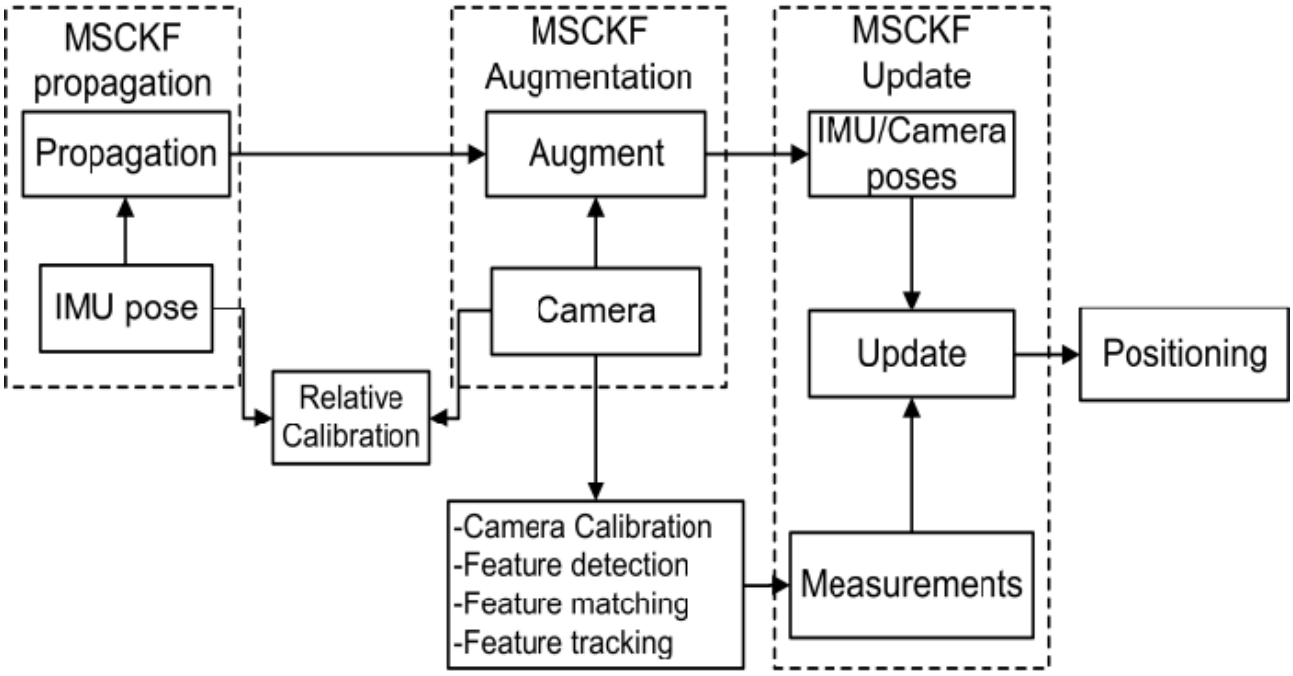

Figure 2.2: System Architecture of MSCKF (Ramezani et al., 2017)

From the Figure 2.2, we can see that the MSCKF extends the standard Kalman filter by introducing multiple state estimates for the camera pose and the landmarks. This allows the filter to maintain a more accurate representation of the system state and to handle the scale ambiguity inherent in monocular SLAM. By incorporating inertial measurements from an Inertial Measurement Unit (IMU), the MSCKF can also improve the accuracy and consistency of the estimated camera pose.

The MSCKF works by propagating multiple state estimates through the nonlinear dynamics of the system and updating them using the available measurements. Constraints are imposed

on the relative relationships between the different state estimates to ensure consistency and to resolve the scale ambiguity. This approach can lead to improved accuracy and robustness compared to traditional monocular SLAM algorithms.

In addition to addressing the scale ambiguity, the MSCKF can also help to mitigate the effects of drift that can occur in monocular SLAM due to accumulated errors. By incorporating inertial measurements and using constraints to maintain consistency, the MSCKF can reduce the accumulation of errors and improve the overall accuracy of the estimated camera pose and map.

2.2.2 Graph Based Methods

2.2.2.1 PTAM

One of the early graph based SLAM systems was Parallel Tracking and Mapping (PTAM), developed by Klein and Murray (2007), was a groundbreaking advancement in the field of monocular SLAM. It introduced a parallel processing framework that significantly improved the real-time performance of SLAM systems.

Unlike previous approaches that relied on sequential processing, PTAM divides the SLAM pipeline into two parallel threads: tracking and mapping. The tracking thread is responsible for estimating the current camera pose relative to the existing map, while the mapping thread updates the map based on the estimated camera poses and detected features. This parallel execution allows for more efficient utilization of computational resources, enabling real-time performance on a wide range of hardware.

One of the key innovations in PTAM was the use of a feature based bundle adjustment algorithm for map optimization. Bundle adjustment is a non-linear optimization technique that simultaneously refines the camera poses and 3D map points to minimize the reprojection error. By parallelizing bundle adjustment, PTAM was able to achieve significant improvements in map accuracy and consistency.

Another important aspect of PTAM was its ability to handle loop closures. When the system detects that the camera has returned to a previously visited location, it can use the existing map information to correct for accumulated drift and improve the overall consistency of the reconstructed environment.

PTAM's parallel processing framework and feature-based bundle adjustment have had a significant impact on the development of subsequent SLAM algorithms. Its approach has been adopted and extended in many modern SLAM systems, demonstrating the effectiveness of parallel processing for improving real-time performance and accuracy.

2.2.2.2 ORB-SLAM

After PTAM's significant contributions, the field of monocular SLAM continued to evolve. One notable advancement was ORB-SLAM (Mur-Artal, Montiel, and Tardos, 2015), which built upon PTAM's foundation and introduced several key improvements over its predecessors, enhancing its versatility and accuracy. One of the most notable features is its use of ORB (Ethan, 2011) features, which are more rotation and scale invariant than those used in previous systems, making it suitable for challenging environments.

ORB-SLAM also incorporates place recognition provided by bag of words (Gálvez-López and Tardós, 2012) to efficiently detect loop closures, improving the overall consistency of the reconstructed map. Additionally, it employs a covisibility graph to focus tracking and mapping efforts on a local area, enabling real-time operation in large environments.

A key advantage of ORB-SLAM is its unified use of ORB features for all tasks, including tracking, mapping, relocalization, and loop closing. This simplifies the system and enhances its efficiency. The system also includes real-time camera relocalization with significant invariance to viewpoint and illumination, allowing for recovery from tracking failures.

Furthermore, ORB-SLAM features an automatic and robust initialization procedure based on model selection, enabling the creation of initial maps for both planar and nonplanar scenes. It also employs a "survival of the fittest" approach to keyframe selection, improving tracking robustness and enabling lifelong operation by discarding redundant keyframes.

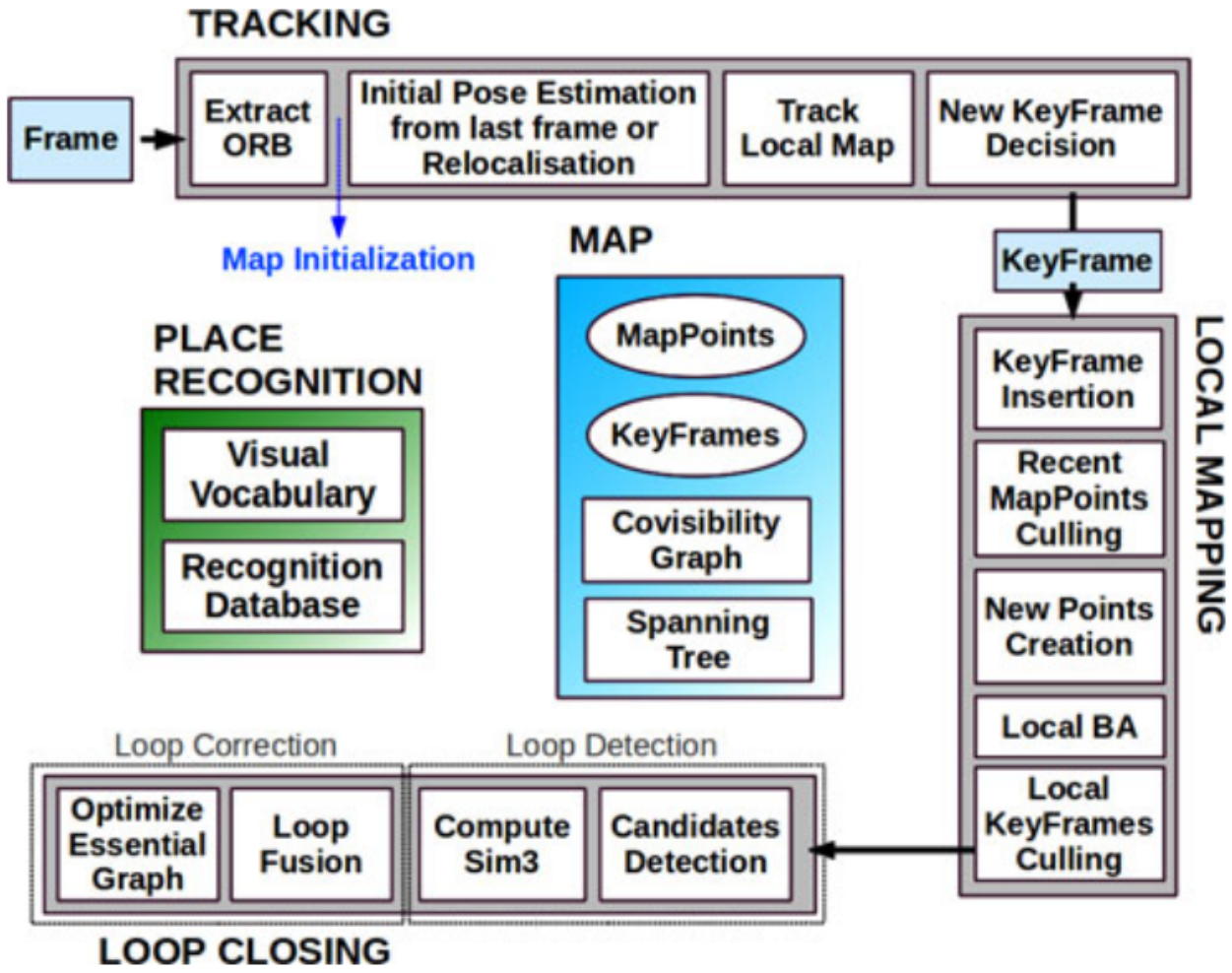

Figure 2.3: System Architecture of ORB-SLAM (Mur-Artal, Montiel, and Tardos, 2015)

Another critical aspect of ORB-SLAM is its architecture, which operates through three parallel threads: Tracking, Local Mapping, and Loop Closing, as depicted in Figure 2.3. The Tracking thread is responsible for estimating the camera pose and deciding when to insert a new keyframe. The Local Mapping thread manages the creation and optimization of the local map

by adding new keyframes and removing redundant ones. The Loop Closing thread detects loops and performs global optimization to reduce drift, ensuring the consistency of the entire map. This multithreaded architecture allows ORB-SLAM to efficiently handle complex tasks simultaneously, contributing to its real-time performance and robustness in diverse environments.

2.2.2.3 ORB-SLAM2

ORB-SLAM2 (Mur-Artal and Tardós, 2017) marks a significant advancement in Visual SLAM by being the first open-source system to support monocular, stereo, and RGB-D cameras. It builds upon the foundation laid by its predecessor, ORB-SLAM (Mur-Artal, Montiel, and Tardos, 2015), while incorporating several critical enhancements. The system is particularly notable for integrating loop closing, relocalization, and map reuse capabilities, which are crucial for robust SLAM performance in diverse environments. These features allow the system to revisit previously mapped areas, correct drift, and continue operation seamlessly even after tracking failures.

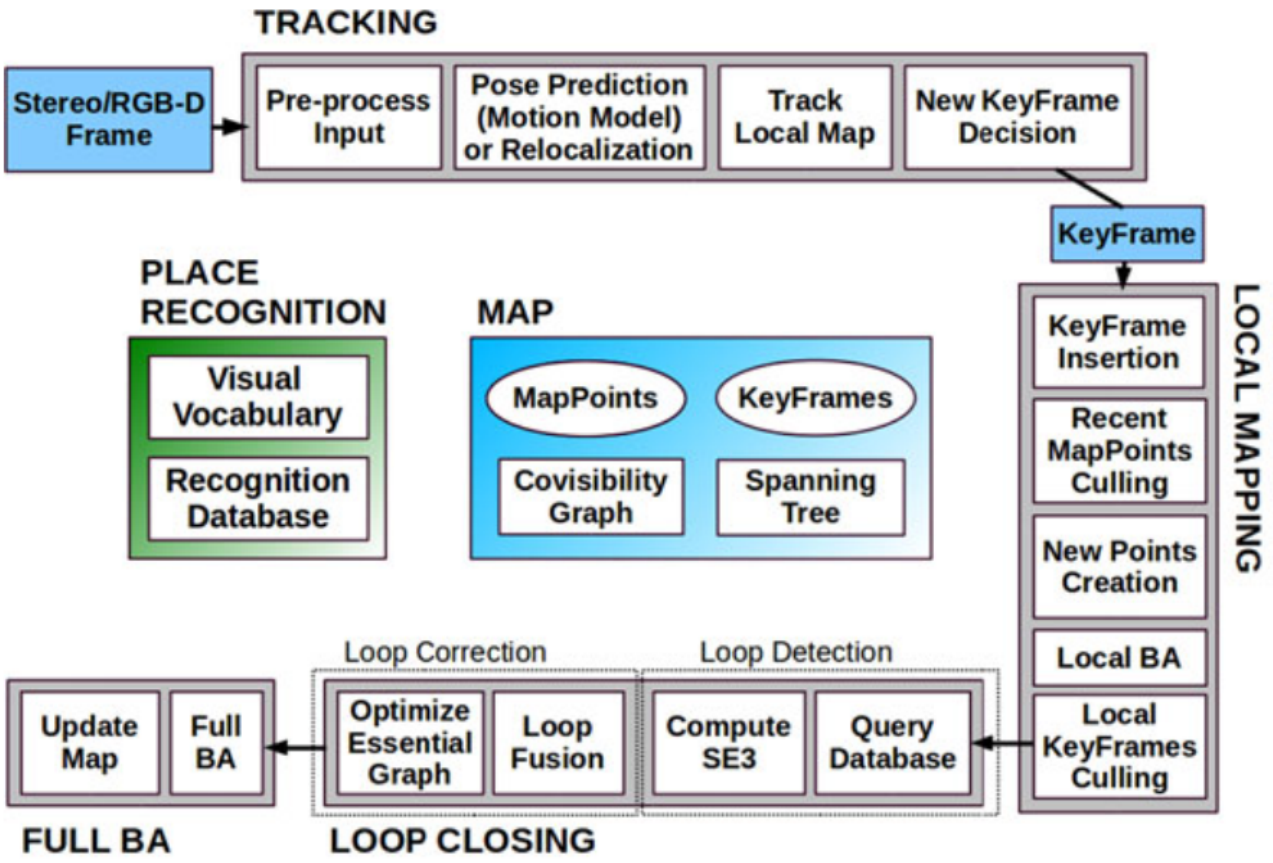

Figure 2.4: System Architecture of ORB-SLAM2 (Mur-Artal and Tardós, 2017)

Architecturally, ORB-SLAM2 (Mur-Artal and Tardós, 2017) is very similar to ORB-SLAM (Mur-Artal, Montiel, and Tardos, 2015), except for the Tracking thread, where a preprocessing step has been added to handle the new types of inputs, as illustrated in Figure 2.4. This preprocessing step, detailed in Figure 2.5, is critical for adapting the system to work with monocular, stereo, and RGB-D cameras by appropriately managing the different sensor modalities before

they are passed to the Tracking thread. This modification allows ORB-SLAM2 to effectively handle the increased complexity introduced by these additional input types, ensuring robust and accurate SLAM performance across various camera configurations.

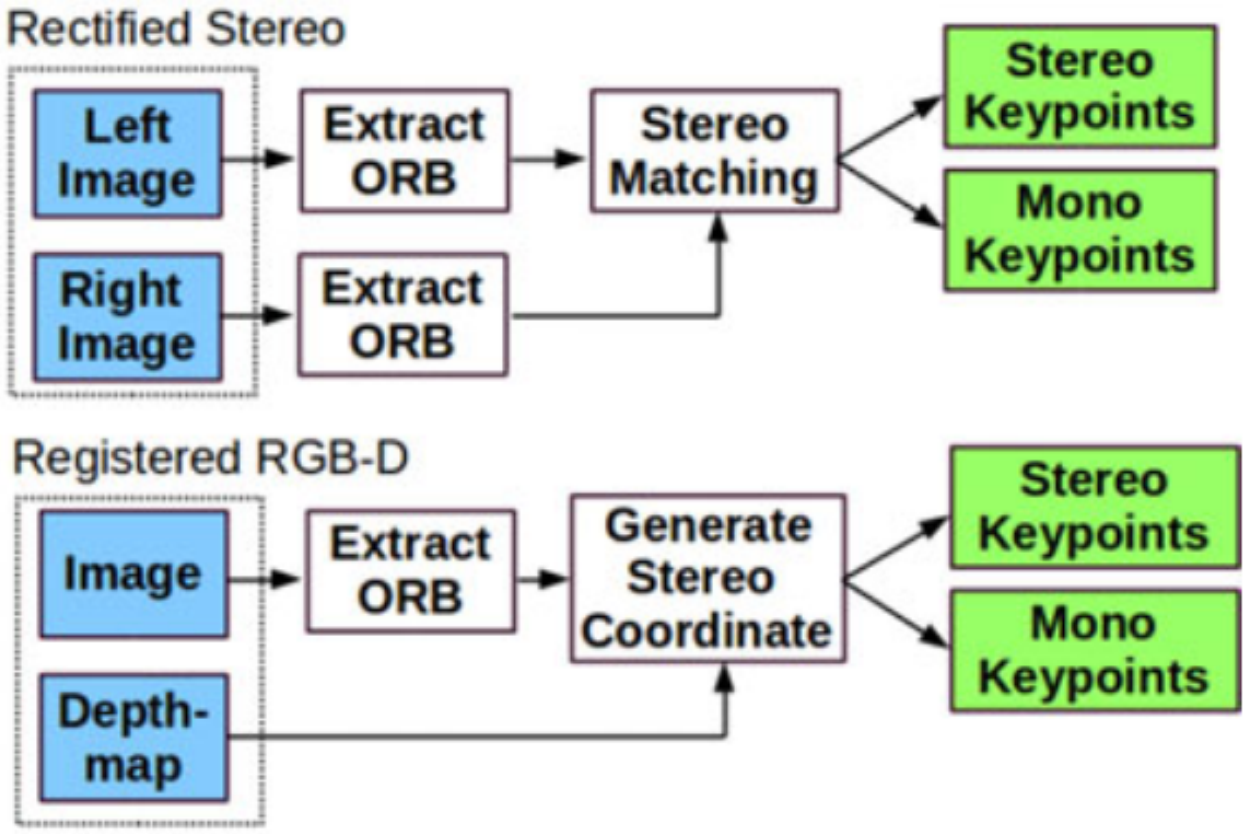

Figure 2.5: Input preprocessing of ORB-SLAM2 (Mur-Artal and Tardós, 2017)

One of the key improvements in ORB-SLAM2 is its ability to utilize bundle adjustment (BA) with RGB-D cameras, significantly enhancing accuracy compared to methods relying on iterative closest point or photometric and depth error minimization. This BA based approach allows the system to achieve superior pose estimation and map quality, as it refines the 3D structure by minimizing reprojection errors across all frames, rather than just aligning depth maps or image intensities. Additionally, in stereo SLAM, ORB-SLAM2 distinguishes itself by employing both close and far stereo points along with monocular observations. This multi-modal input allows it to outperform state-of-the-art direct stereo SLAM methods, particularly in challenging scenarios where stereo vision alone might struggle.

Mur-Artal and Tardós (2017) also introduces a lightweight localization mode that disables mapping while still enabling the reuse of pre-built maps. This feature is especially useful in applications requiring real time localization without the computational overhead of continuous mapping. It ensures that the system can efficiently localize in known environments, making it suitable for long-term deployment in areas where maps have already been established.

However, the system is not without its challenges. As mentioned by Mur-Artal and Tardós (2017) ORB feature extractor, a cornerstone of ORB-SLAM2's architecture, struggles in low-light conditions or environments populated with unique or repetitive textures. These conditions

can lead to inadequate feature detection and matching, potentially degrading SLAM performance. Despite these limitations, ORB-SLAM2's architecture remains largely consistent with ORB-SLAM, with the primary differences lying in how the input processing module handles different sensor modalities. This architectural consistency ensures that the system retains the robustness and efficiency of its predecessor while expanding its applicability to a wider range of camera types.

2.2.2.4 ORB-SLAM3

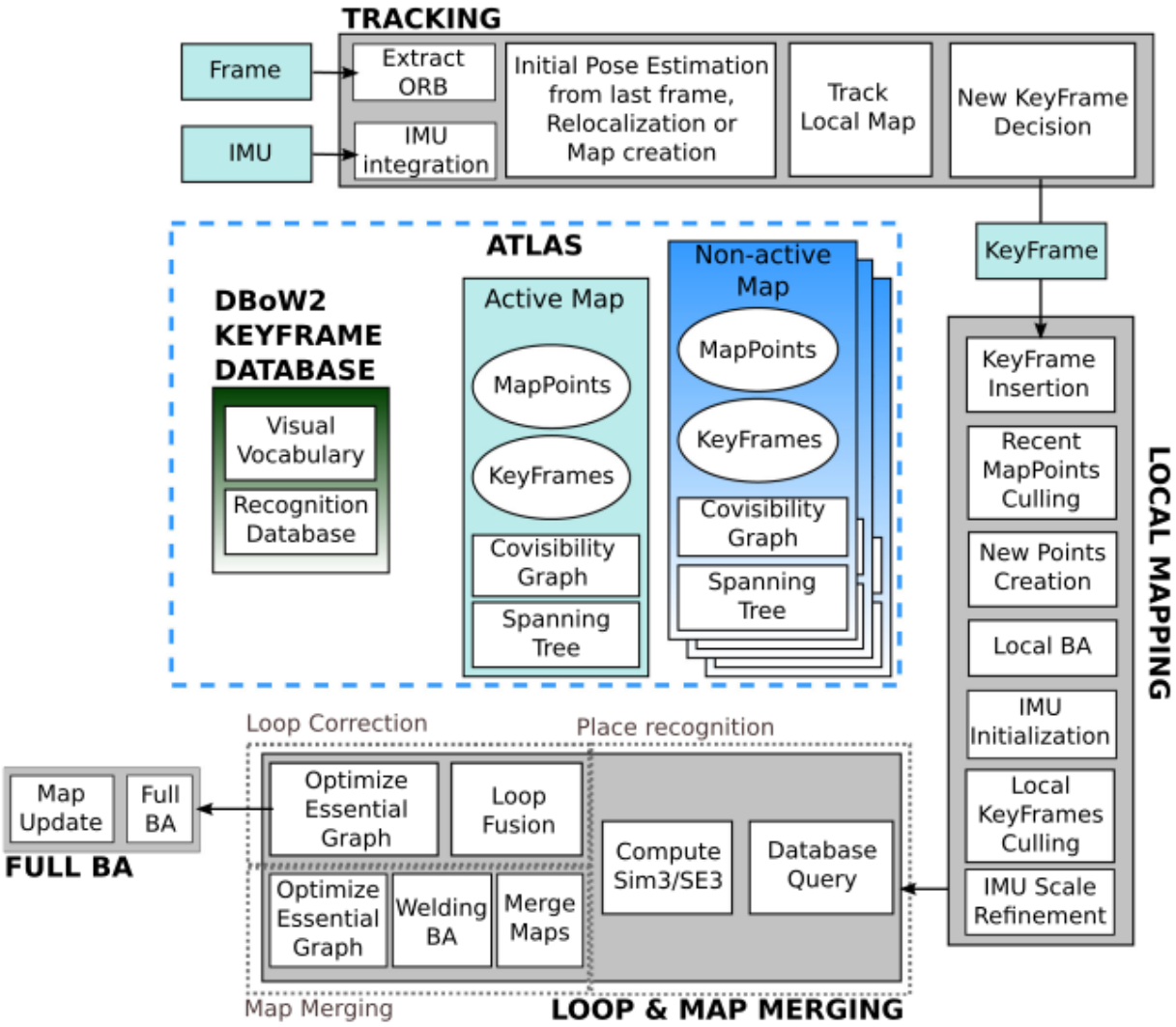

Figure 2.6: System Architecture of ORB-SLAM3 (Campos et al., 2021)

ORB-SLAM3 (Campos et al., 2021) represents a significant evolution in SLAM technology, introducing a robust monocular and stereo visual–inertial SLAM system that fully integrates MAP estimation, even during the inertial measurement unit (IMU) initialization phase. This system builds on an initialization method previously presented in earlier research, integrating it seamlessly with ORB-SLAM's visual–inertial framework. The extension to stereo-inertial

SLAM and its thorough evaluation on public datasets demonstrate the system's robustness and accuracy. Notably, ORB-SLAM3's monocular and stereo visual–inertial systems outperform existing visual–inertial approaches, maintaining high accuracy even in sequences devoid of loop closures. This robustness underscores the system's capability to operate reliably in diverse environments.

Architecturally, ORB-SLAM3 is very similar to its predecessor, ORB-SLAM2, as illustrated in Figure 2.6 The primary differences lie in the integration of IMU data into the Tracking thread, which enhances pose estimation, especially in challenging scenarios. Additionally, the Loop Closing and Map Merging thread has been extended to include a map merging module, which allows for the seamless integration of maps built at different times or in different sessions. These enhancements make ORB-SLAM3 more versatile and robust while retaining the efficiency and reliability of the previous system.

One of the significant enhancements in ORB-SLAM3 is its improved recall in place recognition. Traditional visual SLAM and visual odometry systems (Mur-Artal, Montiel, and Tardos, 2015; Qin, Li, and Shen, 2018), which often utilize the DBoW2 (Gálvez-López and Tardós, 2012) bag of words library, rely on temporal consistency—requiring three consecutive keyframes to match the same area before performing geometric consistency checks. While this approach boosts precision, it often slows down loop closure and the reuse of previously mapped areas. ORB-SLAM3 (Campos et al., 2021) introduces a novel place recognition algorithm that first checks candidate keyframes for geometric consistency and then evaluates local consistency with three covisible keyframes, which are typically already present in the map. This strategy enhances recall, densifies data association, and improves overall map accuracy, though it comes at a slightly increased computational cost.

Additionally, ORB-SLAM3 introduces the ORB-SLAM Atlas (Elvira, Tardós, and Montiel, 2019), the first complete multimap SLAM system capable of handling visual and visual–inertial systems in both monocular and stereo configurations. The Atlas can represent a set of disconnected maps and apply all mapping operations—place recognition, camera relocalization, loop closure, and seamless map merging—across them. This functionality enables the system to automatically use and combine maps built at different times, facilitating incremental multisession SLAM. The system builds on a preliminary version of ORB-SLAM Atlas for visual sensors, incorporating the new place recognition system and the visual–inertial multimap system, which have been rigorously evaluated on public datasets.

Another innovative feature of ORB-SLAM3 is its abstract camera representation, which decouples the SLAM code from the specific camera model used. This abstraction allows for the addition of new camera models by simply providing their projection, unprojection, and Jacobian functions. Implementations for pin-hole (Tsai, 1987) and fisheye (Kannala and Brandt, 2006) models are provided, enhancing the system's flexibility and adaptability to different camera setups.

However, like its predecessors, ORB-SLAM3 retains the underlying system architecture from ORB-SLAM2 (Mur-Artal and Tardós, 2017), particularly in monocular mode, where the ORB feature extractor is a critical component. This feature extraction process continues to struggle in low-light environments and with unique or repetitive objects. While ORB-SLAM3 mitigates these challenges in some cases through the use of IMUs, there are scenarios—such as environments with significant electromagnetic interference or mechanical vibrations—where monocular SLAM is the only viable option. In these situations, the system's reliance on pure vision-based approaches may limit performance, particularly in challenging lighting or dynamic conditions.

According to Joshi et al. (2019), ORB-SLAM3 stands as the current state of the art open source visual SLAM system, outperforming all other SLAM systems in terms of absolute trajectory error in both loop closure and non-loop closure scenarios. This advancement underscores its leading position in the field, solidifying ORB-SLAM3 as a benchmark for accuracy and robustness in visual SLAM technology.

2.2.2.5 Efforts To Improve ORB-SLAM3

Recent efforts to enhance ORB-SLAM3 have focused on incorporating deep learning-based feature extractors, which offer increased robustness in scenarios with varying light intensities and limited unique objects. Notable examples include DX-SLAM (Li et al., 2020), which replaces the traditional ORB feature extractor with HF-Net (Liu and Aitken, 2023), and SP-VSLAM (Yin et al., 2023), which utilizes the SuperPoint (DeTone, Malisiewicz, and Rabinovich, 2018) feature extractor. These systems leverage advanced deep learning techniques to improve the detection of keypoints under challenging conditions. The results from these approaches demonstrate significant improvements in absolute trajectory error (ATE) compared to ORB-SLAM3 (Campos et al., 2021), highlighting the potential of deep learning-based methods to further advance the accuracy and robustness of SLAM systems.

2.3 Deeper Look At ORB Feature Extractor

Having traced the evolution of visual SLAM systems, it is crucial to delve deeper into one of the most widely used feature extractors. This section will explore the technical aspects of ORB, providing a foundation for understanding its strengths and limitations.

ORB (Ethan, 2011) is a feature extraction method widely used in computer vision, particularly for real-time applications like object recognition and image stitching. ORB was introduced as a more efficient alternative to SIFT (Lindeberg, 2012) and SURF (Bay, Tuytelaars, and Van Gool, 2006), both of which, while powerful, are computationally demanding. The development of ORB focused on maintaining high performance in feature detection and matching, while significantly reducing computational complexity, making it well-suited for devices with limited processing power, such as mobile phones and embedded systems.

2.3.1 Understanding ORB

The ORB algorithm builds on two primary components: the FAST corner detector (Viswanathan, 2009) and the BRIEF descriptor (Calonder et al., 2010) . FAST (Viswanathan, 2009) is known for its speed in detecting keypoints but lacks robustness to rotations. ORB addresses this limitation by incorporating a method to compute the orientation of each keypoint, thus making the detection process invariant to in-plane rotations. The BRIEF (Calonder et al., 2010) descriptor, employed by ORB to describe the detected keypoints, is a binary descriptor that offers faster computation than gradient-based descriptors used in algorithms like SIFT. However, since BRIEF is inherently sensitive to rotations, ORB modifies it to account for the orientation of the keypoints, thereby enhancing its robustness.

A significant innovation in ORB (Ethan, 2011) is its approach to achieving rotational invariance. The algorithm calculates the orientation of each keypoint by analyzing the intensity centroid of the image patch around the keypoint. This orientation is then used to align the

BRIEF (Calonder et al., 2010) descriptor with the keypoint's orientation, ensuring that the feature description remains consistent even if the image undergoes rotation. This makes ORB particularly effective in scenarios where images might be captured from different angles or perspectives.

In addition to rotational invariance, ORB improves the discriminative power of the BRIEF descriptor by employing a learning-based method to select the most uncorrelated binary tests. This process involves choosing a subset of binary tests that maximize variance and minimize correlation, thereby producing a descriptor that is both distinctive and efficient. As a result, ORB not only offers faster processing times compared to SIFT and SURF but also delivers high-quality feature matching, making it a versatile tool in computer vision tasks.

2.3.2 Keypoint Detection and Descriptor Generation

2.3.2.1 Keypoint Detection

The steps for keypoint detection in ORB are the following:

1. **Initial Detection with FAST**: ORB first uses the FAST (Viswanathan, 2009) algorithm to detect a large number of keypoints. FAST is a corner detection method that identifies keypoints by checking if a set of contiguous pixels around a candidate pixel are either brighter or darker than a certain threshold.

2. **Top N Selection**: From the potentially large set of detected keypoints, ORB ranks these keypoints based on their Harris (Derpanis, 2004) corner response. It then selects the top-N keypoints, where N is the maximum number specified by the user. The default N value in the openCV (Bradski and Kaehler, 2000) is 500.

3. **Distribution Control**: ORB also incorporates a mechanism to ensure that the selected keypoints are evenly distributed across the image, rather than clustering in specific regions. This is done by dividing the image into a grid and selecting keypoints from each cell of the grid.

2.3.2.2 Descriptor Generation

The Descriptor for each keypoint is generated by ORB with the following steps:

1. **Orientation Assignment**: ORB calculates the orientation of each keypoint. This is done by computing the intensity centroid of the patch around the keypoint. The centroid is determined by calculating the moments of the patch, and the orientation is the angle of the vector pointing from the keypoint to the centroid. This orientation is used to rotate the patch so that the BRIEF (Calonder et al., 2010) descriptor becomes rotation-invariant.

2. **Pairwise Intensity Comparisons**: For each keypoint, ORB samples pairs of points within a small patch (e.g., 31x31 pixels) centered around the keypoint. The BRIEF (Calonder et al., 2010) descriptor is constructed by comparing the intensities of these pairs of points. If the intensity of the first point is greater than the second, a bit is set to 1; otherwise, it is set to 0. This process generates a binary string (vector) of 256 bits (or

32 bytes), which serves as the descriptor for that keypoint. The pair selection in ORB is optimized to minimize correlation and maximize variance, making the descriptor more distinctive.

2.3.3 Limitations of ORB and the Need for Advanced Feature Extractors

While ORB has been a highly effective and computationally efficient feature extractor, particularly in real-time applications, it does have certain limitations that have become more apparent as SLAM systems have evolved. As discussed earlier, ORB struggles in environments with varying light conditions and in scenarios where unique features are sparse. These challenges are particularly problematic in monocular SLAM systems, such as those implemented in ORB-SLAM2 (Mur-Artal and Tardós, 2017) and ORB-SLAM3 (Campos et al., 2021), where the quality of feature extraction directly impacts the system's ability to accurately map and localize.

In low-light environments, the ORB feature extractor's reliance on intensity-based keypoint detection and binary descriptors can lead to poor performance, as the algorithm may fail to detect sufficient or reliable keypoints. Additionally, the binary nature of the BRIEF descriptor (Calonder et al., 2010), while computationally efficient, is inherently less robust compared to descriptors generated by more recent, deep learning based methods. These issues have motivated researchers to explore alternative feature extraction techniques that can offer greater robustness and accuracy, particularly in challenging conditions.

While ORB has proven effective in various applications, its limitations in challenging environments necessitate the exploration of more advanced feature extractors. The next section introduces ALIKED, a deep learning-based method designed to address these shortcomings.

2.4 Exploring ALIKED: A Deep Learning-Based Feature Extractor

Building on the limitations identified in traditional feature extractors like ORB, particularly in challenging environments, researchers (DeTone, Malisiewicz, and Rabinovich, 2018; Liu and Aitken, 2023; Zhao et al., 2022) have increasingly turned to deep learning based methods to enhance robustness and performance. ALIKED (Zhao et al., 2023) is one such feature extractor that leverages deep learning to overcome the challenges faced by classical methods.

Recent studies comparing deep learning-based feature extractors, such as SuperPoint (DeTone, Malisiewicz, and Rabinovich, 2018) and LF-Net (Ono et al., 2018), with classical extractors like ORB (Ethan, 2011), have demonstrated significant advantages. Joshi et al. (2019) states that SuperPoint has been shown to run faster than ORB while extracting a larger number of consistent features across corresponding frames. This consistency is critical in applications such as SLAM, where accurate and reliable keypoint detection directly influences the system's ability to map and localize effectively.

2.4.1 Understanding ALIKED

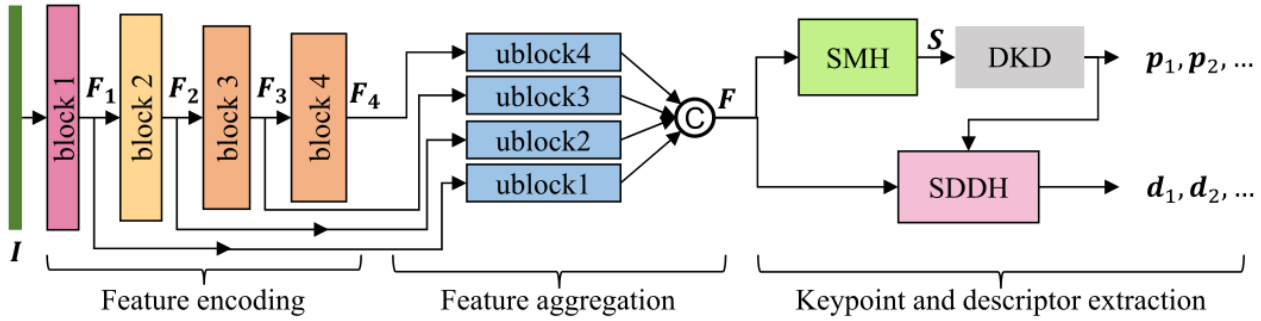

Figure 2.7: Model Architecture of ALIKED (Zhao et al., 2023)

ALIKED (Zhao et al., 2023) is a novel approach designed for efficient and accurate keypoint detection and descriptor extraction in computer vision tasks. This method addresses the challenges of high computational costs and the need for robust performance in real-world scenarios.

The Figure 2.7 shows the architecture of ALIKED is composed of several integral components that contribute to its effectiveness. The core of the architecture includes four Feature Encoding Blocks, which are responsible for capturing and encoding the visual features from the input image. These blocks extract hierarchical features that are crucial for identifying keypoints with high precision. Following this, the extracted features are processed through four Feature Aggregation Blocks. These blocks further refine and aggregate the encoded features, enhancing their representational power and ensuring that the keypoints are robust against various transformations.

The output of the Feature Aggregation Blocks is then concatenated to form the final image feature representation. This comprehensive representation encapsulates the essential details needed for accurate keypoint detection and descriptor extraction.

Key components that play a critical role in the ALIKED architecture include the Score Map Head (SMH), which predicts the likelihood of keypoint presence at each location in the image, and the Differentiable Keypoint Detection (DKD) mechanism, which allows for precise and efficient keypoint localization. Additionally, the Sparse Deformable Descriptor Head (SHHD) is employed to generate descriptors that are both sparse and deformable, enabling the network to handle variations in viewpoint and scale more effectively.

2.4.2 Keypoint Detection and Descriptor Generation

From Figure 2.7 we can see that ALIKED uses Score Map Head (SMH) and Differentiable Keypoint Detection (DKD) (Zhao et al., 2022) for detecting its keypoints. The descriptors are generated by using the Sparse Deformable Descriptor Head (SHHD) (Zhao et al., 2023) based on the detected keypoints. The following sections explain this process in more detail.

2.4.2.1 Keypoint Detection

ALIKED (Zhao et al., 2023) uses SMH which generates a score map, which assigns a likelihood to each pixel representing its potential as a keypoint. This is performed by passing the filtered image through multiple convolution layers.

The score map generated by SMH is process futher by the DKD block using the following steps:

1. **Non Maximum Suppression (NMS)**: NMS is used to filter out non maximal responses, ensuring that only the strongest keypoints are retained. This step helps in identifying distinct keypoints by suppressing weaker, neighboring pixels in the score map that could otherwise be incorrectly detected as keypoints.
2. **Thresholding**: This step discards keypoints with low confidence scores from the score map. This ensures that only keypoints with a high probability are considered for further processing.
3. **Sub Pixel Refinement**: To enhance the accuracy of the keypoints, the DKD block performs sub-pixel refinement using a softargmax operation. This operation refines the keypoint positions within a local patch around each detected keypoint, allowing for precise localization at a sub-pixel level.

At the end of processing of DKD the keypoints detected by it is generated which is then used by SHHD for generating the descriptors.

2.4.2.2 Descriptor Generation

The ALIKED network uses SHHD (Zhao et al., 2023) which uses deformable convolutions to adjust the sampling locations around keypoints dynamically. This allows descriptors to be robust against complex geometric transformations. By adapting to the local geometric structure of keypoints, ALIKED produces descriptors that are more reliable under varying conditions. The descriptors generated by ALIKED are highly discriminative, allowing for accurate matching between images even in challenging conditions, such as changes in viewpoint, illumination, and scale.

2.5 Gap in Literature

While existing research has explored comparisons between ORB and other deep learning-based feature extractors, a direct comparison with ALIKED remains absent in the literature. This study aims to bridge this gap by comprehensively evaluating the performance of ORB and ALIKED feature extractors across various metrics, including extraction speed, keypoint quantity, performance under low-light conditions, and feature repeatability. By addressing these, this study will provide valuable insights into the relative strengths and weaknesses of ORB and ALIKED, informing future research and applications in the field of computer vision and robotics.

3 Methodology

3.1 Dataset

3.1.1 Dataset Selection

The first phase of the project focused on acquiring datasets to evaluate the performance and resilience of different feature extractors. To ensure high data quality and adherence to best practices, priority was given to established sources. The EuRoC Micro Aerial Vehicle (MAV) Dataset, as described by Burri et al. (2016), was chosen for this purpose due to its extensive use in benchmarking for various SLAM systems, including ORB-SLAM3 (Campos et al., 2021), DX-SLAM (Li et al., 2020), and SP-VSLAM (Yin et al., 2023).

The EuRoC dataset was specifically selected for several reasons:

- **Realism and Relevance** The EuRoC dataset captures real world scenarios with diverse environments, including indoor and outdoor settings. These environments present challenges commonly encountered in practical SLAM applications, such as varying lighting conditions, textures, and motion dynamics.
- **Standardization and Comparison:** The EuRoC dataset is widely used within the SLAM research community. This allows for direct comparison of the project's SLAM system performance with existing algorithms and techniques evaluated on the same benchmark.
- **Variety and Difficulty:** EuRoC offer a diverse collection of scenarios with varying difficulty levels. This richness facilitates a comprehensive evaluation of the proposed SLAM system's robustness against a spectrum of environmental challenges.

By utilizing the EuRoC dataset, the project can leverage established benchmarks and ensure the evaluation reflects real world scenarios relevant to practical SLAM applications.

3.1.2 Data Acquisition

The EuRoC (Burri et al., 2016) dataset, provided by Burri et al. (2016), is hosted on the website of the Autonomous Systems Lab at ETH Zurich. For this study, the zip file version of the dataset was selected over the ROS bag format due to its greater ease of use outside the ROS environment, which aligns with the requirements of this research. The dataset was organized into folders with abbreviated names to streamline data management. Specifically, the "Machine Hall" dataset was abbreviated to "MH," while "Vicon Room 1" and "Vicon Room 2" were shortened to "V1" and "V2," respectively. The specific datasets chosen from these groups were further denoted by unique identifiers; for example, the first dataset from "Vicon Room 1" was labeled as "V101."

3.2 ORB

3.2.1 Setting up ORB

The ORB feature extractor, a widely used algorithm in computer vision, was implemented using the OpenCV (Bradski and Kaehler, 2000) library. The OpenCV implementation produces two primary outputs: an array of keypoints, represented by N tuples containing x and y coordinates, and an array of descriptors, each consisting of 32-dimensional integer vectors. The total number of keypoints, N, is dependent on the scene's complexity and the parameters specified for the ORB detector. To avoid clustering of keypoints in specific regions of the image and ensure a more evenly distributed set, OpenCV imposes a maximum limit of 500 keypoints. This constraint helps prevent the generation of excessively random keypoints and promotes a more balanced distribution across the image.

3.2.2 Running ORB

An ORB object is instantiated using the "cv2.ORB_create()" function in OpenCV. Subsequently, the "detectAndCompute" method is invoked on this object, passing the target image as a parameter. This function efficiently performs both keypoint detection and descriptor generation, returning two arrays of size N: one containing N keypoints and the other containing N corresponding descriptors.

3.3 ALIKED

In the following sections, we will compare different versions of ALIKED, including the PyTorch, ONNX, and TensorRT implementations. The comparison will focus on two key aspects: the time taken to extract features and the reproducibility of these features when applied to the same image across different versions. By evaluating the performance and consistency of each version, we aim to provide insights into their suitability for various practical applications, highlighting the strengths and potential trade-offs of each implementation.

3.3.1 Implementation and Setup of the ALIKED Model

This section outlines the setup of the ALIKED model (Zhao et al., 2023) across various frameworks, including PyTorch, ONNX, and TensorRT.

3.3.1.1 Repository Cloning and Dependency Installation

The ALIKED (Zhao et al., 2023) model is obtained by cloning the official GitHub repository. After cloning, dependencies are installed using the requirements.txt file, ensuring the necessary tools are correctly configured for model operation.

3.3.1.2 PyTorch Model Preparation

Custom Operations Compilation

To run the ALIKED model in PyTorch, it is necessary to compile custom operations, specifically the "get patches" operation developed by Zhao et al. (2023). This step is crucial for the proper functioning of the model and must be completed before any further implementation.

3.3.1.3 ONNX Model Conversion

Conversion Challenges

Converting the ALIKED model from PyTorch to the ONNX format presents challenges due to the use of the deform_conv2d operation, which is not supported by PyTorch's ONNX exporter, and the custom operations defined by Zhao et al. (2023). These issues require specific modifications to enable a successful conversion.

Modifications for Compatibility

Jurić (2024) addressed these challenges by integrating the "get patches" function directly into the ALIKED model and replacing the deform_conv2d operation with an alternative provided by Murase (2021). These adjustments allowed for the successful generation of the ONNX model while retaining the original functionality.

3.3.1.4 TensorRT Model Adaptation

Original TensorRT Implementation

Jurić (2024) developed a TensorRT implementation of the ALIKED model, originally compatible with TensorRT version 8.6.1. This implementation facilitates deployment in real-time applications by leveraging TensorRT's optimized performance.

Updating for Compatibility

The TensorRT implementation was updated to support the latest TensorRT release and CUDA 12.2. These updates involved modifications to the TensorRT engine creation process, ensuring compatibility with modern hardware and software environments for improved performance.

3.3.2 Running ALIKED

3.3.2.1 Execution in PyTorch

To execute the ALIKED model using PyTorch, instantiate the model by invoking the ALIKED class from the aliked.py file located in the nets folder. The key parameters are:

1. model_name: Specifies the pretrained model to load (”aliked-t16”, ”aliked-n16”, ”aliked-n16rot”, ”aliked-n32”).
2. device: Selects the computational device (”cpu” or ”cuda”), with ”cuda” being used for GPU acceleration.
3. top_k: Determines the number of keypoints to detect.
4. scores_th: Filters keypoints by a minimum score.
5. n_limit: its the maximum number of detectable keypoints.
6. load_pretrained: A boolean indicating whether to use a pretrained model.

For the pourpose of this study the ”aliked-n32” model of ALIKED is used because because it offers the best balance between computational efficiency and matching performance, especially in handling complex geometric transformations.According to Zhao et al. (2023), it excels in accuracy for image matching and 3D reconstruction tasks compared to other ALIKED variants, making it ideal for applications requiring high precision. All other parameter are left as default.

Inference is performed by passing an image to the model’s run method, which returns keypoints, descriptors, and scores. The code for running the PyTorch version is detailed in Listing 6.2.

3.3.2.2 Running the Model with ONNX

For the ONNX version, initialize the ONNX runtime (Bai, Lu, and Zhang, 2019) with the converted ONNX model file. Run the model using the run function, which provides outputs of keypoints, descriptors, and scores similar to the PyTorch implementation. Refer to Listing 6.3 for the ONNX comparison code.

3.3.2.3 TensorRT Model Execution

In the TensorRT version, initialize the model with a TensorRT logger and engine file, and allocate GPU memory for processing. Perform a warm-up phase by calling the inference function multiple times to cache the engine in GPU memory, which prevents performance bottlenecks due to cold starts. The run function processes the model output, with post-processing to extract keypoints, descriptors, and scores by segmenting the GPU memory data. The relevant code is provided in Listing 6.4.

3.3.3 Problems Encountered

3.3.3.1 Issues with Custom Operations Build

The custom operations developed by Zhao et al. (2023) encountered build failures due to a CMake error related to GCC version requirements. The issue was resolved by upgrading to GCC 10, which provided the necessary features for building the custom operations

3.3.3.2 Dependency Management Challenges

TensorRT requires specific versions of CUDA and cuDNN, and OpenCV. OpenCV was not compatible with cuDNN versions 9 and above. To address this, I experimented with various versions and determined that CUDA 12.2, cuDNN 8.9.7, OpenCV 4.8.0, and TensorRT 10.3.0 were compatible. I also set up separate virtual environments to manage these dependencies effectively due to the lack of comprehensive documentation.

3.3.3.3 TensorRT Model Inference Difficulties

While Jurić (2024) provided a working TensorRT model for version 8.6.1, compatibility issues with newer versions necessitated modifications. Specifically, TensorRT 10.3.0 does not support the 'execute_async_v2' function, requiring a switch to 'execute_async_v3' for GPU memory management. This change, which was not well-documented, involved significant adjustments. The modified TensorRT model file, based on Zhao et al. (2023) work, is detailed in Listing 6.7.

With the ALIKED model set up across various frameworks, the next logical step is to establish a framework for comparing its performance against the traditional ORB feature extractor. The following section outlines the criteria and methods used for this comparative analysis.

3.4 Comparison Framework

To evaluate the performance of the ORB and ALIKED feature extractors, we conducted a comprehensive comparison based on two key metrics: inference time and the number of keypoints generated per frame. The EuRoC dataset (Burri et al., 2016) was used to assess performance under real-world conditions.

3.4.1 Inference Time Analysis

Four distinct feature extractor variants were evaluated:

1. ORB Feature Extractor
2. PyTorch ALIKED Feature Extractor
3. ONNX ALIKED Feature Extractor
4. TensorRT ALIKED Feature Extractor

Each variant was executed on the MH01, MH02, V101, and V201 datasets, with five iterations per dataset. To assess performance consistency, the mean, minimum, and maximum processing times were recorded for each run. This analysis focuses on the mean values to provide a representative assessment of inference time.

The scipts for comparison can be found in Listing 6.1, 6.2, 6.3 and 6.4.

3.4.2 Keypoint Generation Analysis

From the Figure 6.8 and 6.9 we can see the consistent keypoint generation among the ONNX and TensorRT ALIKED variants, which were both capped at 1000 keypoints due to limitations in their conversion processes, the comparison of keypoint quantity was limited to ORB and the PyTorch ALIKED Feature Extractor. The latter, without such limitations, exhibited varying keypoint generation rates across different scenarios. The same experimental setup used for inference time analysis was employed, with the primary focus on the number of keypoints extracted per frame.

By comparing these metrics across different feature extractors and datasets, we aim to provide a comprehensive understanding of their relative strengths and weaknesses.

The scipts for comparison can be found in Listing 6.5 and 6.6,

4 Result and Testing

4.1 Inference Time Analysis

In this section, we present the results of the inference time measurements for different feature extractors across various datasets. The tables below provide detailed insights into the average inference times recorded during multiple tests for each feature extractor. These results will be crucial in understanding the performance efficiency of the different methods.

Table 4.1: Average Inference Time of ORB Feature Extractor Across Datasets (in milliseconds)

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**
MH01	5.38	5.38	5.41	5.4	5.07	5.328
MH02	4.84	4.89	4.8	4.89	4.88	4.86
V101	4.0	3.85	3.85	3.1	3.86	3.732
V201	4.2	3.86	3.86	3.85	3.89	3.932

Table 4.2: Average Inference Time of PyTorch ALIKED Feature Extractor Across Datasets (in milliseconds)

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**
MH01	40.56	41.13	40	40.99	40.54	40.644
MH02	41.03	40.93	39.68	40.9	40.32	40.572
V101	40.78	40.5	39.47	40.64	40.45	40.368
V201	40.86	40.24	40.21	40.48	40.45	40.448

Table 4.3: Average Inference Time of ONNX ALIKED Feature Extractor Across Datasets (in milliseconds)

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**
MH01	691.71	638.84	640.84	640.44	640.91	650.548
MH02	655.58	639.61	641.4	639.69	640.83	643.422
V101	643.13	639.38	638.67	637.47	638.91	639.512
V201	637.89	637.95	636.68	637.85	640.57	638.188

Table 4.4: Average Inference Time of TensorRT ALIKED Feature Extractor Across Datasets (in milliseconds)

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**
MH01	17.17	17.86	17.85	17.85	17.88	17.722
MH02	17.86	17.88	17.87	17.87	17.85	17.866
V101	17.91	17.93	17.91	17.91	17.91	17.914
V201	17.94	17.94	17.94	17.44	17.94	17.84

Table 4.5 compares the mean inference times of all four feature extractors across the different datasets, allowing for a direct comparison of their performance.

Table 4.5: Comparative Inference Time of Feature Extractors Across Datasets (in milliseconds)

	MH01	**MH02**	**V101**	**V201**
ORB	5.328	4.86	3.732	3.932
pytorch ALIKED	40.644	40.572	40.368	40.448
ONNX ALIKED	650.548	643.422	639.512	638.188
TensorRT ALIKED	17.722	17.866	17.914	17.84

Beyond inference time, another critical metric in evaluating feature extractors is their ability to generate consistent and reliable keypoints. The following analysis focuses on comparing the keypoint generation capabilities of ORB and ALIKED under different conditions.

4.2 Keypoint Generation Analysis

This section outlines the results of the keypoint generation capabilities of the ORB and ALIKED feature extractors on multiple datasets. The tables provide a summary of the number of keypoints generated during multiple tests, offering insights into the effectiveness of each method.

Table 4.6: Keypoints Generated by ORB Feature Extractor on multiple datasets

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Mean**
MH01	499.99	499.99	499.99	499.99	499.99	499.99
MH02	499.99	499.99	499.99	499.99	499.99	499.99
V101	493.28	493.28	493.28	493.28	493.28	493.28
V201	499.47	499.47	499.47	499.47	499.47	499.47

Table 4.7: Keypoints Generated by ALIKED Feature Extractor on multiple datasets

	Test 1	Test 2	Test 3	Test 4	Test 5	Mean
MH01	1140.75	1140.75	1140.75	1140.75	1140.75	1140.75
MH02	1135.65	1135.65	1135.65	1135.65	1135.65	1135.65
V101	931.05	931.05	931.05	931.05	931.05	931.05
V201	798.15	798.15	798.15	798.15	798.15	798.15

Table 4.8 provides a comparative view of the keypoints generated by the ORB and ALIKED feature extractors, offering a clear comparison of their keypoint generation efficiency across the datasets.

Table 4.8: Keypoint Generation Comparison: ORB vs. ALIKED

	MH01	MH02	V101	V201
ORB	499.99	499.99	493.28	499.47
ALIKED	1140.75	1135.65	931.05	798.15

5 Discussion and Evaluation

5.1 ORB Feature Extractor

5.1.1 Computational Efficiency of the ORB Feature Extractor

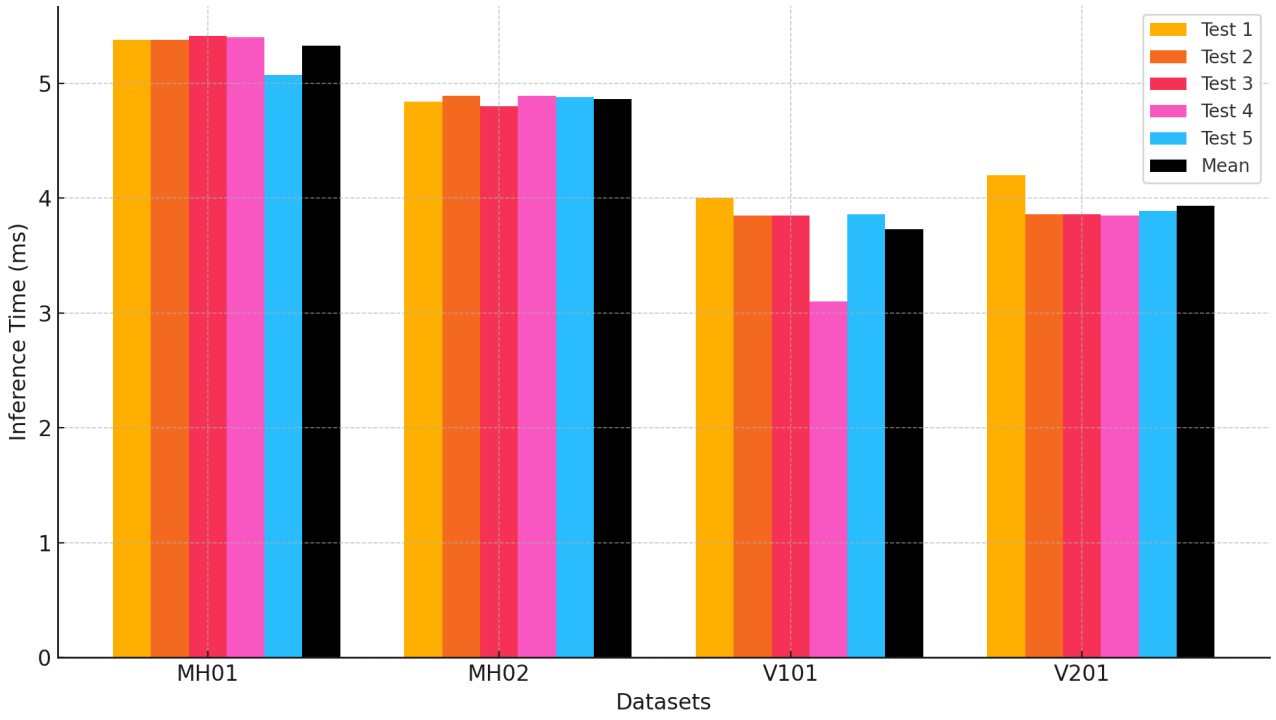

Figure 5.1: Bar plot showing Average Inference Time of ORB Feature Extractor Across Datasets

As illustrated in Table 4.1 and Figure 6.2, the ORB feature extractor demonstrated remarkable computational efficiency, with average inference times ranging from 3.732 milliseconds for dataset V101 to 5.328 milliseconds for dataset MH01. This efficiency makes ORB well-suited for real-time applications.

Surprisingly, the datasets V201 and V101, characterized by their relatively simple visual content, exhibited the fastest inference times as highlighted in Figure 5.1. Given ORB's propensity for extracting keypoints from salient image features, one might expect datasets with more complex visual content to require more computational resources or yield a reduced number of keypoints. This counterintuitive observation suggests that the ORB algorithm's performance may be influenced by factors beyond mere image complexity, warranting further investigation.

5.1.2 Keypoint Extraction Performance of ORB

Table 4.6 and Figure 6.6 reveal that the ORB feature extractor consistently generated a substantial number of keypoints across all datasets, averaging 497.54 keypoints per test. This effectiveness in identifying salient features provides a strong foundation for subsequent tasks like feature matching and object recognition. While datasets MH01, MH02, and V201 exhibited relatively consistent keypoint counts, dataset V101 demonstrated a slightly lower average of 493.28 keypoints.

5.1.3 Analyzing the Anomaly in ORB Performance

The preceding sections highlighted the ORB feature extractor's faster performance on datasets that were initially anticipated to be more challenging. Figure 5.2, a visualization of the feature extractor's output, provides valuable insights into this anomaly. The extracted features are marked as red circles on the image.

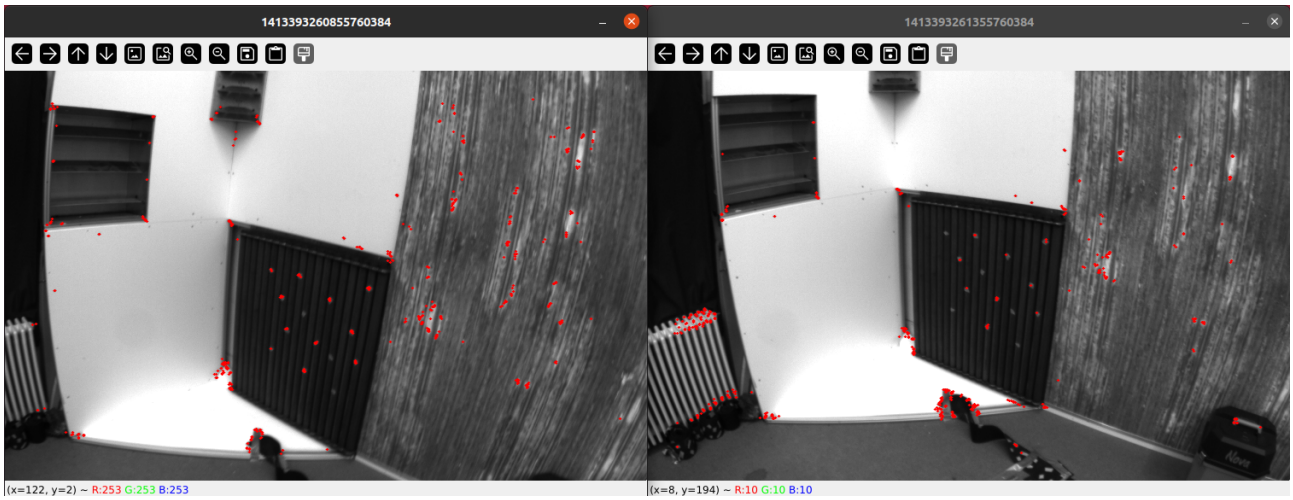

Figure 5.2: Visualisation of ORB Feature Extraction in Consecutive Frames on V201 Dataset

We observe that while the ORB feature extractor extracts a significant number of features, these features may not be consistently located on the same objects across different frames.

This inconsistency in keypoint generation, while contributing to a high overall keypoint count, can negatively impact the quality of feature matching and object recognition. This explains why datasets V101 and V201, despite their simpler visual content, exhibited a higher number of keypoints.

5.1.4 Challenges and Limitations of ORB

As discussed in the previous section, the ORB feature extractor exhibit limitations when dealing with images containing a limited number of unique objects. Additionally, literature review on the work of Mur-Artal and Tardós (2017), have identified challenges in extracting features under low-light conditions.

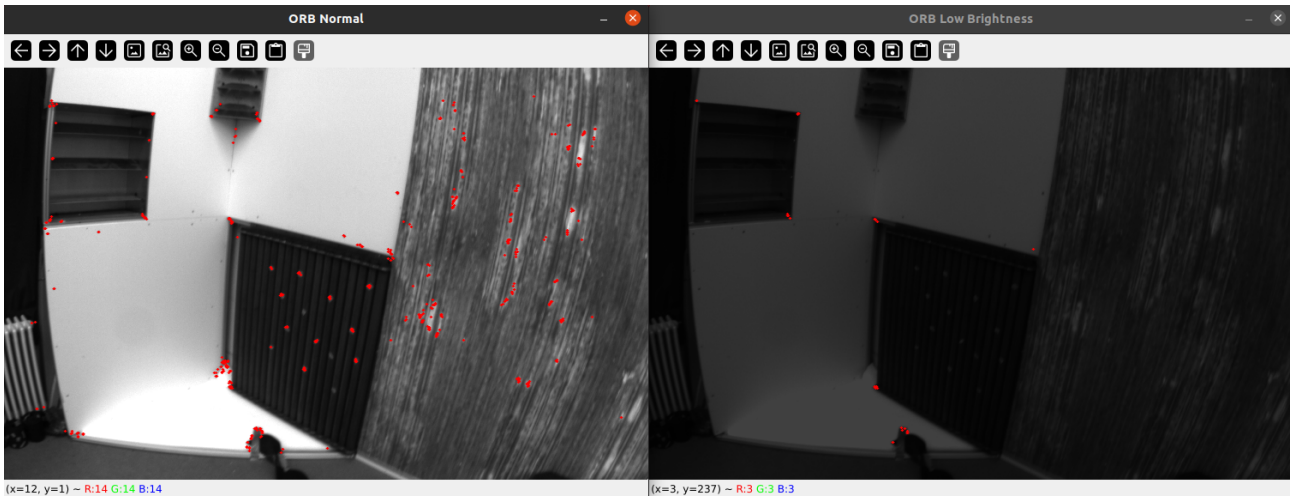

Figure 5.3: Visualisation of ORB Feature Extractor low light Scenario on V201 Dataset

Figure 5.3 provides empirical evidence supporting these observations. By reducing the brightness of the left image by 100%, we simulated a low-light condition and observed a significant reduction in the number of features extracted by the ORB feature extractor on the right image. This finding aligns with the observations of Mur-Artal and Tardós (2017), confirming the ORB algorithm's limitations in low-light environments.

5.2 ALIKED Feature Extractor

5.2.1 Addressing ORB's Shortcomings

As highlighted in Section 5.1.4, the ORB feature extractor faces challenges in extracting consistent keypoints and operating effectively in low-light conditions.

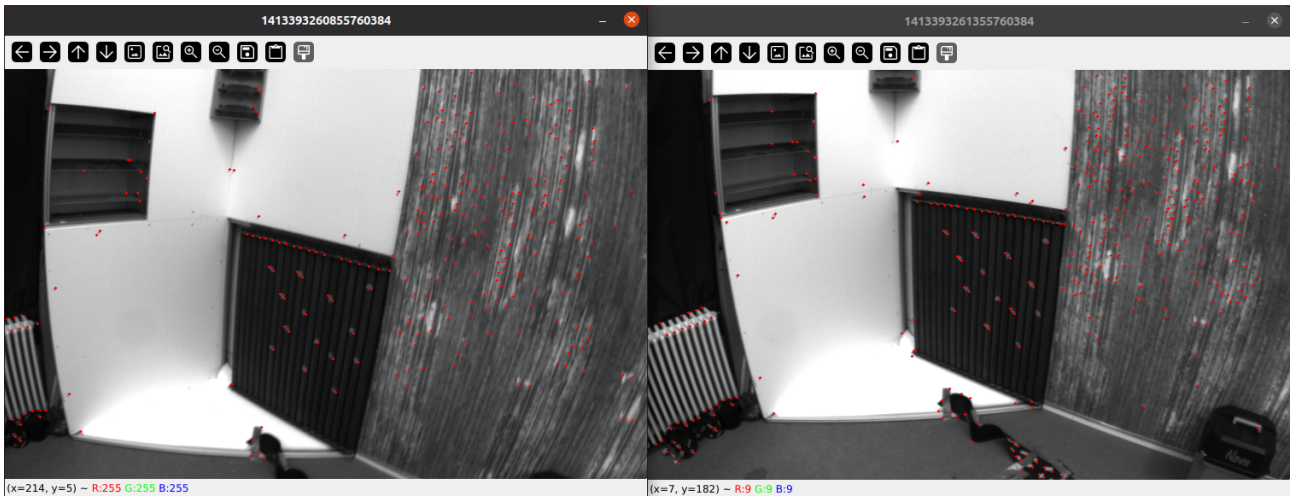

Figure 5.4: Visualisation of ALIKED Feature Extractor in Consecutive Frames on V201 Dataset

Figure 5.4 demonstrates ALIKED's ability to overcome these limitations. ALIKED consistently generates keypoints that are well-aligned with object features, making it particularly valuable for tasks such as scene recognition and feature matching in SLAM applications. Moreover, ALIKED's capacity to produce a larger quantity of keypoints, combined with their consistency, enhances its effectiveness in these domains.

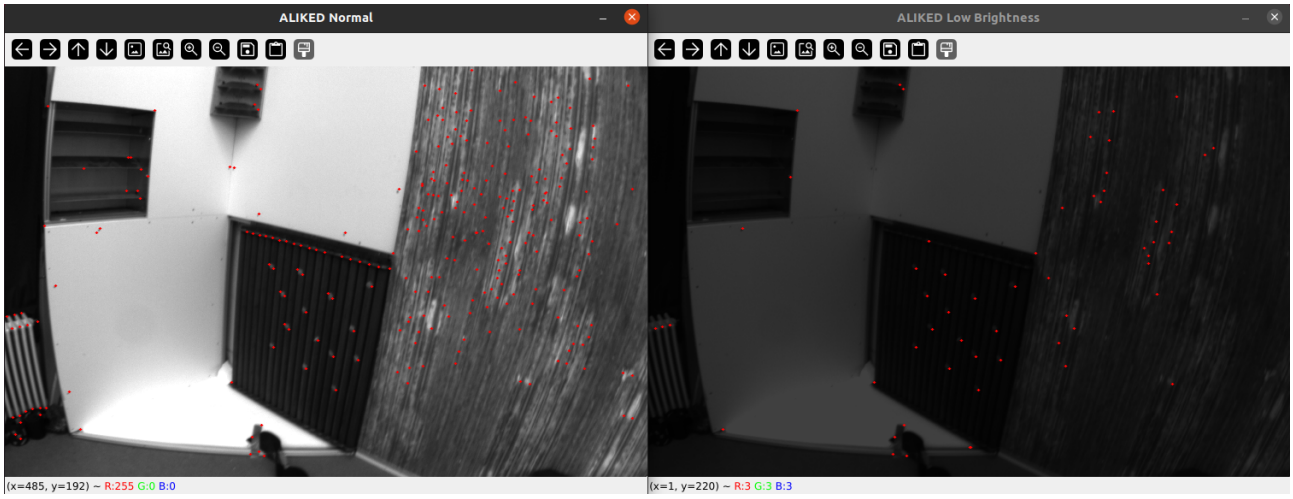

Figure 5.5: Visualisation of ALIKED Feature Extractor low light Scenario on V201 Dataset

Figure 5.5 further underscores ALIKED's advantages. Unlike ORB, ALIKED can extract consistent features even under low-light conditions. This is a critical capability in visual SLAM, as it ensures that maps generated in one lighting condition remain usable under varying lighting conditions, preventing localization issues that ORB frequently encounters, as noted by Mur-Artal and Tardós (2017).

5.2.2 Performance of ALIKED

5.2.2.1 Keypoint Extraction

As evidenced by the data presented in Table 4.2 and Figure 6.7, ALIKED consistently outperformed ORB in terms of keypoint generation. ALIKED produced significantly larger numbers of keypoints across all datasets, ranging from 798.15 for dataset V201 to 1140.75 for dataset MH01, compared to ORB's average of 497.54 keypoints. This substantial increase in keypoint quantity indicates that ALIKED is more effective at identifying salient features within images, providing a richer feature representation that is likely to benefit downstream tasks such as feature matching, object recognition, and pose estimation which is essential for SLAM.

5.2.2.2 Computational Efficiency

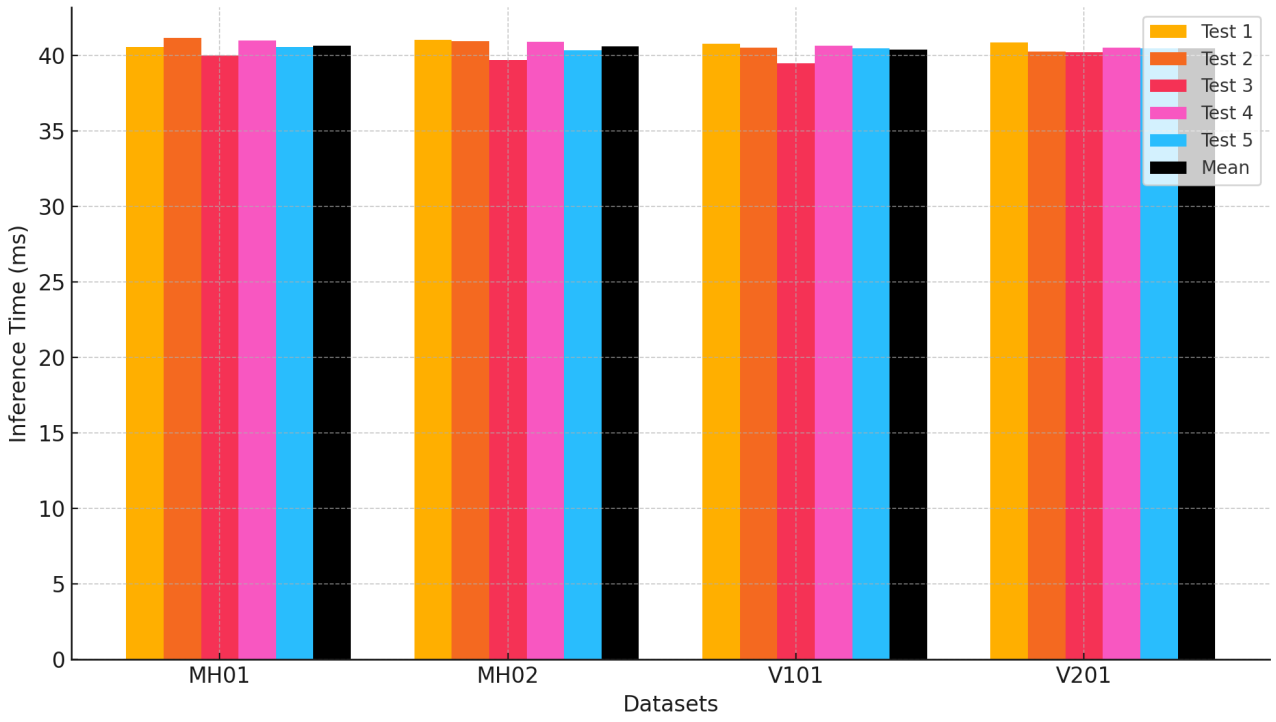

Figure 5.6: Bar Plot of Average Inference Time for PyTorch Based ALIKED Feature Extractor Across Datasets

As shown in the data presented in Table 4.2 and Figure 6.3, ALIKED's superior keypoint generation performance comes at the cost of increased inference time. The average inference time for ALIKED ranged from 40.368 milliseconds for dataset V101 to 40.644 milliseconds for dataset MH01, compared to ORB's average of 5.328 milliseconds. This suggests that ALIKED's increased computational complexity is a trade-off for the improved keypoint generation capabilities. However, the specific impact of this increased inference time on real-world applications will depend on the target frame rate and other system constraints.

5.3 Limitation of ALIKED

ALIKED, being implemented in PyTorch (Paszke et al., 2019), a Python-based deep learning framework, presents certain limitations when deployed in resource-constrained SLAM systems. Python's dynamic memory allocation can lead to performance issues on such hardware. To address this, ALIKED can be converted to the ONNX format (Bai, Lu, and Zhang, 2019), which allows for execution in statically typed languages and other deep learning frameworks.

In contrast to ALiKE, ORB, as an algorithm, can be implemented directly in any programming language. This flexibility can provide advantages in specific deployment scenarios.

While ALIKED exhibits superior feature extraction capabilities, its relatively slow inference times may make it less suitable for real-time applications. To mitigate this limitation, the TensorRT (NVIDIA, 2021) optimization tool can be employed to generate an engine file from the ONNX model. This engine file is highly optimized for the specific hardware platform, ensuring efficient resource utilization and significantly reduced inference times. However, it is important to note that the engine file is not portable, limiting its flexibility in deployment scenarios.

To compensate for this limitation, the engine file can be optimized for specific hardware configurations, potentially achieving even lower inference times. By leveraging high-performance hardware, such as GPUs or specialized AI accelerators, the inference speed can be further improved, making ALIKED a more viable option for real time applications.

5.4 Performance of ONNX model

While ONNX ALIKED offers the advantage of portability compared to the PyTorch implementation, it comes at the cost of significantly increased inference times. As shown in Table 4.3 and Figure 6.4, ONNX model exhibited the highest inference times across all feature extractors, with mean values ranging from 638.188 milliseconds (V201) to 650.548 milliseconds (MH01). This dramatic increase in processing time can be attributed to the overhead associated with model conversion and the ONNX execution environment.

ONNX is designed for cross-platform compatibility, often resulting in trade-offs concerning execution speed. Despite its slower performance, ONNX model can be advantageous when deploying in environments where PyTorch is not supported, particularly when integrating with SLAM systems.

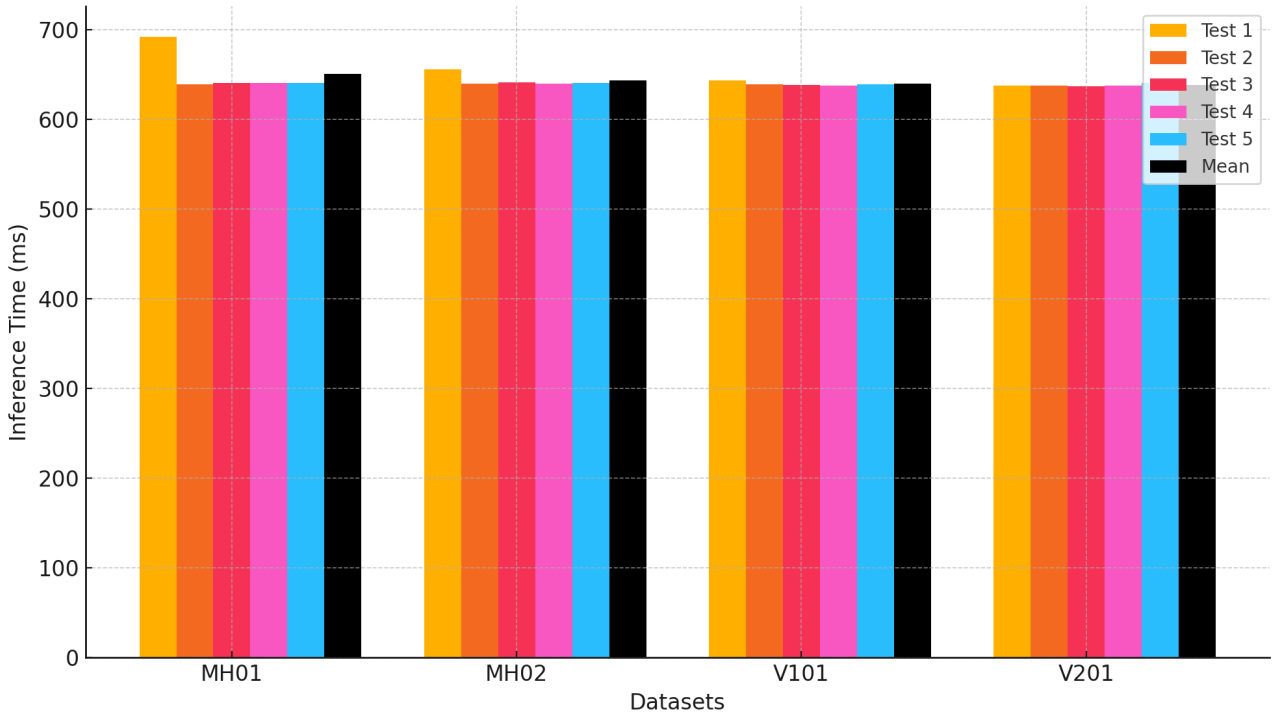

Figure 5.7: Bar Plot of Average Inference Time for ONNX Based ALIKED Feature Extractor Across Datasets

5.5 Performance of the TensorRT model

As evidenced by the data presented in Table 4.4 and Figure 6.5, the TensorRT-based ALIKED feature extractor demonstrates a significant reduction in inference time compared to its PyTorch counterpart. The mean inference times for TensorRT ALiKE range from 17.722 milliseconds (MH01) to 17.914 milliseconds (V101), representing a substantial improvement over the PyTorch implementation.

To quantify the performance boost, we can calculate the percentage reduction in inference time. For example, on the MH01 dataset, the inference time decreased from 40.644 milliseconds (PyTorch) to 17.722 milliseconds (TensorRT), representing a reduction of approximately 56.5%. Similar improvements are observed across the other datasets, highlighting the effectiveness of TensorRT optimization.

While TensorRT model's performance gains are tied to specific hardware configurations, the engine file's efficiency can be further improved by leveraging more powerful hardware. By deploying TensorRT model on systems with advanced GPUs or specialized AI accelerators, even greater performance gains can be achieved. This makes TensorRT model a promising option for resource constrained applications that require both high-speed processing and the flexibility to adapt to different hardware environments.

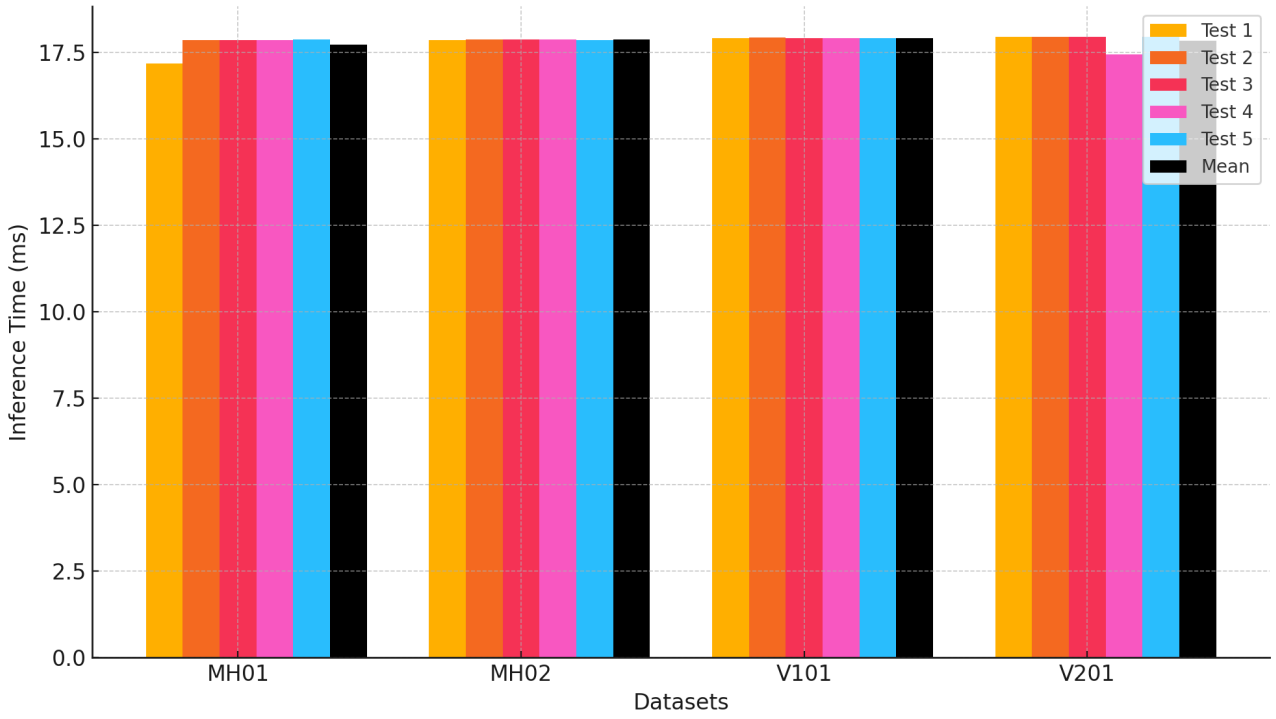

Figure 5.8: Bar Plot of Average Inference Time for TensorRT Based ALIKED Feature Extractor Across Datasets

5.6 Comparative Analysis

From Table 4.5 and Figure 5.9, which compares the inference times of the different feature extractors across datasets, highlights the stark contrast between traditional and deep learning-based methods. ORB, with its rapid inference times, is clearly the most efficient in terms of speed. However, this efficiency comes at the cost of accuracy and robustness, which are areas where ALIKED excels.

Among the ALIKED variants, the TensorRT implementation is markedly faster than both the PyTorch and ONNX versions, underscoring the benefits of using a platform-specific optimization tool like TensorRT for deployment in environments where real-time processing is essential. On the other hand, the ONNX implementation, despite its slower performance, offers the advantage of cross-platform compatibility, making it a versatile option for various deployment scenarios.

Overall, the choice of feature extractor should be guided by the specific requirements of the application. If speed is the primary concern, ORB would be the preferred options. However if the application demands accuracy and robustness, with a slight reduction in speed the TensorRT implementation of ALIKED will be most appropriate.

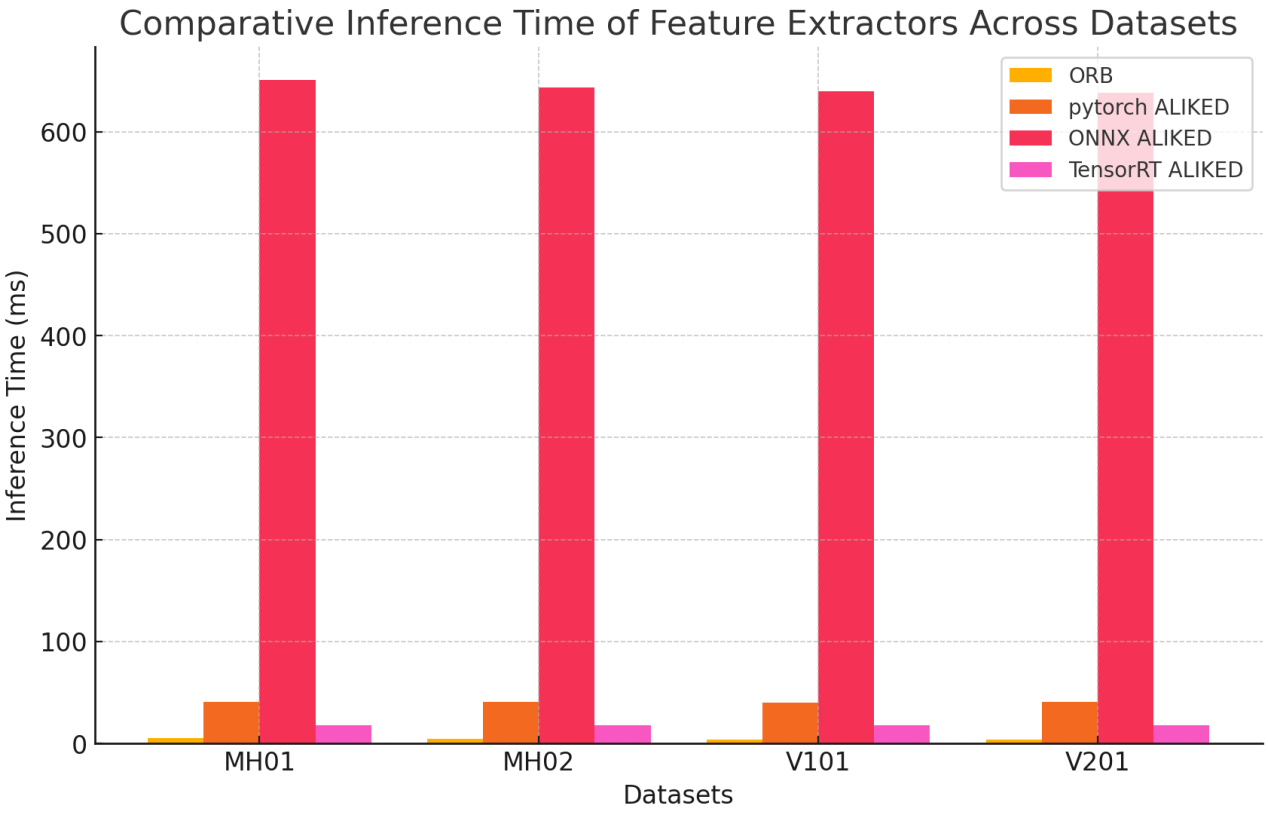

Figure 5.9: Comparative Inference Time of Feature Extractors Across Datasets

The comparative evaluation highlights the strengths and areas for improvement in both ORB and ALIKED. In light of these findings, the conclusion will summarize the key takeaways and propose directions for future research and enhancement.

6 Conclusion and Future Enhancement

6.1 Conclusion

The analysis and evaluation of ORB and ALIKED feature extractors, demonstrate the trade-offs between computational efficiency, keypoint extraction performance, and robustness in diverse scenarios. ORB, a traditional method, exhibited remarkable computational efficiency, with inference times ranging between 3.732 milliseconds (for dataset V101) and 5.328 milliseconds (for dataset MH01). This efficiency underscores ORB's suitability for real-time applications where speed is a critical requirement. However, ORB's performance is not without limitations. The algorithm's efficiency does not necessarily translate to robustness or accuracy, especially in scenarios with complex visual content or challenging conditions like low light. The anomaly observed in ORB's performance, where simpler datasets yielded faster inference times, suggests that ORB's performance may be influenced by factors other than mere image complexity.

On the other hand, ALIKED, a deep learning based feature extractor, addresses several of the shortcomings associated with ORB. ALIKED consistently outperformed ORB in terms of key-

point generation, producing significantly larger quantities of keypoints across all datasets. This robustness is particularly valuable for tasks requiring high accuracy, such as feature matching, object recognition, and pose estimation in SLAM applications. However, ALIKED's superior performance comes at a cost: increased computational complexity and, consequently, longer inference times. The average inference time for ALIKED ranged from 40.368 milliseconds (for dataset V101) to 40.644 milliseconds (for dataset MH01), which is substantially higher than ORB's inference times. Despite this, ALIKED's robustness in low-light conditions, as demonstrated by its consistent feature extraction even under reduced brightness, positions it as a strong candidate for applications where accuracy and robustness are prioritized over speed.

Further optimizations of ALIKED using the TensorRT framework demonstrated significant reductions in inference time, bringing it closer to real-time applicability. The TensorRT-based implementation achieved a reduction of approximately 56.5% in inference time compared to the original PyTorch implementation, with times ranging from 17.722 milliseconds (for dataset MH01) to 17.914 milliseconds (for dataset V101). This improvement, while still not matching the speed of ORB, makes ALIKED more viable for time-sensitive applications, particularly when deployed on high performance hardware.

However, the ONNX version of ALIKED, while offering cross-platform compatibility, exhibited the highest inference times, which could be a limiting factor in real-time applications. The trade off between portability and performance underscores the importance of selecting the appropriate implementation based on the specific requirements of the application. In summary, ORB remains a strong choice for scenarios where computational efficiency is paramount, but ALIKED, particularly when optimized with TensorRT, offers significant advantages in robustness and accuracy, making it a more suitable choice for applications that can accommodate slightly longer processing times.

6.2 Future Enhancements

6.2.1 Development of a C++ API using TensorRT Engine

The first step in enhancing the ALIKED feature extractor involves the development of a C++ API that leverages the TensorRT engine generated from the ALIKED model. TensorRT optimization has already demonstrated a significant reduction in inference time, making it more suitable for real-time applications compared to its PyTorch implementation. By implementing this optimization in C++, additional performance benefits can be realized due to the language's inherent efficiency and lower-level control over system resources. This C++ API will enable seamless integration of the optimized ALIKED model into various SLAM frameworks, ensuring faster processing times and more efficient utilization of hardware resources. Given the increasing need for real-time processing in autonomous systems, the development of this API is crucial for deploying ALIKED in resource-constrained environments, where Python-based implementations might suffer from performance bottlenecks.

6.2.2 Generation of ALIKED Vocabulary for Loop Closure Detection

A key enhancement to the ALIKED feature extractor is the development of a vocabulary specifically designed for loop closure detection, similar to the ORB vocabulary utilized DBOW2

(Gálvez-López and Tardós, 2012) library used in ORB-SLAM3 (Campos et al., 2021). Loop closure detection is an essential component of SLAM systems, allowing them to recognize previously visited locations and correct accumulated drift in the estimated trajectory. By generating a vocabulary based on the ALIKED model, it is possible to leverage its superior keypoint extraction capabilities to improve the accuracy and reliability of loop closure detection. This enhancement will enable more robust place recognition, particularly in environments where ORB might struggle due to its limitations in handling low-light conditions or complex visual scenes. The ALIKED vocabulary will serve as a critical tool in enhancing the robustness of SLAM systems, making them more capable of operating in challenging real-world scenarios.

6.2.3 Integration of C++ API and ALIKED Vocabulary into ORB-SLAM

Following the development of the C++ API and the ALIKED vocabulary, the next logical step is to integrate these components into the ORB-SLAM3 framework. ORB-SLAM3 is renowned for its accuracy and robustness, making it an ideal platform for deploying advanced feature extraction techniques. By replacing the ORB feature extractor with the optimized ALIKED model, ORB-SLAM3 can benefit from improved keypoint detection, especially in challenging conditions such as low-light environments or scenes with high visual complexity. The integration will involve modifying the ORB-SLAM3 pipeline to accommodate the ALIKED feature extractor, ensuring that the system can utilize the ALIKED vocabulary effectively for loop closure detection. This enhancement is expected to significantly improve the overall performance of ORB-SLAM3, making it even more reliable for real time applications in dynamic and unpredictable environments.

6.2.4 Performance Comparison in a Real SLAM Scenario

Once the ALIKED model has been integrated into ORB-SLAM3, it is imperative to evaluate its performance in a real SLAM scenario. This involves conducting extensive tests in diverse environments with varying lighting conditions, visual complexity, and object types. Key performance metrics to assess include keypoint extraction accuracy, loop closure detection reliability, and overall system latency.

To quantify the improvements brought about by integrating ALIKED, these metrics will be compared with those of the original ORB-SLAM3. Detailed benchmark results for ORB-SLAM3, including key performance metrics, are provided in Appendix E. These real-world evaluations will offer valuable insights into the trade-offs between the enhanced robustness and accuracy of ALIKED and the slight increase in processing time. The results will also help identify any areas needing further optimization to ensure the system meets the stringent requirements of real-time SLAM applications.

6.2.5 Optimizing ALIKED for Specialized Hardware

To further enhance ALIKED's performance in real-time applications, it is essential to optimize the TensorRT model for specific hardware configurations, such as GPUs or specialized AI accelerators. By fine-tuning the TensorRT engine to match the capabilities of the target hardware,

it may be possible to achieve even lower inference times, thereby making the ALIKED feature extractor more competitive with traditional methods like ORB. Additionally, exploring other optimization techniques, such as mixed precision training and model quantization, could further reduce computational overhead without compromising the accuracy of keypoint extraction. These optimizations will be particularly important for deploying ALIKED in embedded systems or other environments where computational resources are limited, and real-time performance is critical.

6.2.6 Exploration of ALIKED in Multi-View and Multi-Resolution SLAM

An additional avenue for future research is the exploration of ALIKED's application in multi-view and multi-resolution SLAM systems. These systems combine information from multiple perspectives or resolutions to improve the accuracy and robustness of the SLAM process. Given ALIKED's strong performance in keypoint extraction, it may prove particularly beneficial in these more complex SLAM scenarios. By providing richer and more consistent keypoint data across different views or resolutions, ALIKED could enable more accurate 3D reconstructions and better handling of dynamic environments. This enhancement would not only improve the performance of existing SLAM systems but also open up new possibilities for their application in areas such as autonomous navigation, augmented reality, and robotics.

Each of these enhancements represents a significant step forward in the development and application of the ALIKED feature extractor in SLAM systems. By addressing the limitations identified in the current implementation and leveraging advanced optimization techniques, it will be possible to create a more robust, accurate, and efficient SLAM solution that is capable of operating effectively in a wide range of real-world environments.

Bibliography

Bai, Junjie, Fang Lu, and Ke Zhang (2019). *ONNX: Open Neural Network Exchange*. `https://github.com/onnx/onnx`.

Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool (2006). "Surf: Speeded up robust features". In: *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer, pp. 404–417.

Bradski, Gary and Adrian Kaehler (2000). "OpenCV". In: *Dr. Dobb's journal of software tools* 3.2.

Burri, Michael, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart (2016). "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* 35.10. Accessed: 2024-08-19, pp. 1157–1163.

Cadena, Cesar, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard (2016). "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age". In: *IEEE Transactions on robotics* 32.6, pp. 1309–1332.

Calonder, Michael, Vincent Lepetit, Christoph Strecha, and Pascal Fua (2010). "Brief: Binary robust independent elementary features". In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, pp. 778–792.

Campos, Carlos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós (2021). "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam". In: *IEEE Transactions on Robotics* 37.6, pp. 1874–1890.

Cheeseman, P, R Smith, and M Self (1987). "A stochastic map for uncertain spatial relationships". In: *4th international symposium on robotic research*. MIT Press Cambridge, pp. 467–474.

Davison, Andrew J, Ian D Reid, Nicholas D Molton, and Olivier Stasse (2007). "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6, pp. 1052–1067.

Derpanis, Konstantinos G (2004). "The harris corner detector". In: *York University* 2.1, p. 2.

DeTone, Daniel, Tomasz Malisiewicz, and Andrew Rabinovich (2018). "Superpoint: Self-supervised interest point detection and description". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 224–236.

Dusmanu, Mihai, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler (2019). "D2-net: A trainable cnn for joint detection and description of local features". In: *arXiv preprint arXiv:1905.03561*.

Elvira, Richard, Juan D Tardós, and Jose MM Montiel (2019). "ORBSLAM-Atlas: a robust and accurate multi-map system". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6253–6259.

Ethan, Rublee (2011). "ORB: An efficient alternative to SIFT or SURF". In: *ICCV, 2011*.

Gálvez-López, Dorian and J. D. Tardós (Oct. 2012). "Bags of Binary Words for Fast Place Recognition in Image Sequences". In: *IEEE Transactions on Robotics* 28.5, pp. 1188–1197. ISSN: 1552-3098. DOI: `10.1109/TRO.2012.2197158`.

Grupp, Michael (2017). *evo: Python package for the evaluation of odometry and SLAM.* `https://github.com/MichaelGrupp/evo`. Accessed: 2024-08-21.

Guennebaud, Gaël and Benoît Jacob (2010). *Eigen*. http://eigen.tuxfamily.org.

Herrera-Granda, Erick P, Juan C Torres-Cantero, and Diego H Peluffo-Ordoñez (2023). "Monocular Visual SLAM, Visual Odometry, and Structure from Motion Methods Applied to 3D Reconstruction: A Comprehensive Survey". In: *Visual Odometry, and Structure from Motion Methods Applied to 3D Reconstruction: A Comprehensive Survey*.

Joshi, Bharat, Brennan Cain, James Johnson, Michail Kalitazkis, Sharmin Rahman, Marios Xanthidis, Alan Hernandez, Nare Karaperyan, Alberto Quattrini Li, Nikolaos Vitzilaios, and Ioannis Rekleitis (2019). "Experimental Comparison of Open Source Visual-Inertial-Based State Estimation Algorithms in the Underwater Domain". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. DOI: 10.1109/IROS40897.2019.8968049.

Jurić, Antonio (2024). *ALIKED TensorRT*. https://github.com/ajuric/aliked-tensorrt. Accessed: 2024-08-21.

Kalman, Rudolph Emil (1960). *A new approach to linear filtering and prediction problems*.

Kannala, Juho and Sami S Brandt (2006). "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses". In: *IEEE transactions on pattern analysis and machine intelligence* 28.8, pp. 1335–1340.

Klein, Georg and David Murray (2007). "Parallel tracking and mapping for small AR workspaces". In: *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, pp. 225–234.

Kümmerle, Rainer, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard (2011). "G2o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949.

Li, Dongjiang, Xuesong Shi, Qiwei Long, Shenghui Liu, Wei Yang, Fangshi Wang, Qi Wei, and Fei Qiao (2020). "DXSLAM: A robust and efficient visual SLAM system with deep features". In: *2020 IEEE/RSJ International conference on intelligent robots and systems (IROS)*. IEEE, pp. 4958–4965.

Lindeberg, Tony (2012). *Scale invariant feature transform*.

Liu, Liming and Jonathan M Aitken (2023). "HFNet-SLAM: An Accurate and Real-Time Monocular SLAM System with Deep Features". In: *Sensors* 23.4, p. 2113.

Lowe, David G (2004). "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60, pp. 91–110.

Mourikis, Anastasios I and Stergios I Roumeliotis (2007). "A multi-state constraint Kalman filter for vision-aided inertial navigation". In: *Proceedings 2007 IEEE international conference on robotics and automation*. IEEE, pp. 3565–3572.

Mur-Artal, Raul, Jose Maria Martinez Montiel, and Juan D Tardos (2015). "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE transactions on robotics* 31.5, pp. 1147–1163.

Mur-Artal, Raul and Juan D Tardós (2017). "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras". In: *IEEE transactions on robotics* 33.5, pp. 1255–1262.

Murase, Masamitsu (2021). *deform_conv2d_onnx_exporter*. https://github.com/masamitsu-murase/deform_conv2d_onnx_exporter. Accessed: 2024-08-21.

NVIDIA (2021). *TensorRT*. https://developer.nvidia.com/tensorrt/. Accessed: 2024-08-19.

Ono, Yuki, Eduard Trulls, Pascal Fua, and Kwang Moo Yi (2018). "LF-Net: Learning local features from images". In: *Advances in neural information processing systems* 31.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, and Luca Antiga (2019). "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32.

Qin, Tong, Peiliang Li, and Shaojie Shen (2018). "Vins-mono: A robust and versatile monocular visual-inertial state estimator". In: *IEEE transactions on robotics* 34.4, pp. 1004–1020.

Ramezani, Milad, Debaditya Acharya, Fuqiang Gu, and Kourosh Khoshelham (2017). "Indoor positioning by visual-inertial odometry". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4, pp. 371–376.

Schneider, C. and T. GAP Team (Aug. 2022). *Sophus, Computing in nilpotent Lie algebras, Version 1.27*. `https://gap-packages.github.io/sophus/`. Refereed GAP package.

Shi, Xuesong, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, Yangquan Guo, Zhigang Wang, Yimin Zhang, Baoxing Qin, Wei Yang, Fangshi Wang, Rosa H. M. Chan, and Qi She (2020). "Are we ready for service robots? the openloris-scene datasets for lifelong slam". In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3139–3145.

stevenlovegrove (2021). *Pangolin V6*. `https://github.com/stevenlovegrove/Pangolin`. Accessed: 2024-08-20.

Tang, Jiexiong, Ludvig Ericson, John Folkesson, and Patric Jensfelt (2019). "GCNv2: Efficient correspondence prediction for real-time SLAM". In: *IEEE Robotics and Automation Letters* 4.4, pp. 3505–3512.

Thrun, Sebastian (2002). "Probabilistic robotics". In: *Communications of the ACM* 45.3, pp. 52–57.

Tsai, Roger (1987). "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses". In: *IEEE Journal on Robotics and Automation* 3.4, pp. 323–344.

Umeyama, Shinji (1991). "Least-squares estimation of transformation parameters between two point patterns". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 13.04, pp. 376–380.

Viswanathan, Deepak Geetha (2009). "Features from accelerated segment test (fast)". In: *Proceedings of the 10th workshop on image analysis for multimedia interactive services, London, UK*, pp. 6–8.

Yi, Kwang Moo, Eduard Trulls, Vincent Lepetit, and Pascal Fua (2016). "Lift: Learned invariant feature transform". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI 14*. Springer, pp. 467–483.

Yin, Zhenyu, Dan Feng, Chao Fan, Chengen Ju, and Feiqing Zhang (2023). "SP-VSLAM: Monocular Visual-SLAM Algorithm Based on SuperPoint Network". In: *2023 15th International Conference on Communication Software and Networks (ICCSN)*. IEEE, pp. 456–459.

Zhao, Xiaoming, Xingming Wu, Weihai Chen, Peter CY Chen, Qingsong Xu, and Zhengguo Li (2023). "Aliked: A lighter keypoint and descriptor extraction network via deformable transformation". In: *IEEE Transactions on Instrumentation and Measurement* 72, pp. 1–16.

Zhao, Xiaoming, Xingming Wu, Jinyu Miao, Weihai Chen, Peter C. Y. Chen, and Zhengguo Li (Mar. 2022). "ALIKE: Accurate and Lightweight Keypoint Detection and Descriptor Extraction". In: *IEEE Transactions on Multimedia*. DOI: `10.1109/TMM.2022.3155927`. URL: `http://arxiv.org/abs/2112.02906`.

Appendices

Appendix A: Project Plan

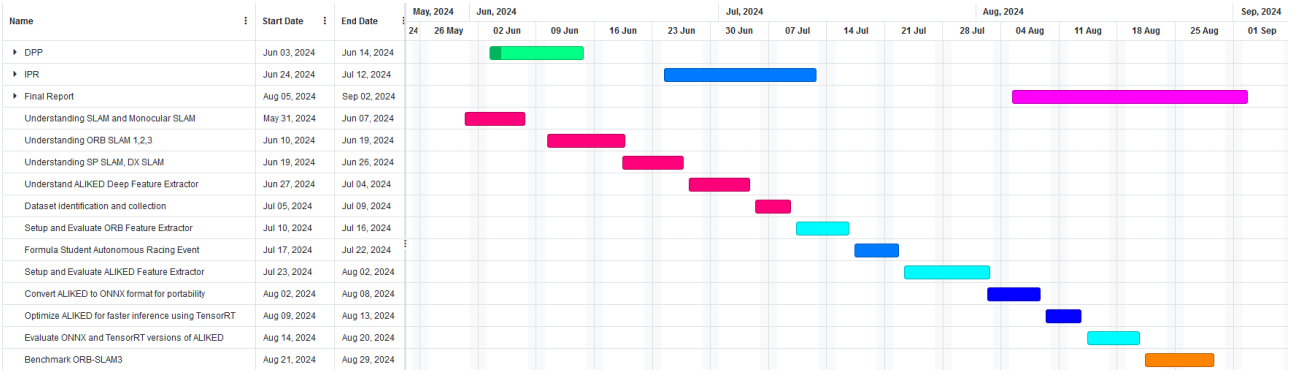

Figure 6.1: Project Plan

Appendix B : Comparison Scripts

This section provides the scripts employed for the comparative analysis of the ORB and ALIKED feature extractors. Detailed instructions on how to execute these scripts and conduct the comparative evaluation are outlined in chapter 3.

6.2.7 Inference Time Analysis

ORB

Listing 6.1: ORB Inference Comparison script

```
import cv2
from time import time
import glob
import os
import logging
import numpy as np

class ImageLoader(object):
    def __init__(self, filepath: str):
        self.images = (
            glob.glob(os.path.join(filepath, "*.png"))
            + glob.glob(os.path.join(filepath, "*.jpg"))
            + glob.glob(os.path.join(filepath, "*.ppm"))
        )
        self.images.sort()
        self.num_images = len(self.images)
```

```
        logging.info("Loading %s images", {self.num_images})
        self.mode = "images"

    def __getitem__(self, item):
        filename = self.images[item]
        img = cv2.imread(filename)
        return img

    def __len__(self):
        return self.num_images

def measure(orb_extractor, image_loader) :
    timings = []
    print("Runing benchmark on {} images".format(len(
        image_loader)))
    for image in image_loader:
        # image: (H, W, C)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        start_time = time()
        kpts, dess = orb.detectAndCompute(image,None)
        end_time = time()
        duration = (end_time - start_time) * 1000  # convert to
            ms.
        timings.append(duration)

    print(f"mean: {np.mean(timings):.2f}ms")
    print(f"median: {np.median(timings):.2f}ms")
    print(f"min: {np.min(timings):.2f}ms")
    print(f"max: {np.max(timings):.2f}ms")

def keypoints_to_img(keypoints,img_tensor):
    # use if keypoints is a numpy array
    _, _, h, w = img_tensor.shape
    wh = np.array([w - 1, h - 1], dtype=np.float32)

    keypoints = wh * (keypoints + 1) / 2
    return keypoints

if __name__ == "__main__":
    orb = cv2.ORB_create()
    dataset_path={"MH01":"/home/nembot/datasets/MH01","MH02":"/
        home/nembot/datasets/MH02","V101":"/home/nembot/datasets/
        V101","V201":"/home/nembot/datasets/V201"}

    for _ in range(5):
        for name,path in dataset_path.items():
            image_path=path+"/mav0/cam0/data"
            image_loader = ImageLoader(image_path)
```

```
        print("Testing ORB Feature Extractor on {} dataset".
            format(name))
        measure(orb, image_loader)
```

Pytorch ALIKED

Listing 6.2: pytorch ALIKED Inference Comparison script

```
import cv2
from tqdm import tqdm
from time import time
import numpy as np
import glob
import os
import logging
from nets.aliked import ALIKED

class ImageLoader(object):
    def __init__(self, filepath: str):
        self.images = (
            glob.glob(os.path.join(filepath, "*.png"))
            + glob.glob(os.path.join(filepath, "*.jpg"))
            + glob.glob(os.path.join(filepath, "*.ppm"))
        )
        self.images.sort()
        self.num_images = len(self.images)
        logging.info("Loading %s images", {self.num_images})
        self.mode = "images"

    def __getitem__(self, item):
        filename = self.images[item]
        img = cv2.imread(filename)
        return img

    def __len__(self):
        return self.num_images

def measure(aliked_model, image_loader) :
    print("Runing benchmark on {} images".format(len(
        image_loader)))
    timings = []
    for image in tqdm(image_loader):
        # image: (H, W, C)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        start_time = time()
        keypoints, scores, descriptors = aliked_model.run(image)
        end_time = time()
```

```
            duration = (end_time - start_time) * 1000  # convert to
                ms.
            timings.append(duration)

        print(f"mean: {np.mean(timings):.2f}ms")
        print(f"median: {np.median(timings):.2f}ms")
        print(f"min: {np.min(timings):.2f}ms")
        print(f"max: {np.max(timings):.2f}ms")
    def keypoints_to_img(keypoints,img_tensor):
        # use if keypoints is a numpy array
        _, _, h, w = img_tensor.shape
        wh = np.array([w - 1, h - 1], dtype=np.float32)

        keypoints = wh * (keypoints + 1) / 2
        return keypoints

    if __name__ == "__main__":
        dataset_path = {"MH01": "/home/nembot/datasets/MH01", "MH02"
            : "/home/nembot/datasets/MH02",
                        "V101": "/home/nembot/datasets/V101", "V201"
                            : "/home/nembot/datasets/V201"}

        for _ in range(5):
            for name, path in dataset_path.items():
                print("Testing pytorch ALIKED Feature Extractor on
                    {} dataset".format(name))
                image_path = path + "/mav0/cam0/data"
                image_loader = ImageLoader(image_path)
                model = ALIKED(model_name="aliked-n32", top_k=1000)
                measure(model,image_loader)
```

ONNX ALIKED

Listing 6.3: ONNX ALIKED Inference Comparison script

```
import cv2
from torchvision.transforms import ToTensor
import onnxruntime
from tqdm import tqdm
from time import time
import torch
import numpy as np
import glob
import os
import logging

class ImageLoader(object):
    def __init__(self, filepath: str):
```

```
            self.images = (
                glob.glob(os.path.join(filepath, "*.png"))
                + glob.glob(os.path.join(filepath, "*.jpg"))
                + glob.glob(os.path.join(filepath, "*.ppm"))
            )
            self.images.sort()
            self.num_images = len(self.images)
            logging.info("Loading %s images", {self.num_images})
            self.mode = "images"

        def __getitem__(self, item):
            filename = self.images[item]
            img = cv2.imread(filename)
            return img

        def __len__(self):
            return self.num_images

    def measure(ort_session, image_loader):
        print("Runing benchmark on {} images".format(len(
            image_loader)))
        timings = []
        for image in tqdm(image_loader):
            # image: (H, W, C)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image_tensor = (
                ToTensor()(image).to("cuda").unsqueeze(0)
            )
            input_data = [image_tensor]
            defined_inputs = ort_session.get_inputs()
            ort_inputs = {
                defined_inputs[index].name: to_numpy(input_data[
                    index])
                for index in range(len(defined_inputs))
            }

            start_time = time()

            pred_onnx = ort_session.run(
                [
                    "keypoints",
                    "descriptors",
                    "scores",
                ],
                ort_inputs,
            )

            end_time = time()
```

```
            duration = (end_time - start_time) * 1000  # convert to
                ms.
            timings.append(duration)

        print(f"mean: {np.mean(timings):.2f}ms")
        print(f"median: {np.median(timings):.2f}ms")
        print(f"min: {np.min(timings):.2f}ms")
        print(f"max: {np.max(timings):.2f}ms")

    def to_numpy(tensor: torch.Tensor):
        return (
            tensor.detach().cpu().numpy()
            if tensor.requires_grad
            else tensor.cpu().numpy()
        )

    def keypoints_to_img(keypoints,img_tensor):
        # use if keypoints is a numpy array
        _, _, h, w = img_tensor.shape
        wh = np.array([w - 1, h - 1], dtype=np.float32)

        keypoints = wh * (keypoints + 1) / 2
        return keypoints

    if __name__ == '__main__':
        onnx_model_path = "./converted_onnx_models/aliked-n32-top1k-
            euroc.onnx"
        dataset_path = {"MH01": "/home/nembot/datasets/MH01", "MH02"
            : "/home/nembot/datasets/MH02",
                        "V101": "/home/nembot/datasets/V101", "V201"
                            : "/home/nembot/datasets/V201"}

        for _ in range(5):
            for name, path in dataset_path.items():
                print("Testing Onnx ALIKED Feature Extractor on {}
                    dataset".format(name))
                image_path = path + "/mav0/cam0/data"
                image_loader = ImageLoader(image_path)
                ort_session = onnxruntime.InferenceSession(
                    onnx_model_path)
                measure(ort_session,image_loader)
```

TensorRT ALIKED

Listing 6.4: TensorRT ALIKED Inference Comparison script

```
from trt_model import LOGGER_DICT, TRTInference
import cv2
```

```
from time import time
from torchvision.transforms import ToTensor
import numpy as np
import glob
import os
import logging
from tqdm import tqdm

class ImageLoader(object):
    def __init__(self, filepath: str):
        self.images = (
            glob.glob(os.path.join(filepath, "*.png"))
            + glob.glob(os.path.join(filepath, "*.jpg"))
            + glob.glob(os.path.join(filepath, "*.ppm"))
        )
        self.images.sort()
        self.num_images = len(self.images)
        logging.info("Loading %s images", {self.num_images})
        self.mode = "images"

    def __getitem__(self, item):
        filename = self.images[item]
        img = cv2.imread(filename)
        return img

    def __len__(self):
        return self.num_images

def measure(engine, image_loader) :
    print("Runing benchmark on {} images".format(len(
        image_loader)))
    timings = []
    for image in tqdm(image_loader):
        # image: (H, W, C)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image_tensor = ToTensor()(image).unsqueeze(0)  # (B, C,
            H, W)
        image = image_tensor.numpy()

        start_time = time()
        engine.infer(image)
        end_time = time()
        duration = (end_time - start_time) * 1000  # convert to
            ms.
        timings.append(duration)

    print(f"mean: {np.mean(timings):.2f}ms")
    print(f"median: {np.median(timings):.2f}ms")
```

```
    print(f"min: {np.min(timings):.2f}ms")
    print(f"max: {np.max(timings):.2f}ms")

def keypoints_to_img(keypoints,img_tensor):

    # use if keypoints is a numpy array
    _, _, h, w = img_tensor.shape
    wh = np.array([w - 1, h - 1], dtype=np.float32)

    keypoints = wh * (keypoints + 1) / 2
    return keypoints
if __name__ == '__main__':
    trt_model_path = "converted_trt_models/aliked-n32-top1k-
        euroc.trt"
    dataset_path = {"MH01": "/home/nembot/datasets/MH01", "MH02"
        : "/home/nembot/datasets/MH02",
                    "V101": "/home/nembot/datasets/V101", "V201"
                        : "/home/nembot/datasets/V201"}
    warmup_img_loc = "./assets/euroc/1403636620963555584.png"  #
        "/home/nembot/Datasets/EuRoc/MH01/mav0/cam0/data
        /1403636642563555584.png"
    trt_logger = LOGGER_DICT["verbose"]
    model_name = "aliked-n32"
    engine = TRTInference(trt_model_path, model_name, trt_logger
        )

    #Warming up TRT Engine
    warmup_image = cv2.imread(warmup_img_loc)
    warmup_image = cv2.cvtColor(warmup_image, cv2.COLOR_BGR2RGB)
    print("Starting warm-up ...")
    image_tensor = ToTensor()(warmup_image).unsqueeze(0)  # (B,
        C, H, W)
    image = image_tensor.numpy()
    image_w = image
    num_iterations = 100  # number of iterartions to warmup the
        model
    for _ in range(num_iterations):
        engine.infer(image_w)
    print("Warm-up done!")

    for _ in range(5):
        for name, path in dataset_path.items():
            print("Testing TensorRT ALIKED Feature Extractor on
                {} dataset".format(name))
            image_path = path + "/mav0/cam0/data"
            image_loader = ImageLoader(image_path)
            measure(engine, image_loader)
```

6.2.8 Keypoint Generation Analysis

6.2.8.1 ORB

Listing 6.5: ORB Keypoint Generation Comparison script

```
import cv2
from time import time
import glob
import os
import logging
import numpy as np

class ImageLoader(object):
    def __init__(self, filepath: str):
        self.images = (
            glob.glob(os.path.join(filepath, "*.png"))
            + glob.glob(os.path.join(filepath, "*.jpg"))
            + glob.glob(os.path.join(filepath, "*.ppm"))
        )
        self.images.sort()
        self.num_images = len(self.images)
        logging.info("Loading %s images", {self.num_images})
        self.mode = "images"

    def __getitem__(self, item):
        filename = self.images[item]
        img = cv2.imread(filename)
        return img

    def __len__(self):
        return self.num_images

def measure(orb_extractor, image_loader) :
    keypoint_count = []
    print("Runing benchmark on {} images".format(len(
       image_loader)))
    for image in image_loader:
        # image: (H, W, C)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        kpts, dess = orb.detectAndCompute(image,None)
        keypoint_count.append(len(kpts))

    print(f"mean: {np.mean(keypoint_count):.2f} keypoints")
    print(f"median: {np.median(keypoint_count):.2f} keypoints")
    print(f"min: {np.min(keypoint_count):.2f} keypoints")
    print(f"max: {np.max(keypoint_count):.2f} keypoints")
```

```
def keypoints_to_img(keypoints,img_tensor):
    # use if keypoints is a numpy array
    _, _, h, w = img_tensor.shape
    wh = np.array([w - 1, h - 1], dtype=np.float32)

    keypoints = wh * (keypoints + 1) / 2
    return keypoints

if __name__ == "__main__":
    orb = cv2.ORB_create()
    dataset_path={"MH01":"/home/nembot/datasets/MH01","MH02":"/
       home/nembot/datasets/MH02","V101":"/home/nembot/datasets/
       V101","V201":"/home/nembot/datasets/V201"}

    for _ in range(5):
        for name,path in dataset_path.items():
            image_path=path+"/mav0/cam0/data"
            image_loader = ImageLoader(image_path)
            print("Testing ORB Feature Extractor on {} dataset".
               format(name))
            measure(orb, image_loader)
```

6.2.8.2 ALIKED

Listing 6.6: ALIKED Keypoint Generation Comparison script

```
import cv2
from tqdm import tqdm
import numpy as np
import glob
import os
import logging
from nets.aliked import ALIKED
import warnings
warnings.filterwarnings("ignore")

class ImageLoader(object):
    def __init__(self, filepath: str):
        self.images = (
            glob.glob(os.path.join(filepath, "*.png"))
            + glob.glob(os.path.join(filepath, "*.jpg"))
            + glob.glob(os.path.join(filepath, "*.ppm"))
        )
        self.images.sort()
        self.num_images = len(self.images)
        logging.info("Loading %s images", {self.num_images})
        self.mode = "images"
```

```
    def __getitem__(self, item):
        filename = self.images[item]
        img = cv2.imread(filename)
        return img

    def __len__(self):
        return self.num_images

def measure(aliked_model, image_loader):
    print("Runing benchmark on {} images".format(len(
        image_loader)))
    keypoint_count = []
    for image in tqdm(image_loader):
        # image: (H, W, C)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        keypoints = aliked_model.run(image)['keypoints']
        keypoint_count.append(len(keypoints))
        # print(len(keypoints))


    print(f"mean: {np.mean(keypoint_count):.2f} keypoints")
    print(f"median: {np.median(keypoint_count):.2f} keypoints")
    print(f"min: {np.min(keypoint_count):.2f} keypoints")
    print(f"max: {np.max(keypoint_count):.2f} keypoints")
def keypoints_to_img(keypoints,img_tensor):
    # use if keypoints is a numpy array
    _, _, h, w = img_tensor.shape
    wh = np.array([w - 1, h - 1], dtype=np.float32)

    keypoints = wh * (keypoints + 1) / 2
    return keypoints

if __name__ == "__main__":
    dataset_path = {"MH01": "/home/nembot/datasets/MH01", "MH02"
        : "/home/nembot/datasets/MH02",
                    "V101": "/home/nembot/datasets/V101", "V201"
                        : "/home/nembot/datasets/V201"}

    for _ in range(5):
        for name, path in dataset_path.items():
            print("Testing pytorch ALIKED Feature Extractor on
                {} dataset".format(name))
            image_path = path + "/mav0/cam0/data"
            image_loader = ImageLoader(image_path)
            model = ALIKED(model_name="aliked-n32", top_k=-1,
                scores_th=0.05)
            measure(model,image_loader)
```

Appendix C : Comparison Screenshots

This section presents screenshots of the outputs generated during the comparative analysis of the various feature extractors.

Inference Time

ORB

```
/home/nembot/python/venv/aliked/bin/python3.11 /home/nembot/aliked-onnx/testorb.py
Testing ORB Feature Extractor on MH01 dataset
Runing benchmark on 3682 images
mean: 5.38ms
median: 4.82ms
min: 3.20ms
max: 1668.55ms
Testing ORB Feature Extractor on MH02 dataset
Runing benchmark on 3040 images
mean: 4.84ms
median: 4.91ms
min: 3.07ms
max: 10.41ms
Testing ORB Feature Extractor on V101 dataset
Runing benchmark on 2912 images
mean: 4.00ms
median: 3.96ms
min: 2.29ms
max: 9.91ms
Testing ORB Feature Extractor on V201 dataset
Runing benchmark on 2280 images
mean: 4.20ms
median: 3.98ms
min: 2.47ms
max: 14.54ms
```

Figure 6.2: ORB Feature Extractor Inference Time Performance Snapshot Across Four Datasets

pytorch ALIKED

```
/home/nembot/python/venv/aliked/bin/python3.11 /home/nembot/aliked-onnx/testaliked.py
Testing pytorch ALIKED Feature Extractor on MH01 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 3682 images
100%|██████████| 3682/3682 [02:41<00:00, 22.86it/s]
mean: 41.13ms
median: 40.46ms
min: 39.81ms
max: 391.55ms
Testing pytorch ALIKED Feature Extractor on MH02 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 3040 images
100%|██████████| 3040/3040 [02:12<00:00, 22.91it/s]
mean: 40.93ms
median: 40.48ms
min: 39.68ms
max: 83.54ms
Testing pytorch ALIKED Feature Extractor on V101 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 2912 images
100%|██████████| 2912/2912 [02:05<00:00, 23.18it/s]
mean: 40.50ms
median: 40.33ms
min: 39.71ms
max: 66.78ms
Testing pytorch ALIKED Feature Extractor on V201 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 2280 images
100%|██████████| 2280/2280 [01:38<00:00, 23.23it/s]
mean: 40.24ms
median: 40.16ms
min: 39.65ms
max: 43.83ms
```

Figure 6.3: pytorch ALIKED Feature Extractor Inference Time Performance Snapshot Across Four Datasets

ONNX ALIKED

```
/home/nembot/python/venv/aliked/bin/python3.11 /home/nembot/aliked-onnx/testonnx.py
Testing Onnx ALIKED Feature Extractor on MH01 dataset
Runing benchmark on 3682 images
100%|██████████| 3682/3682 [43:53<00:00,  1.40it/s]
mean: 691.71ms
median: 660.76ms
min: 631.76ms
max: 1358.32ms
Testing Onnx ALIKED Feature Extractor on MH02 dataset
Runing benchmark on 3040 images
100%|██████████| 3040/3040 [34:16<00:00,  1.48it/s]
mean: 655.58ms
median: 644.99ms
min: 628.59ms
max: 990.31ms
Testing Onnx ALIKED Feature Extractor on V101 dataset
  0%|          | 0/2912 [00:00<?, ?it/s]Runing benchmark on 2912 images
100%|██████████| 2912/2912 [32:06<00:00,  1.51it/s]
mean: 643.13ms
median: 638.35ms
min: 629.41ms
max: 1170.09ms
Testing Onnx ALIKED Feature Extractor on V201 dataset
Runing benchmark on 2280 images
100%|██████████| 2280/2280 [24:56<00:00,  1.52it/s]
mean: 637.89ms
median: 636.03ms
min: 624.60ms
max: 828.30ms
```

Figure 6.4: ONNX ALIKED Feature Extractor Inference Time Performance Snapshot Across Four Datasets

TensorRT ALIKED

```
Starting warm-up ...
Warm-up done!
Testing TensorRT ALIKED Feature Extractor on MH01 dataset
Runing benchmark on 3682 images
100%|██████████| 3682/3682 [01:19<00:00, 46.43it/s]
  0%|          | 0/3040 [00:00<?, ?it/s]mean: 17.86ms
median: 17.86ms
min: 17.42ms
max: 18.86ms
Testing TensorRT ALIKED Feature Extractor on MH02 dataset
Runing benchmark on 3040 images
100%|██████████| 3040/3040 [01:06<00:00, 45.48it/s]
mean: 17.99ms
median: 17.93ms
min: 17.50ms
max: 20.48ms
Testing TensorRT ALIKED Feature Extractor on V101 dataset
  0%|          | 0/2912 [00:00<?, ?it/s]Runing benchmark on 2912 images
100%|██████████| 2912/2912 [01:07<00:00, 43.19it/s]
mean: 18.20ms
median: 18.08ms
min: 17.38ms
max: 21.70ms
Testing TensorRT ALIKED Feature Extractor on V201 dataset
Runing benchmark on 2280 images
100%|██████████| 2280/2280 [00:51<00:00, 44.28it/s]
mean: 18.16ms
median: 18.05ms
min: 17.54ms
max: 20.47ms
```

Figure 6.5: TensorRT ALIKED Feature Extractor Inference Time Performance Snapshot Across Four Datasets

Keypoint Generation

ORB

```
/home/nembot/python/venv/aliked/bin/python3.11 /home/nembot/aliked-onnx/testKeypointOrb.py
Testing ORB Feature Extractor on MH01 dataset
Runing benchmark on 3682 images
mean: 499.99 keypoints
median: 500.00 keypoints
min: 488.00 keypoints
max: 500.00 keypoints
Testing ORB Feature Extractor on MH02 dataset
Runing benchmark on 3040 images
mean: 499.99 keypoints
median: 500.00 keypoints
min: 493.00 keypoints
max: 500.00 keypoints
Testing ORB Feature Extractor on V101 dataset
Runing benchmark on 2912 images
mean: 493.28 keypoints
median: 500.00 keypoints
min: 181.00 keypoints
max: 500.00 keypoints
Testing ORB Feature Extractor on V201 dataset
Runing benchmark on 2280 images
mean: 499.47 keypoints
median: 500.00 keypoints
min: 263.00 keypoints
max: 500.00 keypoints
```

Figure 6.6: ORB Feature Extractor Keypoint Generation Performance Snapshot Across Four Datasets

ALIKED

```
/home/nembot/python/venv/aliked/bin/python3.11 /home/nembot/aliked-onnx/testaliked2.py
Testing pytorch ALIKED Feature Extractor on MH01 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 3682 images
100%|██████████| 3682/3682 [02:37<00:00, 23.42it/s]
mean: 1140.75 keypoints
median: 1139.00 keypoints
min: 466.00 keypoints
max: 1898.00 keypoints
Testing pytorch ALIKED Feature Extractor on MH02 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 3040 images
100%|██████████| 3040/3040 [02:08<00:00, 23.59it/s]
mean: 1135.65 keypoints
median: 1130.00 keypoints
min: 321.00 keypoints
max: 1992.00 keypoints
Testing pytorch ALIKED Feature Extractor on V101 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 2912 images
100%|██████████| 2912/2912 [02:00<00:00, 24.16it/s]
mean: 931.05 keypoints
median: 946.00 keypoints
min: 337.00 keypoints
max: 1715.00 keypoints
Testing pytorch ALIKED Feature Extractor on V201 dataset
loading /home/nembot/aliked-onnx/models/aliked-n32.pth
Runing benchmark on 2280 images
100%|██████████| 2280/2280 [01:32<00:00, 24.68it/s]
mean: 798.15 keypoints
median: 823.00 keypoints
min: 212.00 keypoints
max: 1926.00 keypoints
```

Figure 6.7: ALIKED Feature Extractor Keypoint Generation Performance Snapshot Across Four Datasets

Figure 6.8: Visualisation of ONNX version of ALIKED Feature Extractor on MH01 Dataset

Figure 6.9: Visualisation of TensorRT version of ALIKED Feature Extractor on MH01 Dataset

Appendix D : ALIKED Tensor RT 10.3 Model Script

This section provides the modified TensorRT model generation script, which was utilized to create the optimized version of ALIKED employed in the comparative analysis.

Listing 6.7: ALIKED Tensor RT 10.3 Model Script

```
from typing import Any
import numpy as np
import pycuda.autoinit
import pycuda.driver as cuda
import tensorrt as trt

from torchvision.transforms import ToTensor
from nets.aliked import ALIKED_CFGS


LOGGER_DICT = {
    "warning": trt.Logger(trt.Logger.WARNING),
    "info": trt.Logger(trt.Logger.INFO),
```

```
    "verbose": trt.Logger(trt.Logger.VERBOSE),
}


class TRTInference:
    def __init__(
        self,
        trt_engine_path: str,
        model_type: str,
        trt_logger: trt.Logger,
    ):
        self.trt_logger = trt_logger

        # get configurations
        _, _, _, _, self.dim, _, _ = [
            v for _, v in ALIKED_CFGS[model_type].items()
        ]

        self.cfx = cuda.Device(0).make_context()
        stream = cuda.Stream()

        trt.init_libnvinfer_plugins(self.trt_logger, "")
        runtime = trt.Runtime(self.trt_logger)

        # deserialize engine
        with open(trt_engine_path, "rb") as trt_engine_file:
            buf = trt_engine_file.read()
            engine = runtime.deserialize_cuda_engine(buf)
        context = engine.create_execution_context()

        # prepare buffer
        host_inputs = []
        cuda_inputs = []
        host_outputs = []
        cuda_outputs = []

        tensor_names = [
            engine.get_tensor_name(index)
            for index in range(engine.num_io_tensors)
        ]
        tensor_name = engine.get_tensor_name(0)  # input tensor
        context.set_input_shape(tensor_name,(1, 3, 480, 752) )
          # use your input_shape
        assert context.all_binding_shapes_specified
        for tensor_name in tensor_names:
            if engine.get_tensor_mode(tensor_name) == trt.
               TensorIOMode.INPUT:
```

```
                context.set_input_shape(tensor_name, (1, 3, 480,
                    752))  # use your input_shape
                assert context.all_binding_shapes_specified
            msg = "\n=================================="
            data_type = (
                np.float32
                if engine.get_tensor_dtype(tensor_name) == trt.
                   DataType.FLOAT
                else np.int32
            )
            msg += f"\n{tensor_name}: {data_type}"
            size = trt.volume(engine.get_tensor_shape(
               tensor_name))
            msg += (
                "\nengine.get_tensor_shape(tensor_name): "
                f"{engine.get_tensor_shape(tensor_name)}"
            )
            msg += f"\nsize: {size}"
            msg += "\n=================================="
            self.trt_logger.log(trt.Logger.INFO, msg)
            host_mem = cuda.pagelocked_empty(size, data_type)
            cuda_mem = cuda.mem_alloc(host_mem.nbytes)
            #Reference : https://forums.developer.nvidia.com/t/
               tensorrt-v10-inference-using-context-execute-
               async-v3/289771/2
            context.set_tensor_address(tensor_name, cuda_mem)
            if engine.get_tensor_mode(tensor_name) == trt.
               TensorIOMode.INPUT:
                host_inputs.append(host_mem)
                cuda_inputs.append(cuda_mem)
            else:  # == trt.TensorIOMode.OUTPUT
                host_outputs.append(host_mem)
                cuda_outputs.append(cuda_mem)

        # store
        self.stream = stream
        self.context = context
        self.engine = engine

        self.host_inputs = host_inputs
        self.cuda_inputs = cuda_inputs
        self.host_outputs = host_outputs
        self.cuda_outputs = cuda_outputs

    def warmup(self, image: np.ndarray, num_iterations: int = 3)
        -> None:
        print("Starting warm-up ...")
        for _ in range(num_iterations):
```

```
            self.run(image)
        print("Warm-up done!")

    def run(self, image):
        image_tensor = ToTensor()(image).unsqueeze(0)  # (B, C,
            H, W)
        image = image_tensor.numpy()
        # image = np.expand_dims(image.transpose(2, 0, 1), 0) #
            (1, C, H, W)
        _, _, h, w = image.shape
        wh = np.array([w - 1, h - 1])
        # print(self.dim)
        keypoints, descriptors, scores, = self.infer(image) #
            order of output changed based on findings from self.
            dims
        keypoints = keypoints.reshape(-1, 2)
        keypoints = wh * (keypoints + 1) / 2


        return {
            "keypoints": keypoints.reshape(-1, 2),  # N 2
            "descriptors": descriptors.reshape(-1, self.dim),  #
                N D
            "scores": scores,  # B N D
        }

    def infer(self, image):
        self.cfx.push()

        # restore
        stream = self.stream
        context = self.context
        engine = self.engine

        host_inputs = self.host_inputs
        cuda_inputs = self.cuda_inputs
        host_outputs = self.host_outputs
        cuda_outputs = self.cuda_outputs

        # Copy data to GPU.
        for index, host_input in enumerate(host_inputs):
            pagelocked_buffer = host_input
            flattened_data = image[index].flatten()
            data_size = flattened_data.shape[0]
            np.copyto(pagelocked_buffer[:data_size],
                flattened_data)
        for cuda_inp, host_inp in zip(cuda_inputs, host_inputs):
            cuda.memcpy_htod_async(cuda_inp, host_inp, stream)
```

```
        # Inference.
        context.execute_async_v3(
            stream_handle=stream.handle
        )

        # Copy to host.
        for cuda_out, host_out in zip(cuda_outputs, host_outputs
            ):
            cuda.memcpy_dtoh_async(host_out, cuda_out, stream)
        stream.synchronize()

        keypoints, scores, descriptors = host_outputs

        self.cfx.pop()
        return keypoints, scores, descriptors

    def __del__(self):
        self.cfx.pop()
```

Appendix E : Benchmarking ORB-SLAM3

Setting UP ORB-SLAM3

Installing Dependencies

1. **Pangolin V6** (stevenlovegrove, 2021):
   The Pangolin library, necessary for the visualization of ORB-SLAM3 results, was obtained from the official GitHub repository. The source code was compiled and built using CMake.

2. **OpenCV** (Bradski and Kaehler, 2000):
   OpenCV and the accompanying OpenCV contrib module, version 4.8.0, were downloaded from GitHub and subsequently built using CMake. OpenCV is integral to ORB-SLAM3, providing essential functionality for image and video manipulation during SLAM operations.

3. **Eigen V3.3.0** (Guennebaud and Jacob, 2010):
   Eigen version 3.3.0 was downloaded from the official GitHub repository and built using CMake. The Eigen3 library is employed by ORB-SLAM3 for efficient matrix and vector manipulations.

Building ORB-SLAM3

Once all the necessary dependencies were successfully installed, the next phase involved building the ORB-SLAM3 system itself. This was accomplished by executing the build.sh script located within the project's root directory. This shell script is designed to automate the build process by leveraging CMake to manage and compile the various components of ORB-SLAM3.

The build process begins with the compilation of key third-party libraries that ORB-SLAM3

relies on, including DBow2, g2o, and Sophus. DBow2, developed by Gálvez-López and Tardós (2012), is a library designed for bag-of-words image retrieval, and in ORB-SLAM3, it is used specifically for loop closure detection, a critical feature that ensures the system can recognize previously visited locations. The g2o library (Kümmerle et al., 2011) is used for graph optimization, enabling ORB-SLAM3 to refine its pose estimations by minimizing the overall error in the SLAM graph. Sophus (Schneider and GAP Team, 2022), another essential library, provides tools for performing calculations on Lie groups, which are mathematical structures that are particularly useful in 3D geometry and robotics.

After these dependencies were built successfully, the build.sh script proceeded to extract the ORB vocabulary. This vocabulary is a pre-trained set of features used by the DBoW2 (Gálvez-López and Tardós, 2012) library to perform efficient loop closure detection by matching current observations with those stored in the system's database. The extraction and proper setup of this vocabulary are crucial for the effective operation of ORB-SLAM3, as it directly impacts the system's ability to close loops and maintain an accurate map of the environment.

The final step of the build process involved compiling the ORB-SLAM3 source files themselves. The script meticulously manages this process, ensuring that all components are correctly linked and optimized for performance. Upon the successful completion of the script without any errors, the ORB-SLAM3 system was fully built and ready for deployment. At this stage, the system is prepared to perform SLAM tasks, allowing users to begin testing and benchmarking its performance under various conditions.

Running ORB-SLAM3

ORB-SLAM3 can operate in four distinct modes, each suited to different sensor configurations and application scenarios. These modes are:

- Monocular Mode
- Stereo Mode
- Monocular-Inertial Mode
- Stereo-Inertial Mode

Monocular Mode

In Monocular Mode, ORB-SLAM3 utilizes a single monocular camera to perform visual SLAM. For our experiments, we employed the 'cam0' data from the EuRoc dataset, which, while containing data from two cameras and inertial sensors, was used here to simulate operation with a single camera.

To run ORB-SLAM3 in Monocular Mode, the mono_euroc binary is executed with the following command:

```
./Examples/Monocular/mono_euroc ./Vocabulary/ORBvoc.txt
./Examples/Monocular/EuRoC.yaml ../datasets/MH01
./Examples/Monocular/EuRoC_TimeStamps/MH01.txt dataset-MH01_mono
```

This command requires five specific arguments:

1. **ORB Vocabulary Location**: The path to the file containing the ORB vocabulary, which is a pre-trained set of visual features used for loop closure and map management.
2. **Camera Parameters**: The intrinsic and extrinsic parameters of the camera, provided in a YAML file. These parameters include information such as focal length, principal point, and distortion coefficients, which are crucial for accurate image processing and feature extraction.
3. **Dataset Path**: The directory path to the dataset, which includes the image sequences used for SLAM processing.
4. **Timestamps Path**: The path to the file containing the timestamps for the images in the dataset, which ensures that the images are processed in the correct temporal order.
5. **Output File Name**: A name prefix for the files where the results of the SLAM process will be stored.

Upon execution, the mono_euroc binary uses these inputs to initialize and run the ORB-SLAM3 library. The visual SLAM process then generates two output files:

- **Trajectory File**: Containing the computed trajectory of the camera, named with f_outputFileName.
- **Keyframe Trajectory File**: Containing the trajectory of keyframes used in the SLAM process, named with kf_outputFileName.

These output files are essential for evaluating the performance of the SLAM system and for further analysis of the visual data.

Stereo Mode

In Stereo Mode, ORB-SLAM3 utilizes a pair of synchronized stereo cameras to perform visual SLAM, leveraging the depth information provided by the stereo configuration. This mode is particularly useful for environments where depth perception is critical for accurate mapping and localization.

For our experiments, we employed both the cam0 and cam1 cameras from the EuRoc dataset. This dataset provides synchronized stereo image pairs, which are essential for the stereo mode's operation.

To run ORB-SLAM3 in Stereo Mode, the stereo_euroc binary is used. The command to run ORB-SLAM3 in this mode is as follows:

```
./Examples/Stereo/stereo_euroc ./Vocabulary/ORBvoc.txt
./Examples/Stereo/EuRoC.yaml ../datasets/MH01
./Examples/Stereo/EuRoC_TimeStamps/MH01.txt dataset-MH01_stereo
```

This command requires five specific arguments:

1. **ORB Vocabulary Location**: Path to the file with the ORB vocabulary, here specified as ./Vocabulary/ORBvoc.txt.
2. **Camera Parameters**: YAML file containing the intrinsic and extrinsic parameters of the stereo cameras, provided as ./Examples/Stereo/EuRoC.yaml.
3. **Dataset Path**: Directory containing the stereo image sequences, here ../datasets/MH01.
4. **Timestamps Path**: File with timestamps for the stereo images, specified as ./Examples/Stereo/EuRoC_TimeStamps/MH01.txt.
5. **Output File Name**: Prefix for output files, set to dataset-MH01_stereo.

Executing this command initializes the ORB-SLAM3 system with the provided data. The SLAM process produces two key output files:

- **Trajectory File**: Contains the computed trajectory of the stereo system, named with f_dataset-MH01_stereo.
- **Keyframe Trajectory File**: Records the trajectory of keyframes used during SLAM, named with kf_dataset-MH01_stereo.

These outputs are crucial for evaluating the performance of the SLAM system in Stereo Mode, offering insights into depth estimation accuracy and overall mapping effectiveness.

Monocular-Inertial Mode

In Monocular-Inertial Mode, ORB-SLAM3 integrates visual data from a single monocular camera with inertial measurements from an IMU to enhance SLAM performance, particularly in terms of robustness and accuracy. For our setup, we utilized cam0 and imu0 data from the EuRoc dataset.

To run ORB-SLAM3 in Monocular-Inertial Mode, use the following command:

```
./Examples/Monocular-Inertial/mono_inertial_euroc ./Vocabulary/ORBvoc.txt
./Examples/Monocular-Inertial/EuRoC.yaml ../datasets/MH01
./Examples/Monocular-Inertial/EuRoC_TimeStamps/MH01.txt dataset-MH01_MonoI
```

This command requires five specific arguments:

1. **ORB Vocabulary Location**: Path to the ORB vocabulary file, specified as ./Vocabulary/ORBvoc.txt.
2. **Camera Parameters**: YAML file with the intrinsic and extrinsic parameters for both the monocular camera and the IMU, provided as ./Examples/Monocular-Inertial/EuRoC.yaml.

3. **Dataset Path**: Directory containing the monocular and IMU data, here ../datasets/MH01.
4. **Timestamps Path**: File with timestamps for the monocular images and IMU measurements, specified as ./Examples/Monocular-Inertial/EuRoC_TimeStamps/MH01.txt.
5. **Output File Name**: Prefix for the output files, set to dataset-MH01_MonoI.

Upon running this command, ORB-SLAM3 initializes with the provided visual and inertial data. The SLAM process generates two primary output files:

- **Trajectory File**: Contains the computed trajectory of the camera-IMU system, named with f_dataset-MH01_MonoI.
- **Keyframe Trajectory File**: Records the trajectory of keyframes used in the SLAM process, named with kf_dataset-MH01_MonoI.

These output files are essential for evaluating the performance and accuracy of ORB-SLAM3 in Monocular-Inertial Mode, particularly in terms of how well the system integrates visual and inertial data.

Stereo-Inertial Mode

In Stereo-Inertial Mode, ORB-SLAM3 combines visual data from two synchronized cameras with inertial measurements from an IMU to achieve enhanced SLAM performance. For this mode, we used cam0, cam1, and imu0 from the EuRoc dataset.

To run ORB-SLAM3 in Stereo-Inertial Mode, execute the following command:

```
./Examples/Stereo-Inertial/stereo_inertial_euroc ./Vocabulary/ORBvoc.txt
./Examples/Stereo-Inertial/EuRoC.yaml ../datasets/MH01
./Examples/Stereo-Inertial/EuRoC_TimeStamps/MH01.txt dataset-MH01_stereoI
```

This command requires five specific arguments:

1. **ORB Vocabulary Location**: Path to the ORB vocabulary file, ./Vocabulary/ORBvoc.txt.
2. **Camera Parameters**: YYAML file with the calibration parameters for the stereo cameras and IMU, ./Examples/Stereo-Inertial/EuRoC.yaml.
3. **Dataset Path**: Directory containing the stereo images and IMU data, ../datasets/MH01.
4. **Timestamps Path**: File with timestamps for the stereo images and IMU measurements, ./Examples/Stereo-Inertial/EuRoC_TimeStamps/MH01.txt.
5. **Output File Name**: Prefix for the results, dataset-MH01_stereoI.

The SLAM process produces two key output files:

- **Trajectory File**: Includes the computed trajectory, named with f_dataset-MH01_stereoI.
- **Keyframe Trajectory File**: Contains the keyframe trajectory, named with kf_dataset-MH01_stereoI.

These output files are essential for evaluating the performance and accuracy of ORB-SLAM3 in Stereo-Inertial Mode, particularly in terms of how well the system integrates visual and inertial data.

Problems Encountered

Missing Libraries

Missing GLEW

During the build process for Pangolin, a critical dependency of ORB-SLAM3, we encountered an error indicative of a missing "OpenGL Extension Wrangler Library" (GLEW), as illustrated in Figure 6.6. GLEW is essential for Pangolin's functionality, particularly for rendering images and videos. This issue arises when the operating system lacks the GLEW library, which is necessary for proper OpenGL extension management. To resolve this issue, the GLEW package can be installed using the Ubuntu apt package manager. This installation provides the required library files, allowing Pangolin to correctly handle OpenGL extensions and proceed with the build process. The absence of GLEW was a significant obstacle, but its installation was straightforward and resolved the rendering issues promptly.

Eigen3 Linking Issue

An additional problem encountered during the build process was related to the Eigen3 library, as depicted in Figure 6.7. This issue arises when CMake fails to locate the Eigen3 library during the linking stage of the build. Eigen3, a key dependency for ORB-SLAM3, is used extensively for matrix and vector operations. The solution to this problem involved creating a symbolic link from the include directories where Eigen3 files were installed to the locations expected by CMake. This linking issue is often due to discrepancies in directory paths between where the library files are installed and where CMake expects to find them. By establishing the appropriate symbolic links, we were able to align the library paths correctly, thereby resolving the linking problem and allowing the build process to continue.

Missing Boost

Another issue encountered was the absence of the Boost Library, as shown in Figure 6.8. Boost is a widely-used collection of C++ libraries that provide various functionalities, including support for multithreading, which is crucial for ORB-SLAM3. The CMake tool was unable to locate the Boost Library, leading to build failures. This issue was addressed by installing the Boost package using the Ubuntu apt package manager. Boost's installation ensured that all necessary libraries and headers were available for CMake to find during the build process. The integration of Boost is critical for implementing multithreading capabilities within ORB-SLAM3, and its absence could have significantly impeded the system's performance and functionality.

Memory and Resource Management During the Build Process

During the process of building ORB-SLAM3, the system encountered significant issues related to memory usage, specifically running out of RAM and swap space. These resource constraints led to repeated crashes, severely disrupting the build process and preventing successful compilation. The root cause of the problem was identified as the high default level of parallelism in the build process, which was overwhelming the available system resources.

CMake, the build system used by ORB-SLAM3, typically utilizes multiple threads to accelerate the compilation by parallelizing tasks across available CPU cores. While this approach can significantly reduce build times on systems with ample resources, it can also result in excessive memory consumption, particularly on systems with limited RAM or swap space. In this case, the default configuration attempted to use more threads than the system could handle, leading to memory exhaustion and subsequent crashes.

To mitigate this issue, a specific adjustment was made to the CMake build command by introducing the -j 2 flag. This flag explicitly limits the number of parallel threads used during the build process to two. By restricting the build to use only two threads, the peak memory usage was significantly reduced, thereby preventing the system from exhausting its available RAM and swap space. This adjustment allowed the build process to complete successfully without further incidents.

This experience underscores the importance of considering system resource limitations when configuring build environments, particularly for large and complex software projects like ORB-SLAM3. Limiting the number of parallel threads can be a crucial strategy for ensuring stability in systems with constrained resources, allowing the build process to proceed smoothly and efficiently without risking system crashes or other interruptions.

Misalignment and Pose Correction

A crucial aspect of evaluating a SLAM system's performance is its ability to accurately estimate the robot's pose (position and orientation) within the environment over time. During the initial implementation of ORB-SLAM3, a discrepancy was observed between the trajectory generated by the system and the known ground truth trajectory data. Ground truth data refers to highly accurate and verified information about the robot's movement path, often used for comparison and evaluation purposes.

This misalignment manifested in a significant Absolute Pose Error (APE) when comparing the estimated poses from the ORB-SLAM3 trajectory with the ground truth data. APE remains a common metric used in SLAM to quantify the difference between the estimated pose and the actual pose of the robot at each point in time. In this case, while the overall shape of the estimated trajectory might have resembled the ground truth path, a significant discrepancy existed in terms of the alignment and scale of the trajectory. This misalignment could lead to high APE values even though the general direction of the robot's movement appeared captured.

To rectify this misalignment issue and ensure accurate pose estimation, the project employed the Umeyama alignment within the EVO benchmarking tool (Grupp, 2017). While traditionally used for point cloud alignment, the Umeyama alignment (Umeyama, 1991) can also be applied to trajectory data. In this context, it estimates a rigid body transformation (including translation, rotation, and scaling) that minimizes the distance between corresponding poses in the estimated trajectory and the ground truth trajectory. By leveraging this method, the

project was able to effectively align and scale the ORB-SLAM3 trajectory with the ground truth data, significantly reducing the APE and ensuring that the estimated poses were more accurate.

Missing Map and Camera Viewer

An issue encountered during the operation of ORB-SLAM3 was the absence of the Map and Camera Viewer windows, which are essential for visualizing the SLAM process in real-time. This problem typically arises when the bUseViewer argument is set to false during the initialization of the ORB-SLAM3 object.

The bUseViewer argument controls whether the visualization components of ORB-SLAM3 are activated. When set to false, the Map and Camera Viewer windows are disabled, leading to a lack of visual feedback regarding the SLAM process. This can be particularly problematic as it prevents users from observing the map building and camera tracking in real time.

To resolve this issue, it is necessary to modify the initialization code of ORB-SLAM3 to set the bUseViewer argument to true. This adjustment ensures that the viewer components are enabled, allowing the Map and Camera Viewer windows to be displayed. Consequently, users will be able to monitor the SLAM process visually, facilitating better debugging and analysis.

Benchmarking ORB-SALM3

This section presents the benchmarking of the ORB-SLAM3 system (Campos et al., 2021) to evaluate its performance comprehensively. The benchmarking process aims to establish a clear understanding of ORB-SLAM3's capabilities by analyzing key performance metrics, which will serve as a baseline for further assessments in related applications.

6.2.8.3 Baseline ORB-SLAM3 Analysis

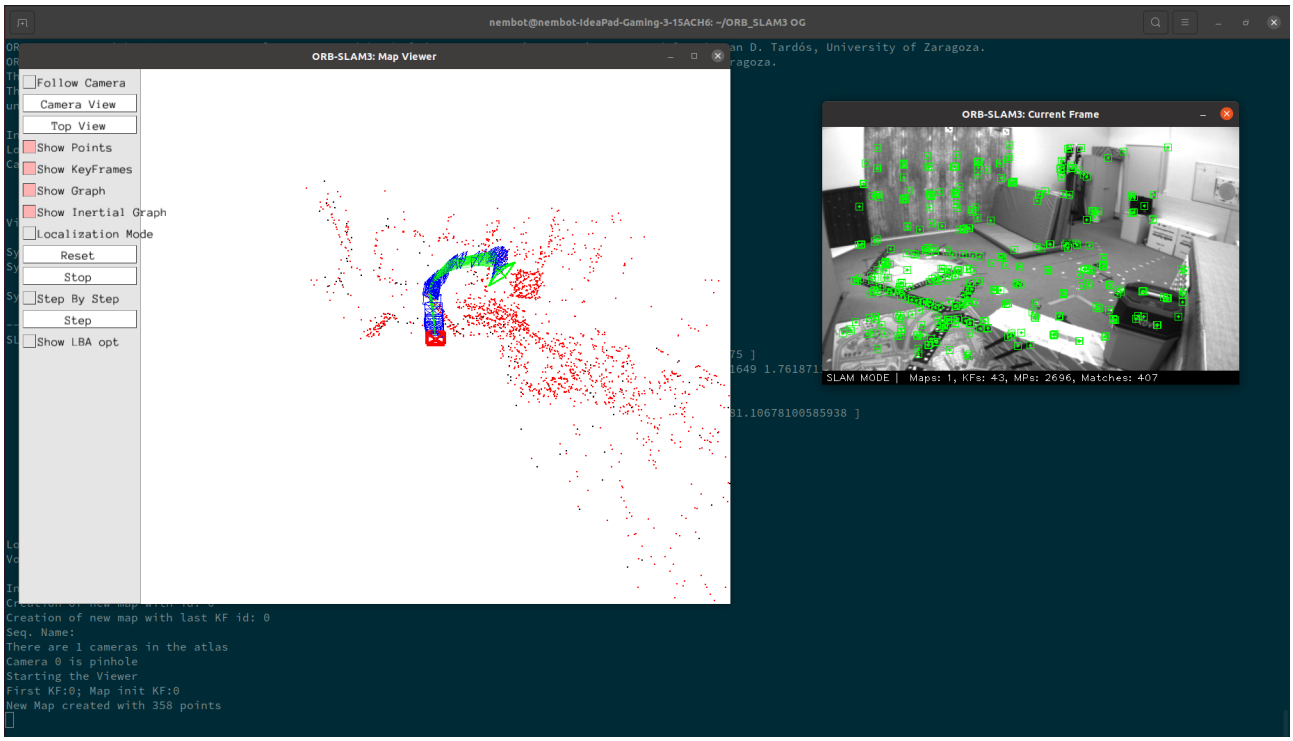

Figure 6.10: Working of ORB-SLAM3 on Euroc V101 dataset

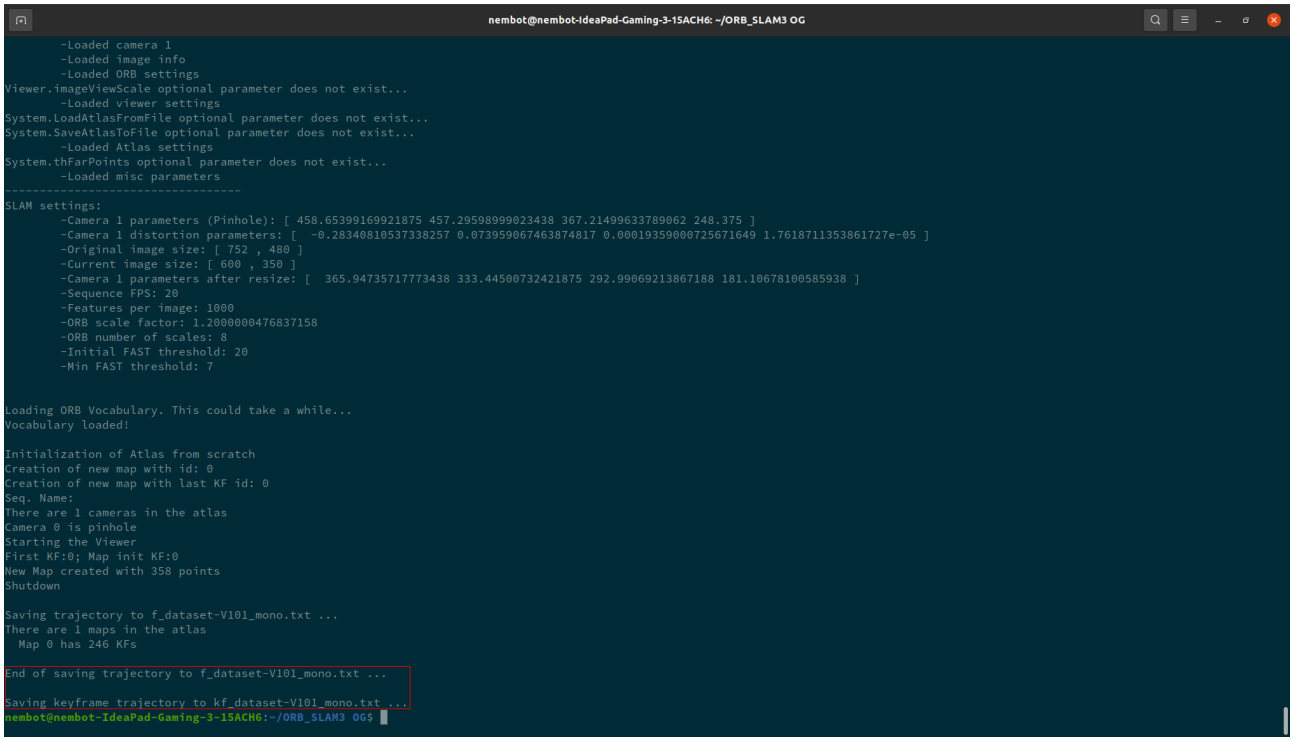

Figure 6.11: Output of the ORB-SLAM3

Figure 6.10 and Figure 6.11 provide insights into the operation of the baseline ORB-SLAM3 system. Figure 6.10 depicts the Map Viewer, showcasing the identified keypoints within the environment and the keyframes utilized for map construction. Additionally, the Current Frame displays the keypoints extracted from the live camera feed using the ORB feature extractor. Figure 6.11 visualizes the estimated camera trajectory, which is the primary output generated by ORB-SLAM3.

Benchmarking Methodology

The benchmarking process was meticulously designed to assess the performance of ORB-SLAM3 across a diverse range of environments by utilizing a comprehensive selection of datasets from the EuroC MAV dataset collection. The datasets chosen for this evaluation MH01, MH02, V101, and V201 were choosen which represent a variety of environmental conditions that a SLAM system might realistically encounter, including different levels of texture, illumination, and motion dynamics. This selection ensures that the benchmarking process captures the robustness and adaptability of ORB-SLAM3 under varying scenarios.

To thoroughly evaluate the system's performance, each dataset was tested under four distinct configurations: Monocular, Stereo, Monocular-Inertial, and Stereo-Inertial modes. These configurations are critical for understanding how the addition of inertial data and stereo vision influences the accuracy and reliability of the SLAM system. Monocular mode relies solely on a single camera feed, while Stereo mode leverages dual camera inputs to enhance depth perception. The Monocular-Inertial and Stereo-Inertial modes integrate inertial measurements, providing additional data that can improve trajectory estimation, especially in scenarios with rapid motion or poor visual features.

For each configuration, the ORB-SLAM3 system was executed five times to account for variability in performance due to factors such as initialization differences and random noise. The Root Mean Square Error (RMSE) of the Absolute Pose Error (APE) was computed for each individual run. APE is a critical metric that quantifies the deviation between the estimated and ground truth trajectories, offering a precise measure of the system's accuracy. By averaging the RMSE values from these five executions, a robust and reliable benchmark result was obtained for each dataset configuration, as summarized in Table 6.5. This approach ensures that the reported performance metrics are not skewed by outliers or anomalies, providing a more accurate reflection of the system's typical performance.

Benchmarking Results

The benchmarking results presented in this section provide a comprehensive overview of the performance of the ORB-SLAM3 system across multiple datasets and configurations. By systematically evaluating the system under various operational modes: Monocular, Stereo, Monocular-Inertial, and Stereo-Inertial on diverse EuroC datasets, the robustness and adaptability of ORB-SLAM3 can be thoroughly assessed.

Each table below encapsulates the results from repeated trials, ensuring that the data reflects a reliable measure of the system's accuracy and consistency. These trials were meticulously conducted to account for any variability that might arise from different environmental conditions and configurations, offering a holistic view of the system's performance.

The data collection process involved rigorous testing, with each configuration being executed

five times to ensure that the results are not only representative but also statistically sound. This approach underscores the importance of reproducibility and reliability in benchmarking, as it allows for the identification of patterns and trends that might not be immediately apparent from a single test run.

The presented tables serve as a foundational reference point for further analysis, offering valuable insights into how ORB-SLAM3 performs under different scenarios. The results captured here will play a crucial role in subsequent sections, where a deeper dive into the implications of these findings will be undertaken.

These tables are integral to understanding the broader implications of the benchmarking process, setting the stage for a detailed exploration of how the ORB-SLAM3 system navigates the challenges posed by varying dataset conditions and configurations. As we move forward, these results will be critically analyzed to uncover key performance trends and potential areas for enhancement.

Table 6.1: RMSE of APE for ORB-SLAM3 on MH01 dataset

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
Monocular Mode	0.039244	0.038408	0.037171	0.035134	0.036906	0.0373726
Stereo Mode	0.035304	0.036859	0.038141	0.037016	0.032881	0.0360402
Mono-Inertial Mode	0.022658	0.019283	0.028174	0.031555	0.030749	0.0264838
Stereo-Inertial Mode	0.0216	0.017413	0.019795	0.021912	0.020501	0.0202442

Table 6.2: RMSE of APE for ORB-SLAM3 on MH02 dataset

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
Monocular Mode	0.032609	0.032461	0.033502	0.031907	0.030939	0.0322836
Stereo Mode	0.035588	0.035936	0.048144	0.042661	0.039658	0.0403974
Mono-Inertial Mode	0.041248	0.050337	0.043488	0.044884	0.036347	0.0432608
Stereo-Inertial Mode	0.031146	0.026957	0.026905	0.037401	0.03354	0.0311898

Table 6.3: RMSE of APE for ORB-SLAM3 on V101 dataset

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
Monocular Mode	0.087669	0.088604	0.088416	0.088049	0.086789	0.0879054
Stereo Mode	0.0890000	0.086856	0.088958	0.087508	0.087879	0.0880402
Mono-Inertial Mode	0.032922	0.032974	0.032775	0.032734	0.032568	0.0327946
Stereo-Inertial Mode	0.034221	0.034174	0.034185	0.034229	0.033898	0.0341414

Table 6.4: RMSE of APE for ORB-SLAM3 on V201 dataset

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
Monocular Mode	0.032609	0.032461	0.033502	0.031907	0.030939	0.0322836
Stereo Mode	0.035588	0.035936	0.048144	0.042661	0.039658	0.0403974
Mono-Inertial Mode	0.041248	0.050337	0.043488	0.044884	0.036347	0.0432608
Stereo-Inertial Mode	0.031146	0.026957	0.026905	0.037401	0.03354	0.0311898

The averages presented in the following table consolidate the performance of ORB-SLAM3 across various datasets and configurations. These averages offer a high-level summary, encapsulating the system's overall behavior and providing a baseline for subsequent detailed analysis. This summary allows for a quick comparison between different modes and datasets, setting the stage for a deeper exploration of the underlying performance trends in the following sections.

Table 6.5: Average RMSE of APE for ORB-SLAM3 from all datasets

	MH01 Euroc	**MH02 Euroc**	**V101 Euroc**	**V202 Euroc**
Monocular Mode	0.0373726	0.0322836	0.0879054	0.1770838
Stereo Mode	0.0360402	0.0403974	0.0880402	0.06716
Mono-Inertial Mode	0.0264838	0.0432608	0.0327946	0.049213
Stereo-Inertial Mode	0.0202442	0.0311898	0.0341414	0.023122

Dataset-Specific Insights

MH01 Benchmarking

The MH01 dataset presents a controlled environment where the camera navigates through a large industrial building. It incorporates challenges such as varying light intensities and similar objects that can be difficult for a SLAM system to differentiate through feature extraction and descriptor matching. Table 6.1 demonstrates consistent results for Monocular and Stereo modes across the five runs. However, slight variations appear in the Monocular-Inertial and Stereo-Inertial modes, which are attributed to the ORB feature extractor's inability to consistently extract similar numbers of features from the images due to fluctuating lighting conditions. Interestingly, the APE error exhibits a decreasing trend from Monocular to Stereo, then to Monocular-Inertial, and finally Stereo-Inertial.

MH02 Benchmarking

The MH02 dataset shares similarities with MH01 but features a longer path, potentially posing a challenge for loop closure detection. Table 6.2 reveals consistent results for the Monocular mode across all runs. However, the data in the other three configurations exhibit inconsistencies.

V101 Benchmarking

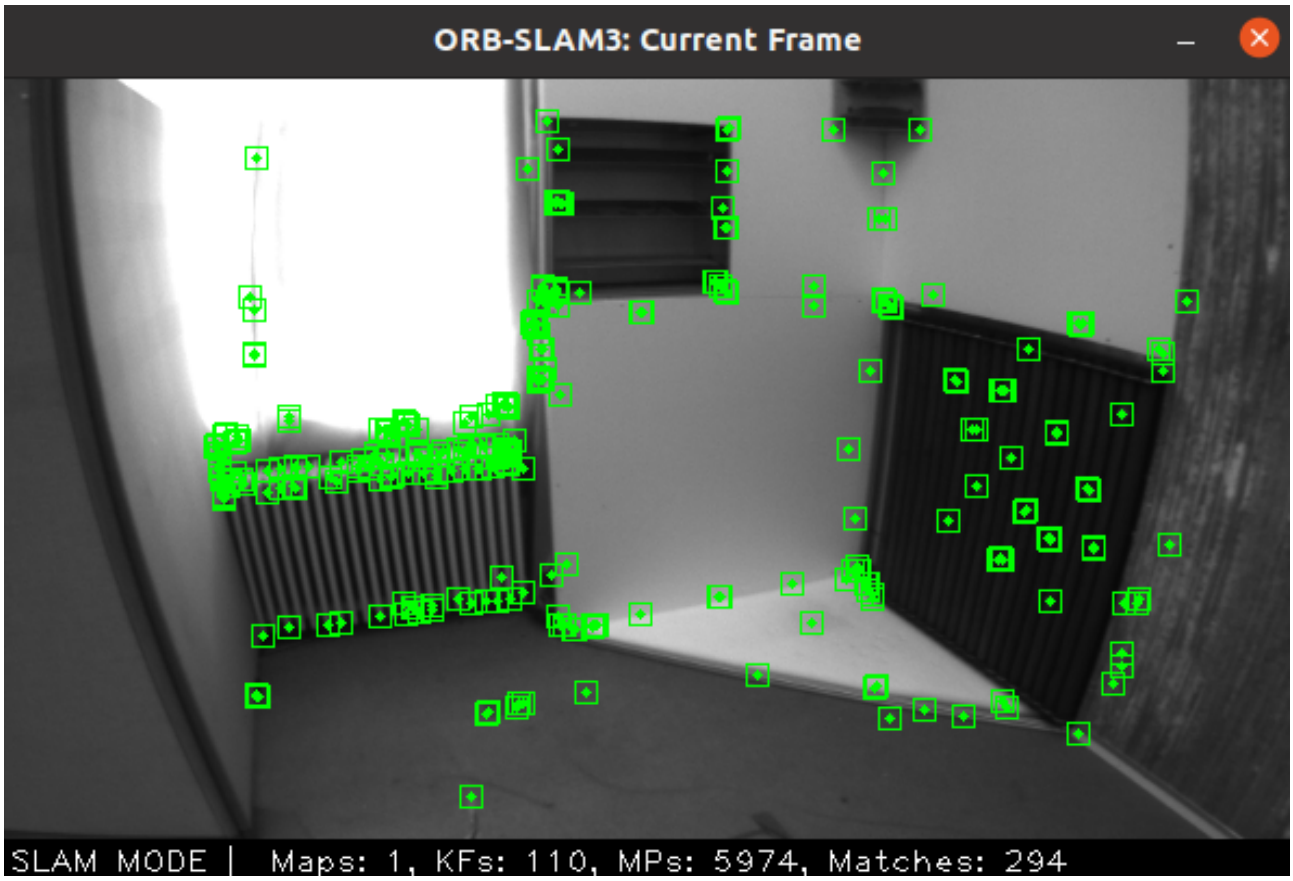

Figure 6.12: Visualization of ORB Feature Extraction in the V101 Dataset.

The V101 dataset comprises recordings within a room where most images are dominated by the walls, with only a few objects present. Feature extraction in this dataset primarily focuses on these objects as shown in Figure 6.12, posing a challenge for the SLAM system as it may encounter similar environments in real world applications. Table 6.3 showcases high APE for Monocular and Stereo modes, as these configurations rely solely on camera data for trajectory estimation. Conversely, the Monocular-Inertial and Stereo-Inertial modes exhibit lower and more consistent results. This improvement aligns with the findings of Mourikis and Roumeliotis (2007), where incorporating IMU data enhances pose estimation accuracy.

V201 Benchmarking

The V201 dataset also features an indoor environment with all the challenges present in V101. Additionally, the camera motion is faster in this dataset, leading to significant variations within scenes. The data in Table 6.4 reflects this issue, as the recorded APE is generally higher compared to V101, which has a similar environment. Along with the higher APE values, inconsistencies are also observed across all SLAM modes, highlighting the impact of scene variations on the existing ORB-SLAM3 system.

Consolidated Average Results

The average RMSE values of the APE across all datasets and configurations, as presented in Table 6.5, provide a comprehensive snapshot of ORB-SLAM3's overall performance. These consolidated results serve as a crucial reference point, highlighting the system's strengths and limitations under different operational modes and environmental conditions.

The patterns observed in the average results reinforce the dataset specific observations discussed earlier. For instance, the consistently lower errors in the Stereo-Inertial mode across all datasets underscore the significant advantage of incorporating inertial measurements into the SLAM pipeline, particularly in environments with challenging lighting or minimal features, such as those encountered in the V101 and V201 datasets. This mode's ability to maintain accuracy despite rapid scene changes, as seen in the V201 dataset, further validates its robustness.

On the other hand, the higher error rates observed in the Monocular and Stereo modes, especially in the V101 and V201 datasets, reflect the inherent challenges of relying solely on visual data in environments where feature scarcity and repetitive patterns can lead to unreliable feature matching and pose estimation. These results align with the earlier analysis, which noted the difficulties faced by ORB-SLAM3 in scenarios with limited distinctive features and rapid camera movements.

Overall, the consolidated averages not only encapsulate the system's response to different dataset conditions but also pave the way for a deeper examination of how ORB-SLAM3 can be further optimized. In the following sections, these findings will be dissected to identify key trends and areas where targeted improvements could enhance the system's performance, particularly in challenging environments.

Comparison with ORB-SLAM3 Benchmark Results

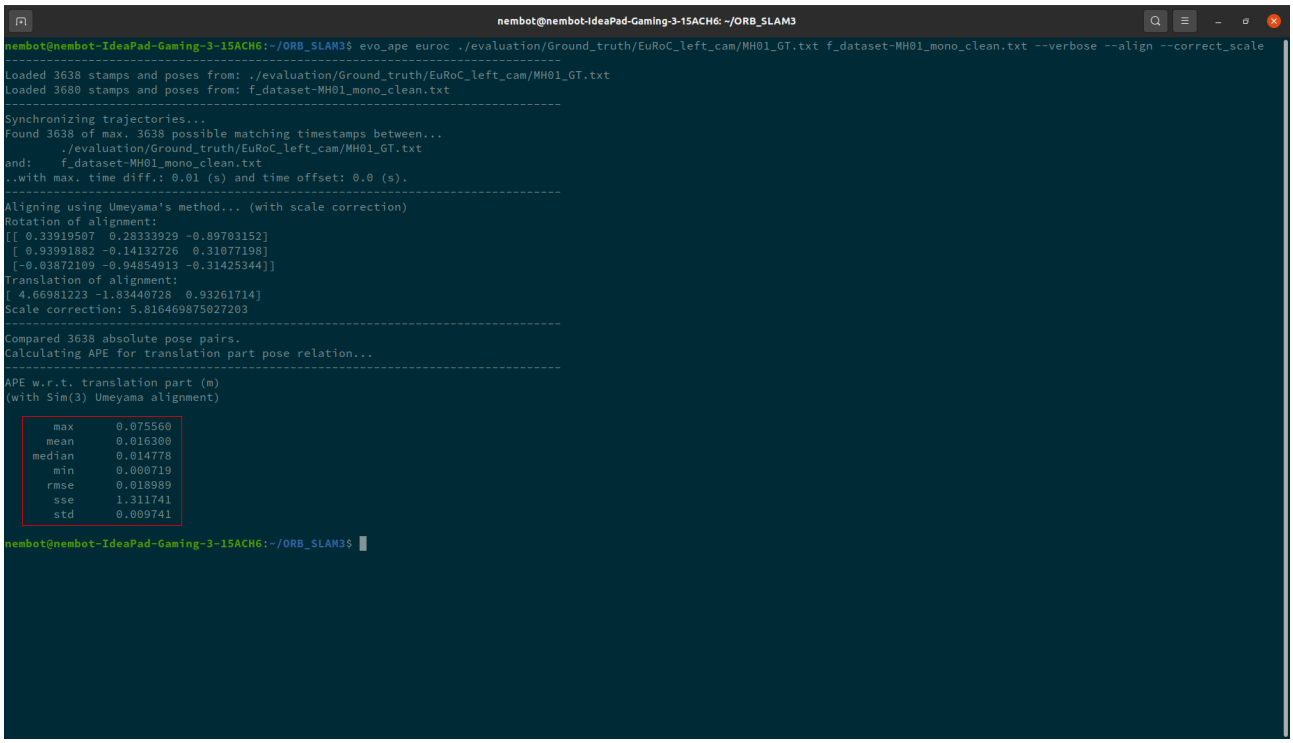

Figure 6.13: Benchmarking Results with MH01 Ground Truth from ORB-SLAM3's Repository

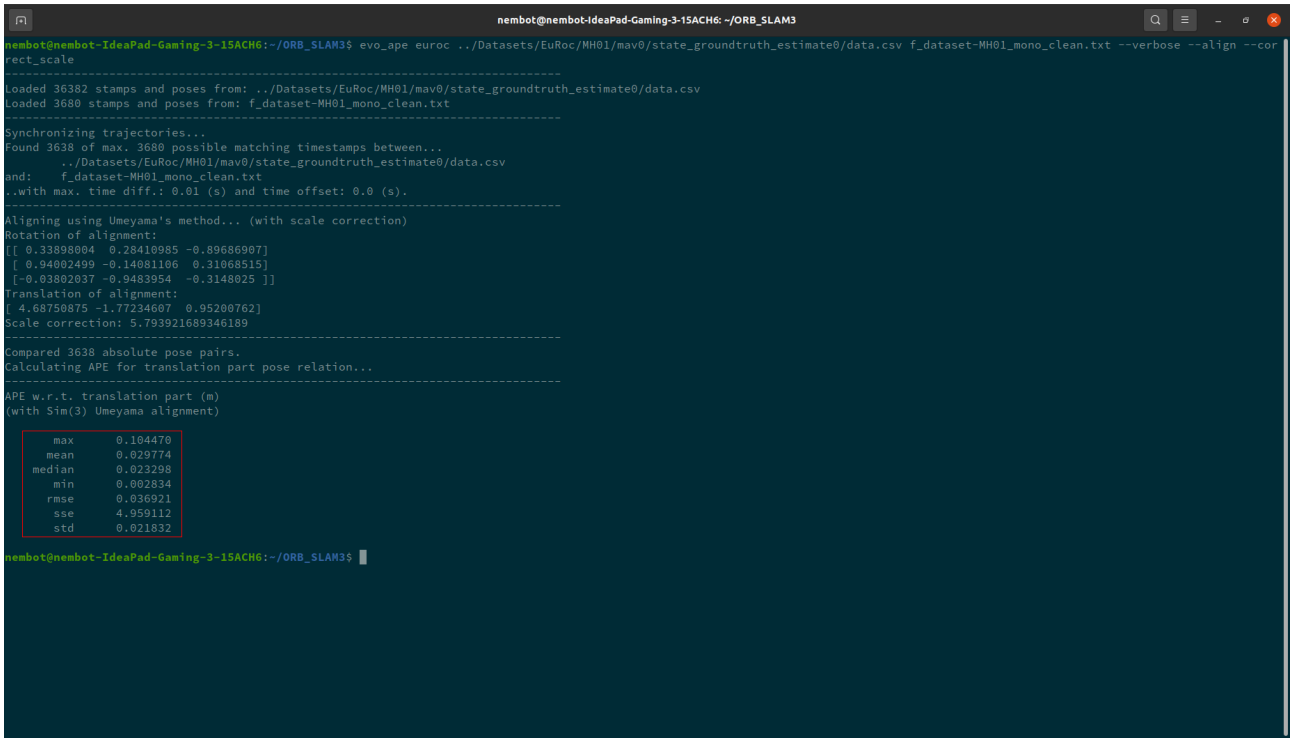

Figure 6.14: Benchmarking Results with Ground Truth from Euroc MH01 dataset

A comparison of the data in Table 6.5 with the results reported in the ORB-SLAM3 paper (Campos et al., 2021) revealed discrepancies. Further investigation indicated that ORB-SLAM3 can achieve the reported performance when utilizing the ground truth data provided within the official ORB-SLAM3 repository Figure 6.13. However, for consistency and transparency within this study, the ground truth values included with the publicly available benchmark datasets will be employed for evaluating both the proposed system and ORB-SLAM3 Figure 6.14. This approach ensures a fair and replicable comparison based on a common set of reference values.

Conclusion

The Appendix E has detailed the ongoing development of a system designed to address limitations in existing ORB-SLAM3 system. The section has focused on establishing a benchmarking using the ORB-SLAM3 system using the EuroC datasets.

The benchmarking process employed a diverse range of datasets and configurations to encompass various environmental conditions a SLAM system might encounter. The results from the ORB-SLAM3 benchmarking provide valuable insights into the performance of this baseline system under different scenarios. Notably, the findings highlight the impact of factors such as:

- **Sensor Modality**: The inclusion of inertial data generally led to lower APE values compared to solely visual data configurations. This aligns with previous research by Mourikis and Roumeliotis (2007).
- **Lighting Conditions**: The MH01 dataset, with its controlled lighting, yielded more consistent results across all configurations compared to datasets with varying lighting conditions.
- **Scene Complexity**: The V101 and V201 datasets, featuring limited features and fast camera motion, respectively, presented challenges for the ORB-SLAM3 system. This emphasizes the need for robust feature extraction, which the proposed system aims to address.

Moving forward, the focus will shift towards developing the ALIKED Intergrated system. The established benchmark will be utilized to evaluate the proposed system's performance and compare it against the ORB-SLAM3 baseline. This comparison will be based on the publicly available ground truth data from the EuroC datasets, ensuring a fair and replicable evaluation process. The goal is to illustrate the system's ability to elevate SLAM performance within a variety of environmental contexts.