# 16-833: Localization and Mapping, Fall 2024
# Final Report
# Stereo Visual Inertial Odometry using KLT feature Tracking

Ashwin Biju Nair, Atharva Sunder Ramdas, Preet Patel and Vinayak Patel

*Abstract*— **This project is an implementation of a Visual Inertial Odometry (VIO) system, specifically, the Stereo Multi-State Constraint Kalman Filter (S-MSCKF). The proposed system aims to address the limitations of MSCKF approach by creating a filter-based stereo VIO that uses the multistate Kalman Filter, enhancing accuracy and robustness in complex environments. Key aspects of the project include developing an efficient feature selection strategy, by implementing Kanade-Lucas-Tomasi (KLT) tracking. The original implementation by researchers from Kumar Lab, University of Pennsylvania, [2], is on C++ and ROS Kinetic. We aim to improve the existing feature management by improving outlier removal methods on a python implementation of this algorithm. The project repository is available on the following link: https://github.com/Ashwin-B-Nair/SLAM-Project-Fall-24.**

## I. INTRODUCTION

Visual Inertial Odometry (VIO) is a technique used in robotics to estimate the position, orientation, and velocity of a moving object relative to its starting point. This method combines visual data from cameras with inertial measurements from an Inertial Measurement Unit (IMU) to provide accurate and robust motion tracking. VIO operates on two main components:

- Visual Odometry: Uses camera images to estimate motion and track visual features across consecutive frames. The aim of this component is to calculate the relative motion between frames.
- Inertial Measurements: Provides high frequency motion estimates using gyro and accelerometer data from an IMU. It helps to compensate for blurred images caused due to rapid movements.

Data from both these sources share a symbiotic relationship as IMU helps bridge the gap between visual measurements, and visual data helps correct IMU drift. The combined data improves the accuracy and robustness of the motion estimation.

VIO can be employed by a variety of algorithms that are available these days. Widely, they all can be classified into filter-based and optimization-based approaches. Filter-based methods generally use Extended Kalman Filter (EKF) or other Bayesian Filters to process data sequentially. These methods are generally more efficient and have accuracy comparable to the optimization-based methods. In contrast, the optimization-based methods performs joint optimization over a batch of measurements like . These methods are often more accurate but computationally more expensive due to the iterative optimization process.

Dynamic environments pose a significant challenge to VIO systems, particularly in densely populated areas, by confusing the visual tracking algorithms, leading to inaccurate motion estimates. Another issue is variable lighting conditions as VIO heavily relies on visual features. Changes in illumination or reflective surfaces can all degrade the quality of visual data, making it harder for the system to accurately track motion. The accumulation of errors over time, known as drift, remains a persistent issue in VIO systems. While the integration of inertial data helps to reduce short-term drift, long-term accuracy can still be compromised, especially in the absence of loop closures.

Multi-State Constraint Kalman Filter (MSCKF) combines visual data from cameras with inertial measurements from an IMU to estimate the position, orientation, and velocity of a moving object. The core innovation of MSCKF is its ability to process feature measurements without including feature positions in the state vector, resulting in computational complexity that is only linear in the number of features ([4]). The algorithm maintains a sliding window of past camera poses in the state vector and uses them to create multi-state constraints when processing visual measurements. MSCKF has become a foundation for many subsequent VIO systems due to its ability to handle large numbers of visual features while maintaining real-time performance.

One such VIO system that builds on MSCKF is Stereo Multi-State Constraint Kalman Filter (S-MSCKF). A stereo configuration is preferred over monocular systems because it is robust to different environments and motion. The S-MSCKF system implemented in [2] is what we aim to recreate and build on in our project. This system contradicts the belief that stereo vision based estimation incurs a much higher compute cost than monocular approaches. S-MSCKF is able to achieve similar or even higher efficiency than state-of-art monocular solutions.

## II. BACKGROUND

Here we discuss some state-of-the-art open-source VIO systems:

### *A. Keyframe-Based Visual-Inertial SLAM Using Nonlinear Optimization*

This paper presents a tight approach for integrating/fusing visual and inertial measurements in a keyframe-based visual SLAM system [5] instead of opting for conventional filtering techniques. This approach exploits the sensing cues effectively and uses nonlinear estimation wherever possible to mitigate suboptimalities arising from linearization. The nonlinear optimization framework combines visual reprojection errors and IMU error terms in a single cost function. A keyframe-based approach is used, which maintains a sparse graph of camera poses and landmarks. To ensure real-time performance and reduce computational complexity, partial marginalization technique is used to bound the optimization window. Their method also incorporates IMU kinematics and bias models into the state estimation.

The innovation in this paper is that they are allowing the keyframes to be spaced arbitrarily in time instead of using time-successive poses. This system is also using inertial cues for outlier rejection in feature mapping to improve robustness.

However, the downside is that the model uses marginalization where linearization points are being fixed. This can lead to inconsistency if the linearization point is away from the true state. Marginalization can also lead to information loss, thus affecting long-term accuracy. This method also does not directly address loop closures which are essential for consistency in larger environments.

### *B. Robust Visual Inertial Odometry Using a Direct EKF-Based Approach*

This paper presents a VIO algorithm using EKF for tracking [6]. Specifically, the pixel intensity errors of image patches are used for tracking. It considers the robot at the center and estimates the 3D landmark relative to the current robot pose. New visual features can be easily integrated in this method as it has an undelayed feature initialization process. IMU measurements are used for state propagation and visual information is used for filter updates.

Key innovations of this algorithm include using QR factorization to reduce high-dimensional error terms and causing Kalman updates to be more tractable. This method also uses multi-level patch features closely coupled to the EKF. Having such a robocentric co-estimation of camera extrinsics and IMU biases is also a novelty of this research article.

This method, with its reliance on tracking visual features, can perform worse in low light and low texture environments. As mentioned by the authors themselves, this algorithm may perform worse in situations involving sudden camera movements.

### *C. VINS-Mono A Robust and Versatile MonocularVisual-Inertial State Estimator*

This paper presents VINS-Mono [7], a monocular visual-inertial state estimator designed to solve the SLAM problem with a single camera and IMU combination. Their implementation addresses challenges such as initialization, long-term drift and map reuse.

Feature extraction and tracking is done through the camera images with pre-integration with the IMU measurements between consecutive frames in this algorithm. Initial values of parameters like pose, velocity, gravity vector and scale is done by a vision based structure from motion (SfM) procedure followed by visual-inertial alignment. Drift is reduced by implementing a 4-DOF pose graph optimization. The system can save multiple local pose graphs which leads to effective map reuse.

Though, it should be noted that this system cannot be started from a stationary condition and needs to launch from an unknown moving state in order to get initial estimation of the aforementioned parameters. The system proposed here is also highly non-linear and can turn out to be more challenging for estimator initialization. 4-DOF pose graph may not be sufficient to reduce drift in some specific applications. Moreover, as this method relies on feature estimation and tracking, it may perform worse in low texture and low-light environments. The method shown also seems to give inconsistent estimations in a varied range of scenarios.

## III. METHODOLOGY

### *A. Overview of MSCKF VIO Algorithm*

The system state is defined as a combination of camera and IMU states. The IMU states are its position, velocity and orientation, relative to the world frame along with bias terms. The camera state comprises its position and orientation relative to the world frame at different time steps, and the number of camera states in the state vector is restricted by a threshold. The IMU runs at a higher frequency than the camera, therefore these states are updated at a higher rate, using an IMU motion model. Each time a new camera frame is obtained, the image is processed for detecting features, and the camera's pose at that instant computed from the IMU pose using the transformation between the camera frame and IMU frame is added to the system state. On reaching the threshold number of camera states in the state vector, or when a feature is lost from the camera's view, a measurement update is carried out where state covariances are corrected from IMU induced errors with the help of a residue computed from the feature positions in each camera frame. This process repeats, with the IMU improving estimates of camera pose as compared to those computed by estimating poses from feature positions on multiple image frames, and the camera correcting IMU measurement drifts.

### *B. Overview of Image Processing Algorithms*

The image processing algorithm is responsible for detecting features from stereo inputs and efficiently maintaining a set of features that are tracked across frames. These features are utilized to estimate the pose of the camera and IMU at various time steps. The image processing component is divided into four main subcomponents: Feature Detection, Feature Tracking, Outlier Rejection, and Feature Management, each contributing to robust and accurate motion estimation.

The FAST (Features from Accelerated Segment Test) algorithm is used for corner detection in the input stereo images. It identifies key points by examining the intensity of pixels around a candidate pixel and classifies it as a corner if the intensity gradient crosses a defined threshold. The algorithm outputs the coordinates of detected features along with response values that measure the quality of these features. FAST is chosen for its robustness and computational efficiency, making it suitable for real-time applications.

Feature tracking is performed using the Lucas-Kanade (LK) optical flow method to track features across consecutive frames by estimating their motion. The LK method assumes that pixel intensities within a small image patch remain constant over time and that inter-frame motion is small. It leverages IMU data to predict feature motion and uses image pyramids to handle larger displacements, starting at lower resolutions and refining at higher ones. This approach updates feature positions in each frame, ensuring robust and accurate motion estimation.

Outlier rejection ensures that only reliable feature correspondences are used by removing incorrect matches. This step improves the robustness of the algorithm by filtering out noisy or inconsistent data points, ensuring that the remaining features adhere to the geometric constraints required for accurate motion estimation. This uses RANSAC algorithm which is discussed in the next section in detail.

Feature management ensures that features are uniformly distributed across the image and maintains an optimal number of features for robust tracking. The image is divided into grids, and features are grouped based on their spatial location. Excess features in overpopulated grids are pruned using pruning methods while new features are detected and added to underpopulated grids. This balanced approach ensures a well-distributed and high-quality set of features for consistent tracking across frames.

### C. *RANSAC Outlier Removal*

RANSAC, which stands for Random Sample Consensus, is an iterative algorithm used to estimate parameters of a mathematical model from a data set containing outliers. The key idea behind RANSAC is to iteratively select random subsets of the data, fit a model to this subset, and evaluate how well the model fits the entire dataset by counting inliers- data points that fit the model within a certain threshold. The reason why RANSAC is used is two-fold. Firstly, it excels in scenarios where datasets include significant noise or outliers. RANSAC is robust in its approach by providing an accurate regression model in conditions where traditional least-squares fitting methods fail. Secondly, by focusing only on subsets of data and iteratively refining the model, RANSAC avoids exhaustive computation, thus improving efficiency.

The `two_point_ransac` function in `image.py` applies RANSAC to identify inliers among feature correspondences between two camera frames. How `two_point_ransac` works:

- Normalizes pixel units using the camera intrinsics and computes the number of iterations required based on the success probability and the inlier ratio.
- Undistort input points using camera parameters and compensate for rotation between frames by applying `R_p_c`.
- Computes the differences between the corresponding points (`pts1` and `pts2`) and removes pairs with large differences or degenerate motion (pure rotation without translation).
- Randomly selects two pairs of points to construct a translation model and evaluates all point pairs against this model to identify inliers.
- Refits the model using all identified inliers and updates the best model if it improves upon previous iterations.
- Finally, returns an array marking each point as an inlier (1) or outlier (0).

## IV. DATASET

### A. *EuRoC MAV*

The EuRoC MAV dataset [1] was collected by the Institute of Robotics and Mechatronics at the German Aerospace Center and is widely used for benchmarking algorithms in Visual-Inertial Odometry (VIO) and Simultaneous Localization and Mapping (SLAM). The dataset was recorded using a Micro Aerial Vehicle (MAV) equipped with synchronized stereo cameras and an IMU. The data was gathered in both indoor and industrial environments, making it suitable for evaluating algorithms in diverse settings.

The dataset was collected in two types of environments: the Machine Hall (Fig 1), a large industrial space, and the Vicon Room (Fig 2), a smaller, controlled environment equipped with a motion capture system. The MAV used stereo cameras (Aptina MT9V034 global shutter cameras) capturing images at 20 Hz, along with an ADIS16448 MEMS IMU providing inertial measurements at 200 Hz. Ground-truth data is provided using a Leica MS50 laser tracker in the Machine Hall and a Vicon motion capture system in the Vicon Room, ensuring accurate pose estimation for benchmarking.

Data collected is categorized into 11 sequences that vary in difficulty (easy, medium, difficult), with each sequence providing stereo images, IMU data, and ground-truth poses.The dataset also includes calibration files to ensure accurate alignment between the camera and IMU data.

Fig. 1: ETH Machine hall

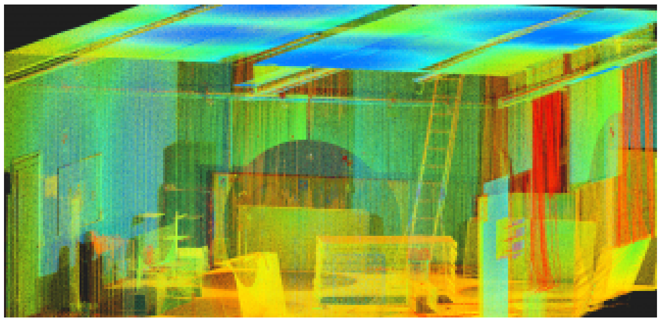


Fig. 2: EuRoC: Ground truth 3D scan of Vicon

The dataset has a few known issues that could impact the performance of algorithms. One key challenge is the automatic exposure control of the visual-inertial sensor, which operates independently for each stereo camera. This causes different shutter times between the cameras, resulting in variations in image brightness. While this complicates stereo matching and feature tracking, the cameras' mid-exposure times remain synchronized, so temporal alignment is not affected. Additionally, some sequences involve high-speed motions, which can reduce the accuracy of the laser tracking system used for ground-truth measurements. The manufacturer's reported accuracy may be overly optimistic during these dynamic movements, complicating comparisons for precise visual odometry methods. However, this effect is minimal during less dynamic sections, particularly at the start and end of each sequence. Another issue arises from the synchronization between sensor data and motion ground truth, as they were recorded on separate systems without device timestamps for the Vicon system. To mitigate this, a temporal offset and clock drift are estimated, though raw data is available for those interested in exploring alternative synchronization methods.

## V. EXPERIMENTS/ IMPLEMENTATION

### A. Effect of outlier Removal Methods on Pose Estimation

Using the `two_point_ransac` function, a test was conducted to check the effect of removing outliers on pose estimations by comparing it with the existing VIO model and ground-truth trajectory. The aim of this experiment is to understand whether the elimination of outliers has a positive effect of trajectory and what combination of hyperparameters provides the best results. The metric used to evaluate performance was the Root Mean Square Error (RMSE) between the estimated coordinates (x,y & z) and the ground truth. The hyperparameters and results can be found in section VII.A. Hyperparameters used and their meaning:

- `inlier_error`: Defines the maximum allowable error for a point to be considered an inlier.
- `num_features`: Determines the maximum number of features in each grid of a given frame.

### B. Pruning Features

The current implementation sorts features in descending order based on their lifetime, which is a basic approach. To enhance feature selection, we incorporated a linear combination of lifetime and response values to compute an importance characteristic for each feature within a grid. Response values, which measure the quality of features, are included to improve the robustness of the selection process. This approach aims to prioritize features that are both long-lived and of high quality, leading to better overall performance.

### C. Visualizing the Outputs

Visualizations are carried out by the `viewer.py` file, which is executed at each timestep specified in the `vio.py` file, updating the plots and visualizations with new predictions and new stereo feature matches accordingly. Made use of the `matplotlib` package to plot the graph of ground truth and predictions with time, individually in the x, y and z directions. Visualization for the Stereo Camera feed showing real-time feature matching was also incorporated. A figure showing the entire 3D ground truth trajectory along with real-time updating predicted trajectory along with the dataset video feed was also created using `matplotlib` and `opencv-python` packages.

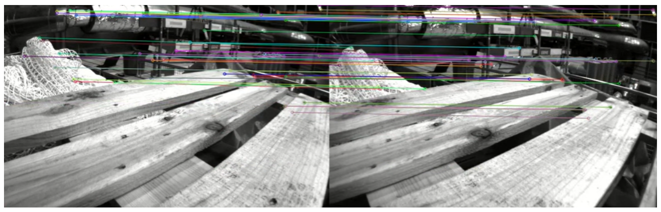


Fig. 3: Stereo Feature Matching

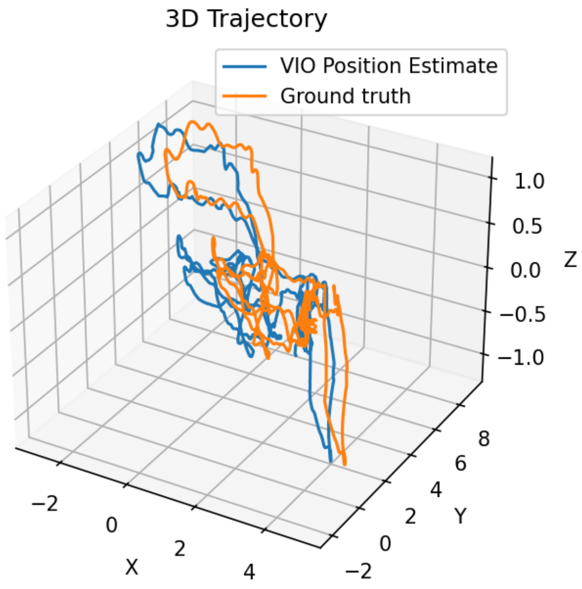
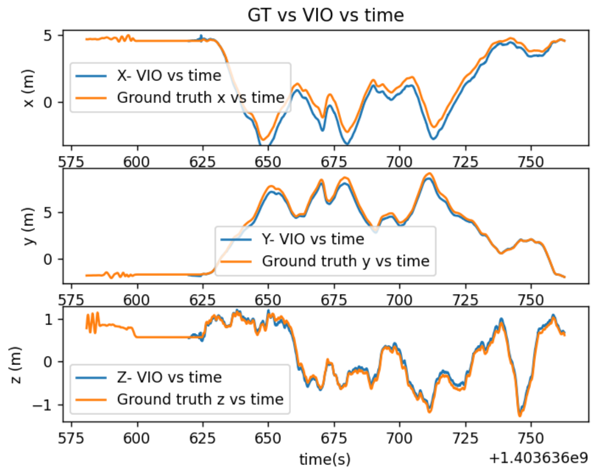


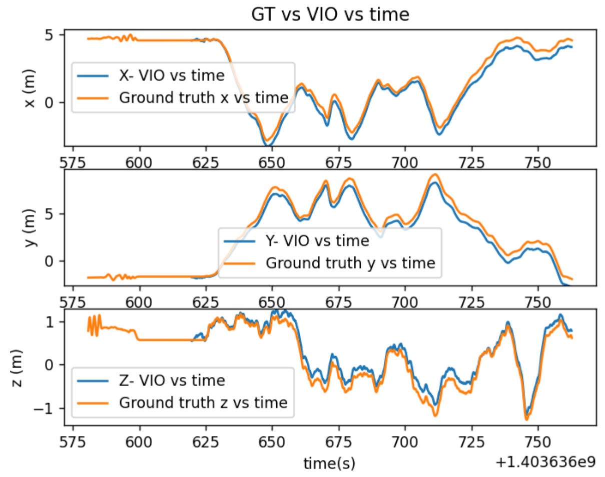
Fig. 4: 3D trajectory in MH01

## VI. RESULTS & DISCUSSION

The following section provides a comprehensive description of results obtained from our study and their inferences.

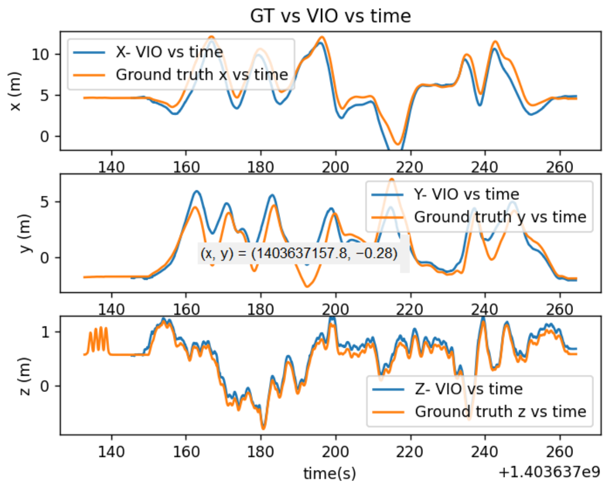


(a) Without RANSAC

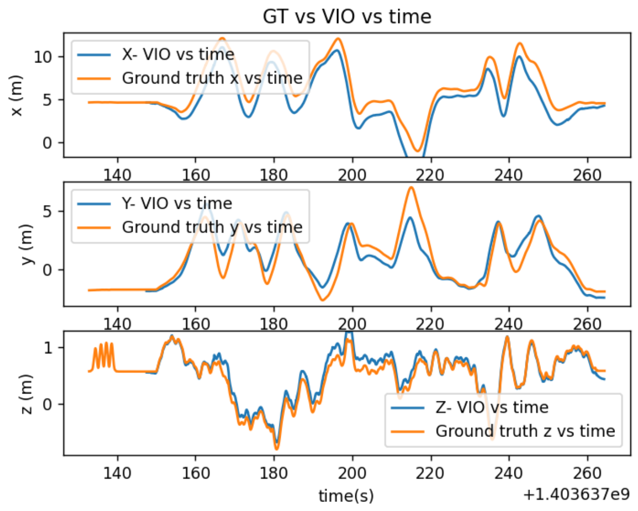


(b) With RANSAC

Fig. 5: Comparison of trajectories using RANSAC and without RANSAC on MH-01-easy dataset



(a) Without RANSAC



(b) With RANSAC

Fig. 6: Comparison of trajectories using RANSAC and without RANSAC on MH-03-medium dataset

### *A. RANSAC Outlier Removal*

TABLE I: RMSE Values for MH-01-easy

| | $RMSE_x$ | $RMSE_y$ | $RMSE_z$ | Mean Distance |
|---|---|---|---|---|
| Ground Truth | 0.468 | 0.5 | 0.275 | 0.7 |
| Trial 1 | 0.3 | 0.72 | 0.44 | 0.88 |
| Trial 2 | 0.27 | 0.44 | 0.36 | 0.58 |
| Trial 3 | 0.52 | 0.43 | 0.24 | 0.66 |

TABLE II: Hyperparameter values for all Trials

| | Inlier Error | No. of Features | Grid configuration | |
|---|---|---|---|---|
| | | | Grid row | Grid col |
| Trial 1 | 3 | 5 | 4 | 5 |
| Trial 2 | 5 | 5 | 4 | 5 |
| Trial 3 | 5 | 7 | 4 | 5 |

The experiments conducted on the EuROC MH-01 dataset yielded insightful results regarding the performance of RANSAC in trajectory estimation. The analysis focused on comparing estimated trajectories with ground truth values, evaluating RMSE across multiple trials with varying RANSAC parameters.

Contrary to expectations, the implementation of RANSAC in Trial 1 resulted in a slight increase in RMSE values compared to the scenario without outlier removal.(Fig 6a & Fig 6b) This unexpected outcome is attributed to RANSAC maintaining a lower average number of features, suggesting that accurate pose estimation may require a higher feature count. To optimize the balance between essential feature retention and computational efficiency, multiple RANSAC hyperparameter configurations were tested. The most promising results are presented in Table II.

Trial 2 employed a higher RANSAC threshold for inlier feature filtration while maintaining the same number of features per grid. This approach yielded improved results-Decreased RMSE values for x and y coordinates, Reduced average Euclidean distance and a computational efficiency comparable to the ground truth run time. Notably, Trial 1 exhibited a 20% reduction in run time due to its lower feature count, albeit at the cost of accuracy.

Trial 3 maintained the RANSAC threshold from Trial 2 but increased the number of features. However, this approach proved computationally costly with increased run time and no significant improvement in RMSE values.

Based on these findings, Trial 2 emerges as the optimal configuration, offering the best balance between accuracy in trajectory estimation and computational efficiency. It reasonably maintains ground-truth trajectory without incurring additional computational overhead compared to the original run.

TABLE III: RMSE Values for MH-03-medium

| | $RMSE_x$ | $RMSE_y$ | $RMSE_z$ | Mean Distance |
|---|---|---|---|---|
| Ground Truth | 1.24 | 1.16 | 0.06 | 1.47 |
| Trial 1 | 1.39 | 1.05 | 0.11 | 1.55 |
| Trial 2 | 2.31 | 0.92 | 0.12 | 2.39 |
| Trial 3 | 1.61 | 0.95 | 0.11 | 1.73 |

The tests carried out on the EuROC MH-03 medium dataset yielded RMSE values that were measured using the same methodology as in previous experiments. The results presented in Table III reveal interesting patterns. In the first trial, a slight increase in RMSE values was observed. This trend aligns with the findings from the MH-01 dataset, where the Random Sample Consensus algorithm maintained a reduced number of features. While this approach resulted in faster computation, it led to a less accurate pose estimate.

Contrary to the MH-01 dataset results, the second trial for MH-03 produced more erroneous pose estimates. This can be attributed to the elevated threshold employed in RANSAC. The higher threshold diminished RANSAC's effectiveness in outlier filtration, consequently skewing the results. A comparison between Trial 2 of MH-01 and MH-03 datasets reveals contradictory outcomes. For MH-01, the use of essential features led to accurate trajectory predictions. However, in the case of MH-03, where the drone exhibited more rapid movement, the higher RANSAC threshold failed to provide the algorithm with essential information for accurate predictions. This suggests a greater reliance on Inertial Measurement Unit (IMU) data for pose estimation in the MH-03 scenario.

Trial 3 successfully addressed the shortcomings observed in the previous trials. This trial processed a larger number of features while maintaining the RANSAC threshold from Trial 2. This approach allowed for more effective selection of essential features without significantly increasing computational overhead. Notably, the processing time remained relatively constant, varying within a 10-second margin.

In conclusion, this study underscores the importance of careful hyperparameter tuning in RANSAC implementation for trajectory estimation, highlighting the delicate balance between feature quantity, RANSAC threshold settings, and computational resources.

### B. Pruning Methods

Several trials were conducted by varying different parameters and testing various combinations of lifetime and response values to optimize the mean distance results. In

TABLE IV: RMSE Values for MH-01-easy

| | $RMSE_x$ | $RMSE_y$ | $RMSE_z$ | Mean Distance |
|---|---|---|---|---|
| Ground Truth | 0.48 | 0.50 | 0.27 | 0.70 |
| Trial 1 | 0.37 | 0.42 | 0.37 | 0.64 |
| Trial 2 | 0.37 | 0.42 | 0.37 | 0.64 |
| Trial 3 | 0.56 | 0.69 | 0.43 | 0.96 |
| Trial 4 | 0.56 | 0.69 | 0.43 | 0.96 |
| Trial 5 | 0.51 | 0.53 | 0.26 | 0.71 |
| Trial 6 | 0.51 | 0.53 | 0.26 | 0.71 |

TABLE V: Parameter values for all Trials)

| | $Condition$ | $RANSAC$ | $Reversed$ |
|---|---|---|---|
| Ground Truth | $1 * l$ | False | True |
| Trial 1 | $1 * l + 0.005 * r$ | False | False |
| Trial 2 | $1 * l + 0.002 * r$ | False | False |
| Trial 3 | $1 * l + 0.002 * r$ | True | False |
| Trial 4 | $1 * l + 0.001 * r$ | True | False |
| Trial 5 | $1 * l + 1/r$ | True | False |
| Trial 6 | $1/r$ | True | False |

Trial 1, the response values were scaled down by a factor of 200 to reduce the impact of scale differences between response and lifetime values. To further refine the results, the response values were later scaled by a factor of 500. However, it was observed that the results remained exactly the same despite the change in scaling factor. Since these results were decent, we tested them alongside a RANSAC implementation. Unfortunately, this approach did not yield promising outcomes. In Trial 5, we experimented with non-linear combinations of parameters, which produced better results compared to linear combinations but still fell short of outperforming RANSAC alone. Building on the observation that scaling factors did not affect the results, we hypothesized that the influence of lifetime values was negligible. To test this assumption, in Trial 6, we used only the response values and observed identical results once again, confirming our hypothesis. This behavior was unexpected, as we initially predicted that combining lifetime and response values and sorting them in descending order would improve the results. However, this was not observed during our tests. One key reason might be the negligible contribution of lifetime values, as scaling experiments showed no impact on outcomes. During other tests it is also observed that lifetime values were 1 for most of the features.

## VII. Challenges

### A. Pangolin

Initially, we planned to use a Python binding for a popular C++ and OpenGL based SLAM visualization package called Pangolin in order to visualize our results. The python binding/wrapping of this package is known as `pypangolin`. This binding however is not a part of the official python packages which can be installed from the pip package installer. Instead pypangolin needs to be installed through cmake. However, the problem was that the pangolin package for C++ was maintained properly, the python binding was not. There were a few syntax errors in the cmake installation file for the python binding as that file was made on a much older version of cmake and a few methods had depricated.

We changed the cmake file as per the latest version's syntax and then tried to run it. Through that I was able to get `pypangolin` to install. However, due to the package being outdated, we were getting a lot of compatibility issues wherein `pyoangolin` did not work with newer versions of many other packages like `opencv-python` and `open3d`. We tried to downgrade these packages but the more we attempted to fix the compatibility issues, more and more newer conflicts came up to the point where we decided to completely scrap the idea of using `pypangolin`.

After consideration on how to visualize the results, we decided to write the code for the `viewer.py` file using `matplotlib` and `opencv-python` packages. To get an idea about which parameters to plot, we referred to the examples shown on `pypangolin`'s github page and tried to recreate similar looking visualizations using the aforementioned two packages.

This was a significant learning experience for us as none of us were familiar with cmake and resolving package conflicts and dependencies. We had to learn these things on the fly while working on other parts of the project. We tried to install `pypangolin` on an Arch Linux, an Ubuntu and a Windows system but we met with the same issues on all OSes implying the the issue was not OS specific but in general with how ill-maintained the package is.

### B. *Navigating Code base - multi threading & queue*

The code base that has been referred to is not structured around ROS; data publishing is fundamentally based on python's mulithreading feature. Here, multiple functions run simultaneously and data is added into and accessed from queues. Unlike a general ROS codebase with features such as servies, this code provided less flexibility for controlling the update rate of each queue.

### C. *RMSE calculations*

The IMU queue, the image queue and the ground truth queue run as separate threads, and where each queue is updated from the recorded measurement values. Despite these queues being updated at similar rates, the image data was further processed to detect features, and the IMU queue was processed to update the system state. Since these steps consumed significant time, the IMU queue and image queue increased in length slower than the ground truth queue (whose data was not processed.) Thus, calculated RMS values of position components were initially inaccurate due to a mismatch of corresponding timestamps between the ground truth queue and the VIO measurement queue.

This challenge was overcome by matching timestamps between the ground truth and VIO output queue before publishing data to the plot function (`viewer.py`).

## References

[1] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik and R. Siegwart, The EuRoC micro aerial vehicle datasets, International Journal of Robotic Research, DOI: 10.1177/0278364915620033, early 2016.

[2] Sun, Ke, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. "Robust stereo visual inertial odometry for fast autonomous flight." IEEE Robotics and Automation Letters 3, no. 2 (2018): 965-972.

[3] Jaekel, Joshua, Joshua G. Mangelson, Sebastian Scherer, and Michael Kaess. "A robust multi-stereo visual-inertial odometry pipeline." In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4623-4630. IEEE, 2020.

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

[5] Leutenegger, Stefan, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. "Keyframe-based visual–inertial odometry using nonlinear optimization." The International Journal of Robotics Research 34, no. 3 (2015): 314-334.

[6] Bloesch, Michael, Sammy Omari, Marco Hutter, and Roland Siegwart. "Robust visual inertial odometry using a direct EKF-based approach." In 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp. 298-304. IEEE, 2015.

[7] Qin, Tong, Peiliang Li, and Shaojie Shen. "Vins-mono: A robust and versatile monocular visual-inertial state estimator." IEEE transactions on robotics 34, no. 4 (2018): 1004-1020.