Kimera - an open source library for metric-semantic Visual-Inertial Simultaneous Localization And Mapping*

Ken Schlesiger
Informatics XVII
University of Würzburg
Am Hubland, D-97074 Würzburg, Germany
ken.schlesiger@stud-mail.uni-wuerzburg.de

*This paper has been written as part of the seminar 3D mapping and geometry processing for underwater and aerospace applications in Winter 2023/24

Abstract

Kimera is a modular and accurate library for real-time Visual inertial odometry, pose graph optimisation, 3D-Mesh construction and semantic segmentation. It is able to be run in a performance constraint environment without the use of a GPU for processing.

Kimera is meant to provide a flexible and extensible basis for future research in the fields of semantic segmentation, Visual inertial odometry and 3D mesh generation.

1 Introduction

In recent years the fields of computer vision, robotics and autonomous systems have undergone remarkable advancements. The ability to quickly estimate the 3D geometry of a scene and understand its contents enable such complicated tasks as the interaction of robots with their surroundings or self driving cars. These processes require real-time and accurate spatial perception for safe navigation and effective interaction with the environment. Kimera [6] is an open-source Library designed for metric-semantic Visual-Inertial Simultaneous Localization And Mapping (SLAM), which is meant as a basis for further research and development of such systems. The Kimera Library offers state of the art approaches to Visual inertial Odometry, pose graph optimisation, mesh reconstruction, and 3D semantic segmentation. The Library is designed to run in real time on only a CPU instead of a GPU, as is the used by most other 3D mapping libraries. Kimera achieves this by fusing the fields of 3D-geometric reconstruction of an area with semantic segmentation, which have traditionally been seen as separate areas of research. Kimera goes beyond existing visual and visual-inertial SLAM libraries by not only providing fast and accurate state estimates, but also allowing for real time mesh reconstruction and semantic labeling. The Library is comprised of four main components, a visual inertial odometry module, a pose graph optimisation module, a lightweight 3D mesh generation module and 3D metric-semantic reconstruction module.
Unlike many existing libraries, which use RGB-D cameras or LiDAR, Kimera focuses on visual and inertial sensing using a pair of stereo cameras, to broaden the range of environments that can be reliably mapped.
This write up is split into 2 main parts.

- An overview over the state of the art at the time Kimera was released and an overview of what has been done since in 2
- An overview over Kimera itself and its performance in 3 This overview breaks down Kimera into its four main components. Before the explanation of each part the preliminaries will be explained to give novice readers understanding of topics vital to the understanding of Kimera.

For further details, the complete documentation and supplementary material can be found on the Kimera GitHub repository: `https://github.com/MIT-SPARK/Kimera`
A demonstration video is available on YouTube: `https://www.youtube.com/watch?v=-5XxXRABXJ`

2 Background

This section is mainly split into parts: *(i)* A historical overview over simmilar Libraries before Kimera, *(ii)* A short overview over the work Kimera is based on, *(iii)* The additions onto kimera since its release and *(iv)* a general overview over Simillar libraries since the release of Kimera.

(i) A historical overview over simmilar Libraries

There are several libraries that enable a subset of the functions that Kimera can achive. todo

3 Kimera

3.1 Structure

Kimera is build from four independent modules:

- **Kimera-VIO:** A GTSAM [1] based visual inertial odometry Module. Kimera-VIO allows for IMU-rate state estimation.
- **Kimera-RPGO** A pose graph optimizer, with modern outlier rejection.
- **Kimera-Mesher** A module that efficiently generates both per frame and multi frame regularized 3D meshes.
- **Kimera-Semantics:** A module that builds a more accurate and volumetric 3D mesh than Kimera-Mesher and semantically annotates the mesh using 3D semantic segmentation.

Each module is able to perform independent from the other modules. This allows the user to replace or modify the Modules as they see fit swell as run each module on their own if only a subset of the functionality is required. Due to Kimeras Lightweight design, being able to run on only a CPU, each module is not only able to run offline using a previously generated dataset, but also able to run online on a real time system using the Robot operating system (ROS).

3.2 Kimera-VIO

3.2.1 to 3.2.3 are meant to explain important topics to Kimera-VIO. 3.2.6 explains the inner workings of Kimera-VIO.

3.2.1 On-manifold preintegration

In theory the IMU measurements can be integrated between time t_i and t_j. Problems do however arise in the context of optimisation based state estimation. Here the state of the system at time t_i is unknown and changes as more data points become available during successive steps of optimisation. T. Lupton and S. Sukkarieh [5] propose the use of pseudo measurements independent from the instal conditions which only describe the motion between measurements. Forster *et al.* [2] improve on this by providing a more mathematically rigorous approach that avoids the problem of singularities caused by the use of euler-angles. Their approach also generates a factor graph allowing for the use of incremental smoothing algorithms. They also use a structureless model which reduces the computational load while retaining accuracy. todo rewrite section

3.2.2 Shi-Tomashi Corners

Corners denote areas where a slight shift in location leads to a large change in pixel value. The Shi-Tomashi [8] algorithm is a modification of the Harris corner detection algorithm. It is comprised by three steps: *(i)* identification of which areas lead to large shifts in pixel value, *(ii)* calculate the *"R"* score and *(iii)* select the important corners.

***(i)* identification of areas of interest**

This is done with the following formula[1]:

$$E(u,v) = \sum_x \sum_y w(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{1}$$

x and y denote the position of a pixel, while u and v denote the amount of displacement of a pixel. w(x,y) is a function dependent on the shape of the window that is being calculated. The functions $I(x+u,y+v)$ and $I(x,y)$ denote the Intensity of a pixel in an Image. For example in a greyscale image this would be the brightness of a pixel. Therefore the term $I(x+u,y+v) - I(x,y)$ describes the relative shift of the intensity between two pixels. For a corner this value would be very large.

***(ii)* calculating the *"R"* score**

This is done with the following simple equation:

$$R = min(\lambda_1, \lambda_2) \tag{2}$$

λ_1 and λ_2 are the Eigenvalues of the Taylor-series expansion of $E(u,v)$

***(iii)* corner selection**

The selection of satiable corners is done by classifying all R values above a certain threshold as corners.

3.2.3 Lucas-Kanade tracker

The Lucas-Kanade tracker is a algorithm that tracks the positions of points in a video, that are moving due to camera movement or movement in the foreground. The Lucas-Kanade tracker works on the assumption that the brightness between two frames in a Video-stream is relatively constant. Mathematically this assumption can be written as:

$$I_x u + I_y w + I_t = 0 \tag{3}$$

Where the 2D positional gradients are expressed by $I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}$ and the temporal radian by $I_t = \frac{\partial I}{\partial t}$. u and v denote the movement in the x and y directions respectively.
Equation 3 cannot be solved as it is under constraint. Therefore the movement is observed over multiple points p_0 - p_n. This can be mathematically expressed as:

$$\begin{pmatrix} I_x(p_0) & I_y(p_0) \\ ... & \\ I_x(p_n) & I_y(p_n) \end{pmatrix} \begin{pmatrix} u & v \end{pmatrix} = \begin{pmatrix} I_t(p_0) \\ ... \\ I_t(p_n) \end{pmatrix} \tag{4}$$

Noise in real data will render equation 4 unsolvable as the system is overconstrained. Therefore instead of solving for u and v analytically the equivalent least squares problem needs to be solved to determine the camera movement.

[1]In actuality the Taylor-series expansion of E(u,v) is used as its performance requirements are lower.

3.2.4 Smoothing based state estimation

There exist two main smoothing approaches in the field of VIO, Fixed-Lag and full smoothing. In Full smoothing all recorded optimizes all state-measurements, while Fixed-Lag smoothing marginalizes out all states and only considers those in a certain time frame. The general Approach is otherwise similar: It begins with estimating the state in the relevant Time frame. The optimisation is done by comparing the measured IMU and visual data with a state prediction by using a cost function. The goal of the optimisation is to minimize this cost function at all points in time to get a trajectory that best fits reality.

3.2.5 Random sample consensus

RANSAC is a method that randomly selects n points and fits a function through them. The distances of each point to the straight are then determined. A score is then calculated by counting the number of points which support this model. This process is repeated several times or until all possible combinations of n points have been considered. The function with the largest highest score is then selected as the best fit.

3.2.6 Overview

The Kimera visual inertial odometry (VIO) module outputs state estimates of the robot. As input it requires both stereo camera frames and IMU data. While approaches based on filtering (e.g. Observationally constraint EKF, iterative EKF) are used in Visual Inertial Odometry they tend to degrade in performance over time as they render themself inconsistent [2]. Therefore Kimera employs Fixed-Lag smoothing (see sec. 3.2.4) based state estimation in form of a modified version of the On-Manifold Preintegration (see sec. 3.2.1) algorithm presented in [2] which computes a maximum a posteriori estimate in real time. This paper improves on the runtime constraints of usual VIO approaches by preintigrating the inertial measurements between keyframes into a single motion. This means that certain quantities will be computed between keyframes reducing the amount of variables that need to be optimized. Kimera expands on this to use both monocular and stereo cameras instead of only monocular cameras and also to allow both full lag and fixed lag smoothing , though fixed lag smoothing is used preferentially to limit the time the algorithm takes. Fixed lag smoothing denotes the marginalisation of measurements outside of a certain time window, while full lag smoothing estimates the state over all measurements.
Kimera-VIO is split into two parts: *(i)* The front-end and *(ii)* the back-end. The front end is responsible for feature tracking, preintigration of measurements and state estimation, while the back-end is responsible for improving the state estimation.

(i) The front-end

The Kimera-VIO front-end is split between the IMU based front-end and the vision based front-end. The IMU based front-end is processing IMU-data by performing the modified on-manifold preintegration to generate the relative measurements between two keyframes. The vision based front-end first detects Shi-Tomashi (see 3.2.2) corners. These corners are then tracked using the Lucas-Kannade Tracker [4] (see 3.2.3). Matches between the detected features in the left and right stereo images are then detected.
Kimera performs this geometric verification on both monocular vision using 5-point RANSAC (see sec. 3.2.5) and stereo vision using 3-point RANSAC. Kimera also offers the ability to verify the IMU data using mono and stereo verification using 2 and 1-point RANSAC respectively.

(ii) **The back-end**

The VIO back-end uses the preintegrated IMU and a structureless [2] visual model similar to [2] and adds them to a fixed lag smoother, in this case a factor graph. iSAM2 implemented in the GTSAM library is then used to solve the factor graph. iSAM2 is a incremental soothing technique which uses factor graphs to reduce the number of variables full smoothing approaches inherently have to compute. After each iSAM2 computation produces the newly refined state estimate the 3D-positions of features tracked by the VIO front-end are estimated.
This is done by the use of a direct linear transformation (DLT). DLT is a technique that leverages the geometry of the stereo camera to calculate the 3D position of the tracked features. These features are then removed from the factor graph as they are not required for state estimation.
Degenerate points and outliers are then removed from the data to ensure better performance. The last step of the VIO back-end algorithm is the marginalisation of all states that fall out of the VIO time horizon as these points will no longer be smoothed over.

3.3 Kimera-Mesher

Sec.3.3.1 to ? are meant to explain important topics to Kimera-VIO. Sec.3.3.2 explains the inner workings of Kimera-VIO.

3.3.1 Delaunay Triangulation

This is a method for triangulating a number of points. This means that a mesh gets generated that connects the points. This is done by drawing ovals of the points. These ovals are chosen in such a manner that every circle has exactly three points[3] within its shape. These three points are then connected.

3.3.2 Overview

The Kimera-Mesher produces two kinds of 3D mesh: *(i)* a per-frame mesh and *(ii)* a multi-frame mesh. The per-frame mesh is generated every frame from the stereo camera input, while the multi-frame mesh is generated from the per-frame mesh. Kimera-Mesher also has optional 2D fast local metric-semantic reconstruction to texture the mesh produced by the per-frame mesher.

***(i)* The per-frame mesh** The points that Kimera-VIO front-end produces in the current key frame are triangulated (see 3.3.1). This triangulation is then projected onto the results of the Kimera-VIO back-end. This results in a 3D mesh.

***(ii)* The multi-frame mesh** The multi-frame mesh is constructed from the output of the per-frame mesh. To do this only the per-frame meshes are considered which have fallen out of view. The multi-frame mesher determines which edges and vertices aren't jet contained in the multi frame mesh and adds them to it accordingly. The positions of all vertices are then updated with the position estimates produced by the Kimera-VIO back-end. Next old features vertices that have fallen behind the VIO time horizon get removed. This leads to objects which have moved being removed from the data. (todo mayby planar surfaces?)

[2] In this context structureless means that the Positions of landmarks (tracked features) are ignored by the fixed lag smoother

[3] This only applies in 2D in higher dimensional spaces this number can be bigger

3.4 Kimera-RPGO

3.4.1 Incremental Consistent Measurement Set Maximization

Incremental Consistent Measurement Set Maximization is a method of detecting loop closures in multi robot SLAM applications.
It takes a set of putative loop closures as input and tries to filter out all inconsistent loop closures. This is done by finding pairs of measurements that are with each other i.e. finding multiple measurements that seem probable. A maximum subset of these consistent measurements is then selected as the proper trajectory. This is efficiently solved by converting the problem into a Maximum Clique problem over a consistency graph. A Maximum Clique problem looks for the largest subset of vertices where all vertices have edges between them. A consistency graph is defined as a graph where each vertex represents a measurement and each edge denotes a consistency. In this context consistency refers to how similar two measurements are.

3.4.2 Overview

Kimera-RPGO is split between two parts: *(i)* a loop closure detection algorithm and *(ii)* a Pose Graph optimisation module that calculates a globally consistent pose.

3.4.3 Gauß-Newton

***(i)* Loop closure detection** is the detection of areas that the robot has previously visited. This is done so errors in the estimated map and pose can accumulate over time resulting in a offset. If loop closures are detected then these offsets can be corrected. Kimera-RPGO detects putative [4] loop closures by a bag of words representation and the use of DBoW2 [3] . A bag of words representation is a set that usually contains a histogram which groups visual words together. A visual word is created by grouping similar visual features in the observed environment together. Common features include keypoints and descriptors, which capture the unique characteristics of different parts of an image. If two of these bags of words are significantly similar a putative loop closure is detected.
Kimera-RPGO then verifies loop closures using mono and stereo geometric verification similar to 3.2.6*(i)*. This set can however still contain outliers for example through similar rooms in a building. The remaining loop closures are then passed to the robust PGO-solver.

***(ii)* Robust PGO-solver** uses a modified version of *Consistent measurement Set Maximization* (see 3.4.1) (PCM). PCM has been modified to not only work on a single robot, but also in a real time (online) use case. PCM has been further modified to also check its output using 1.Odometry and 2.Comparing previous loop closures to the newly detected one. In 1. outliers are detected by comparing odometry and PCM. If the PCM is a outlier it should differ from the odometry by a larger difference than that off measurement noise[5]. After this the loop closures are compared to previous ones. Kimera-RPGO also differs in how it builds its adjacency Matrix $\mathbf{A}$. While PCM builds a new $\mathbf{A}$ every time a loop closure is detected, Kimera-RPGO adds new rows and columns to its adjacency matrix each time a new loop closure is detected. This ensures the performance constraints needed for real time operation. Finally the largest subset is determined using a maximum clique algorithm (see 3.4.1) and the results added to the pose graph after some optimisation using the Gauß-Newton algorithm.

[4] putative loop closures are candidates for loop closures.
[5] Anomalous odometry outliers are filtered out through a Chi-squared test.

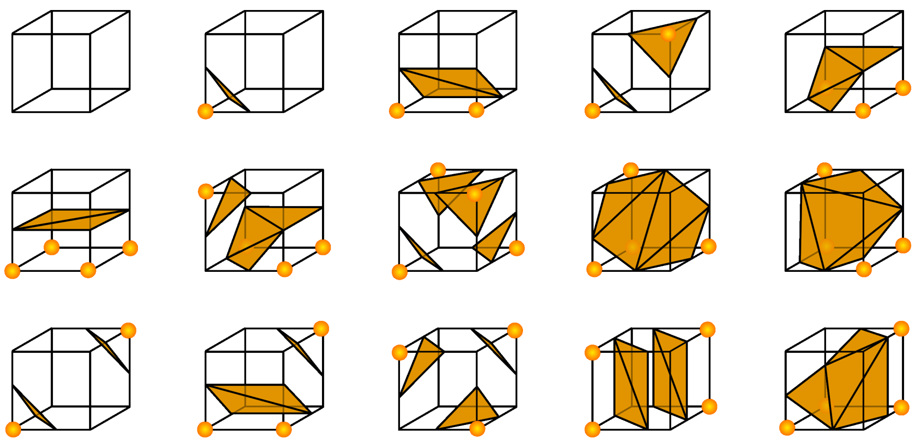

Figure 1: The 15 basic shapes of marching cubes (source [7])

3.5 Kimera-Semantics

3.5.1 Semi-global matching

Semi-global matching is a method used to compute the correspondences between pixels in a stereo image. This then yields a disparity image between the two images. A disparity image is a image which describes the apparent motion[6] between the images. The disparity image is calculated by finding pixels with similar intensity in the stereo pair. This does however result in many erroneous matches for each pixel. To prevent this a regularisation factor is used which penalises jumps in the disparity between adjacent pixels.

3.5.2 Truncated singed distance function (TSDF)

A signed distance function describes the distance of points in space to the nearest surface. If the point is outside of the object its sing is positive. If the point is inside the object the sing is negative. If the point is on the surface of a object it is 0. In this context truncation means that we only consider points with a certain maximum distance and discard the rest.

3.5.3 Marching cubes

Marching cubes is a algorithm that approximates a 3D mesh from a voxel grid. A voxel grid is a list of intensities that can be reconstructed to a Volume filled with 3D points with a certain intensity. Every corner gets its own index. If a Corner is active (i.e. on the inside of a object or in other words its intensity is positive[7]) its connecting edges form a triangle. This leads to 2^8 different configurations with 15 unique cases[8] as seen in 1.
Marching cubes loops over the voxel grid and creates the boxes filled with one of the 2^8 shapes according to the values of the voxels.

3.5.4 Overview

Kimera semantics serves two main functions: *(i)* Building a accurate 3D-Mesh and *(ii)* semantically annotating that mesh. If label are provided these two steps will occur at the same

[6] An example of apparent motion would be the perceived jump in the position of ones thumb when covering ones eye.
[7] the sings of the TSDF are flipped
[8] The missing cases are all symmetries of the 15 cases.

time. If no labels are provided then only the 3D-mesh will be build as semantic annotation is not possibe. The user of Kimera-Semantics also has to provide their own label generation. According to a comment by one of the contributors when asked about appropriate models on the GitHub of Kimera-Semantics any "any that outputs semantic labels on a per-pixel basis will do". Kimera-Semantics doesn't use the mesh created by Kimera-Mesher. This is due to higher level tasks requiring a more accurate 3D-mesh than those required by low level tasks like obstacle avoidance.

3D mesh generation and annotation is done in Kimera Semantics by producing 3D meshes using semi-global matching (see 3.5.1) by leveraging information about the camera position and geometry and triangulating the corresponding 3D positions. The provided labels are attached to the generated 3D-Point cloud. Voxblox is then used to create a TSDF (see 3.5.2) using grouped ray casting. In voxbloxes grouped ray casting each point is mapped to the nearest voxel on the voxel grid. The mean of all positions and colors is then taken and a ray cast to this mean. The probabilities of the labels are simultaneously generated. This happens by counting the number of label occurrences and normalizing that list. While the ray travels through the voxel grid the probabilities of the labels are updated accordingly, by Bayesian updates. This means that at every voxel the likelihood matrix containing the label likelihood is multiplied with the measurement frequency and added to the prior measurements. The result is then renormalized. The TSDF then gets constructed from the distances of the voxels to the surface. (todo stimmt das?) The label attached to each voxel is then most likly label in that voxel. Finally the marching cubes algorithm is used to extract a 3D-mesh with the labels.

3.6 Performance

To allow the Kimera Modules to run independently Kimera uses four Threads. This is required due to the differing run time requirements of the different components. The authors measured the preintegrating part to have a runtime of 40µs while the visual feature tracking takes a lot longer at an average of around 4.5 ms with a maximum of more than 60ms. This allows the front-end to estimate the position at over 200Hz. The VIO back-end runs slower at a mean runtime of 45ms, most of this time is the solving of factor graphs at 40ms. It is notable that the time for feature tracking at each frame is significantly shorter than the Feature detection, stereo matching, and geometric verification executed at each keyframe as seen in 2. The Kimera-Mesher generates the per-frame mesh at below 5ms and the multi-frame at below 15ms leading to a total run time of under 20ms. Due to Kimera-RPGO and Kimera-Semantics usually not being time critical for real time systems, as they would be used for higher level tasks like path planning, they run on slower threads then Kimera-VIO and Kimera-Mesher which are necessary for vital processes like obsticale avoidance. Kimera-RPGO has a measured runtime of less then 55ms. The longest runtime is that of Kimera-Semantics at an average of 100ms at each keyframe. Kimera-VIO runs on the first two threads, with its front-end running on the first thread. The second thread is shared by the Kimera-VIO back-end and Kimera-Mesher. This is due to Kimera-Mesher using the calculated pose of the VIO back-end to update the positions of the vertices of the multi-frame mesh. The third and fourth thread run Kimera-RPGO and Kimera-Semantics respectively.

3.7 Results

The authors of Kimera tested the modules of Kimera independently from one another. All tests were done on either measured real world datasets or done in a simulator. The authors of

Kimera compared their VIO approach to 5 other open source VIO piplines: OKVIS, MSCKF, ROVIO, VINS-Mono, and SVO-GTSAM. They tested these pipelines on 11 different datasets. Among these approaches four (OKVIS, MSCKF, ROVIO, VINS-Mono) implement fixed lag smoothing. SVO-GTSAM allows for full-lag smoothing and VINS-LC for Loop closure detection. In fixed-lag smoothing Kimera performed outright best on 8 out of the eleven datasets and being tied on one of the datasets. On the reaming two datasets Kimera measured second best. In full smoothing Kimera outperformed GTSAM 6 times while being tied twice. It is also notable that GTSAM failed completely three times while Kimera was able to run all tests. In PGO Kimera outperformed VINS-LC 9 out of 11 times. It is also Notable that Kimera-VIO incurs a small drift of below 0.2 % in longer trajectories. The authors also measured the impact of the tunable parameter of Kimera-RPGO and found that it only impacts the performance of Kimera by a negligible amount when using PCM. They measured a RMSE of 0.05 m at a α of 10 and a RMSE of 0.49 m at a α of 0.001, when using. Without the use of PCM the error increases to a maximum of 1.59 m at a α of 0.001. To evaluate the performance of Kimera-Mesher testing on six sequences in the EuRoC V1 and V2 datasets was done. The performance of both the Multi-frame mesh as well as the global mesh was determined. The Multi-frame mesh hat a RMSE of 0.374 m and 0.530 m, while the error of the Global mesh was between 0.353 m and 0.480 m. The Global Mesh outperformed the Mult-frame mesh in four out of the six tests, ranging from 12 % to 24 % lower RMSE. In the cases where the multi-frame mesh was more accurate the difference was less than 3 %. (todo mayby completness?) The testing of Kimera-Semantics was done using a unity based simulator. The authors noticed small geometric and semantic errors with bundled raycasting, while dense stereo introduces notable errors, especially in texture-less areas like walls. Using ground truth depth and pose information provided by the simulator a semantic accuracy of 94.68 % was achieved, with a geometric RMSE of 0.079 m. The impact of the use of Kimera-Vio is mainly visible in the 3D mesh with its RMSE increasing to 0.131 m, while the semantic accuracy remains at 94.50 %. The use of Dense stereo instead of ground truth depth sees both measures worsen to a semantic accuracy of 80.74 % and a RMSE of 0.215 m. It should be noted that the RMSE of the metric-semantic mesh of 0.215 m is still much smaller than the best case Kimera-Mesher RMSE of 0.353 m.

4 Running Kimera

Compiling and running Kimera-Semantics was possible after changing the c++ version Kimera was compiled with. When testing Kimera-Semantics with the provided rosbag Kiemera-Semantics seemed to perform as described.

Kimera-VIO and by extension Kimera-RPGO and Kimera-Mesher are able to be run in three main ways. The first is compiling all the packages from source. The second way is through the use of a docker container and third way is by using ROS. While Kimera-VIO was able to run through docker without any problems [9] running Kimera using ROS already failed during compilation. This seems to be due to using the development branch of GTSAM. After finding versions of GTSAM Kimera-VIO could compile with Kimera-VIO still does not behave propel. The Kimera-VIO front-end ROS node seems to die after a minute of use. During the time Kimera-VIO is working it does seem to perform as intended using the EuRoC dataset.

[9] As far as i am aware the is no graphical output

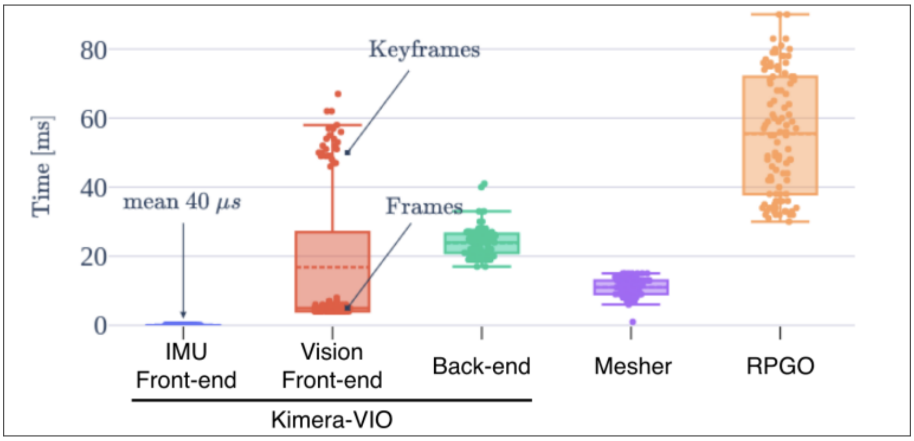

Figure 2: Kimera timings (see [6])

5 Conclusion

Kimera presents a c++ Library for metric-semantic Visual-Inertial Simultaneous Localization And Mapping that is modular and able to perform in a low performance environment, allowing further research and developments in several distinct field, ranging from simultaneous localisation and mapping to semantic segmentation. Kimeras Modules generally perform well, with the combination of Kimera-VIO and Kimera-RPGO outperforming similar state of the art libraries. Kimera-Meshers mesh is build quickly and allows for low level navigation tasks, while Kimera-Semantics builds an accurate metric-semantic mesh.

References

[1] F. Dellaert et al. GTSAM - georgia tech smoothing and mapping. https://gtsam.org/. Accessed: December 28, 2023.

[2] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, February 2017.

[3] Dorian Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[4] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th international joint conference on Artificial intelligence*, volume 2, pages 674–679, 1981.

[5] Todd Lupton and Salah Sukkarieh. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Transactions on Robotics*, 28(1):61–76, 2012.

[6] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping, 2020.

[7] Ryoshoru. Marching cubes cases. Accessed: January 2,2024.

[8] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.