# Build an Offline Python SLAM using COLMAP

Vukasin Bozic Alan Savio Paul Maximum Wilder-Smith Markus Pobitzer
ETH Zürich
{vbozic, alpaul, mwilder, pobmarku}@student.ethz.ch

Supervised by Sarlin Paul-Edouard

## Abstract

*State-of-the-art Structure from Motion algorithms such as COLMAP are highly robust in reconstruction but are slow and often don't scale well. This makes them unsuitable for long video data. On the other hand, SLAM systems can process videos (sequential images) in real-time but fall behind COLMAP in map quality. The goal of this project is to combine the best of both worlds to obtain a fast, robust and scalable SLAM system. We demonstrate that we partially achieved our goals by utilizing components of COLMAP and ideas from ORB-SLAM. The quality of our map, however, is not yet comparable with the state-of-the-art.*

## 1. Introduction

Simultaneous localization and mapping (SLAM) algorithms have numerous use cases from robotics to XR applications. They work by generating a map, often from some form of visual data, and localize that sensor as it moves through an environment. This results in a map of the traversed environment as well as the path that the sensor took. This approach is often quite good at positioning the sensor in a 3D space as well as running quickly, in near real-time, and efficiently. The downside however, is that the generated maps often lack robust details. Structure from motion (SfM) algorithms on the other hand generate very detailed maps, however they can be quite cumbersome. One of the leading structure from motion algorithms, COLMAP [16], was designed to process images from the internet to reconstruction detailed structures. While this allowed for reconstruction that was robust to different camera properties and types, it meant massive correspondence graphs needed to be created. This can result in very long processing time as well as large memory requirements.

This work aims to combine the efficient sequential tracking of SLAM systems with the robust reconstruction capabilities of COLMAP. To do so we utilize the Python bindings, pyCOLMAP [6], and our own SLAM pipeline with inspiration from ORB-SLAM [4]. While these systems individually have the functionality that we desire, this pipeline aims to combine the robust reconstruction and triangulation of COLMAP with the benefits of sequential data processing from SLAM. In this way the system can keep processing time and memory low for each additional keyframe, while building up the COLMAP reconstruction.

## 2. Related Work

**Structure from Motion (SfM)** is the process of reconstructing a 3D structure from a series of images from different viewpoints. There exists many types of SfM systems such as incremental, hierarchical, and global approaches. In this project we are most interested with one of the most popular SfM systems, COLMAP [16], which uses the incremental strategy.

In terms of performance, COLMAP works better on average when compared against other SfM methods and has gained widespread attention since it's open-source release in 2016. It has been shown to produce 3D structures with superior robustness, completeness, and accuracy, compared to previous state-of-the-art systems. However, SfM systems are not designed to work specifically on video data and thus we cannot leverage their benefits for such datasets feasibly. In our work, we restructure the COLMAP pipeline to make it work like a SLAM system. Figure 1 illustrates the pipeline.

**Simultaneous Localization and Mapping (SLAM)** can be viewed as a specific case of the SfM problem. It consists of estimating a map of the environment as well as localizing the agent which is moving through this environment. SLAM is a fundamental problem in robotics, providing crucial information for autonomous navigation of cars, drones, and consumer robots.

Our work is most interested with the subset of SLAM algorithms that take only monocular camera images as input. There are other methods that utilize various other sensors like GPS, IMU and depth. One of the most popular and successful feature-based SLAM algorithms is ORB-SLAM

[9, 10, 4].

In our work, we only took a few ideas from ORB-SLAM such as their keyframe insertion methods, and built our own SLAM system using the components of COLMAP. The most important difference in our keyframe decisions is that we include the optical flow constraint (See section 3.2). Our system performs all processing on a single thread and uses a separate thread only for the purpose of visualization (See Section 3.7), and is overall much simpler than ORB-SLAM. Moreover, we did not implement loop closure.

**Feature Detection and Matching** techniques can be divided into traditional and deep learning-based methods. The traditional detectors considered in our project are SIFT [8] and ORB [12]. As its name suggests, SIFT is scale invariant, but it is not always suitable for SLAM-like systems because it is not real-time. ORB builds on the well-known FAST keypoint detector [11] and the BRIEF descriptor [3], which have good performance and work in real-time. This makes ORB a good fit for SLAM and has been successfully employed in state-of-the-art SLAM algorithms like ORB-SLAM [9].

However, such traditional methods often fail to work in the presence of noise, and can sometimes return clustered keypoints which can hurt homography estimation. Various machine learning and deep-learning based methods have recently gained attention because of superior improvement in the quality of detected feature points and their descriptors. The main reason for this improvement is the replacement of handcrafted extractors with extremely powerful and expressive neural networks. In our project we are most interested in SuperPoint [5].

Typically, descriptors are matched with a Nearest Neighbour (NN) search. This method gives good results and is suitable for real-time applications. Recently, more powerful deep learning-based matching techniques have started replacing the simple NN matchers. SuperGlue [14] is one recent work that we are most interested in. It performs context aggregation, matching and filtering in a single end-to-end architecture and has been shown to provide extremely high quality matches. The downside, however, is that it consumes much more time than simple methods like NN matching. In our project, we work with SuperPoint + SuperGlue and ORB + NN.

# 3. Method

We present an offline SLAM pipeline built on COLMAP a SfM system. Since SLAM and SfM try to solve similar questions, we can reuse some main components of COLMAP. This includes the Reconstruction object that stores the map and all its relevant information, the correspondence graph for 2D correspondences between images and the Triangulator object. An overview of the pipeline is depicted in Figure 1.

## 3.1. Feature Detection and Matching

Our system is built with two feature detectors and two matchers.

The first is ORB detector which is available in the OpenCV library. We couple this with Nearest Neighbour matching where the distances are computed using the Hamming Norm. This is a suitable norm because the ORB descriptors are vectors containing binary values.

The second detector we used is the SuperPoint detector, which is publicly available [1] along with pretrained weights. The features are then matched using the SuperGlue network which is also available in the hloc library [13].

## 3.2. Optical Flow Estimation

We estimate flow between two frames with the 2D correspondences found in the previous step. We work here with 2D images and therefore this is an ill posed problem since the movements are in 3D space. Given image point $p_1 = (x_1, x_2)$ in the first image and $p_2 = (x_1, x_2)$ in the second image we estimate the flow between them as follows:

$$flow(p_1, p_2) = \left( \frac{|x_1 - x_2|}{f_x}, \frac{|y_1 - y_2|}{f_y} \right)$$

where $f = (f_x, f_y)$ is the focal length of the camera. We divide through the focal length to factor in different cameras.

For the final flow value we take the median of all points and sum the two dimensions up. The flow value needs to exceed a threshold when adding a new keyframe. This helps us filter out image pairs that have a small flow and therefore baseline. Our method fails when there is only rotation or the feature matcher produced too many outliers.

## 3.3. Map Initialization

An accurate map initialization is vital [9]. Only if the initial image pair has a good pose estimation, a point accurate reconstruction can be made and absolute poses of new frames can be estimated. The initialization is based on two frames chosen from the beginning of the video where we consider only the first $N$ frames, $N$ can be chosen arbitrarily by the user. The features in a frame get detected with the chosen detector, and then we exhaustively match the frame with all the other $N - 1$ frames.

Similar to COLMAP we select our first image as the frame that has the most correspondences to the other frames. Based on the chosen image, we try to find a good second image to start our map. The second image should have a lot of correspondences to the first as COLMAP must succeed in estimating the relative pose between the images and the motion between the images needs to be large enough

[1] https://github.com/cvg/Hierarchical-Localization

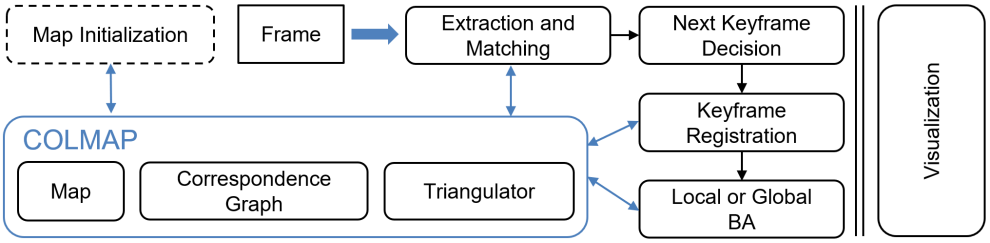


Figure 1. Overview of the SLAM pipeline. In the bottom left corner, in blue, are the used COLMAP components. The small blue arrows indicate the use of COLMAPs components from our pipeline. After Map Initialization we sequentially look at every frame and decide if it can be used as our next keyframe. In parallel we visualize the map and the last keyframe compared to the current frame.

as discussed before. COLMAP´s Triangulator object is used to triangulate map points from these two images if all mentioned conditions hold. Afterwards, we use global bundle adjustment and filter out inaccurate map points. If there are enough map points, we continue, otherwise we try out another image pair and repeat the process.

The choice of $N$ is dependent on the frame rate, the number of trackable features in a frame, and the movement speed and rotation of the camera. $N$ should be high enough to capture a sequence that provides enough movement leading to a wide baseline between poses and some good features to track and match.

### 3.4. Next Keyframe Selection

Inspired by the keyframe-based approach in ORB-SLAM, our work also uses keyframes although with different criteria in the selection process. It is crucial to correctly decide which frames become keyframes and which ones are discarded as this has a significant effect on the final reconstruction. Poor keyframe selections can gradually accumulate errors and eventually result in a low quality reconstruction. To insert a keyframe, two conditions must be met:

1) The median of all the optical flow constraint values in the current image must be above 0.05 (see Section 3.2)

2) Current frame tracks at least 50 points. In other words, there must be at least 50 matches between the current image and the last registered keyframe.

Condition 1 ensures sufficient parallax which will lead to lower uncertainties in the triangulation. This also acts as a proxy for motion estimation (assuming there is no rotation). We chose a median score here so that outliers in the matching cannot significantly affect the final score for this image. Outliers are likely to have very high disparity, and thus could have an unwanted effect on the score for the image.

Condition 2 ensures that we have a good tracking.

The limitation with our keyframe selection is that images with pure rotation will be selected as keyframes and triangulation will be attempted. Handling such degenerate rotation-only camera motion is a challenge that is inherent to visual monocular SLAM.

### 3.5. Registration and Triangulation

For registering a new keyframe and triangulating it, we heavily rely on COLMAP´s functionality. After registration of the new keyframe the corresponding image is triangulated in the map. When we triangulate it, 3D map points get created based on 2D image correspondences to the previous keyframes. We proceed with a step of filtering and a check if the keyframe has at least 50 map points. If the check succeeds, we move on to the optimization step, otherwise we deregister it and start over with selecting a new keyframe.

### 3.6. Bundle Adjustment

During bundle adjustment optimization, the reprojection error gets minimized by refining camera parameters and point parameters jointly [16, 9]. Whenever a new keyframe gets successfully registered, we either optimize a small part of the map with local BA or the whole map with global BA, depending on how much the map grew since the last global BA step. During the optimization we fix some camera parameters as seen in [16] as this helps for convergence. In local BA, we find at most six keyframes that have the highest correspondences to the current keyframe.

The actual minimization problem gets solved with Ceres [1], where we use corresponding Python bindings [15]. For optimization purposes we solve problems with a small number of cameras with a dense Schur method [7] and ones with more cameras with an iterative approach [2] as discussed in [16].

After the optimization we normalize the map, merge and complete tracks and filter out points that have a large reprojection error. The normalization helps to improve numerical stability of the BA algorithm [16]. The merged and completed tracks make the map more accurate and the filtering gets rid of points that could otherwise be used to inaccurately estimate absolute poses of coming frames.

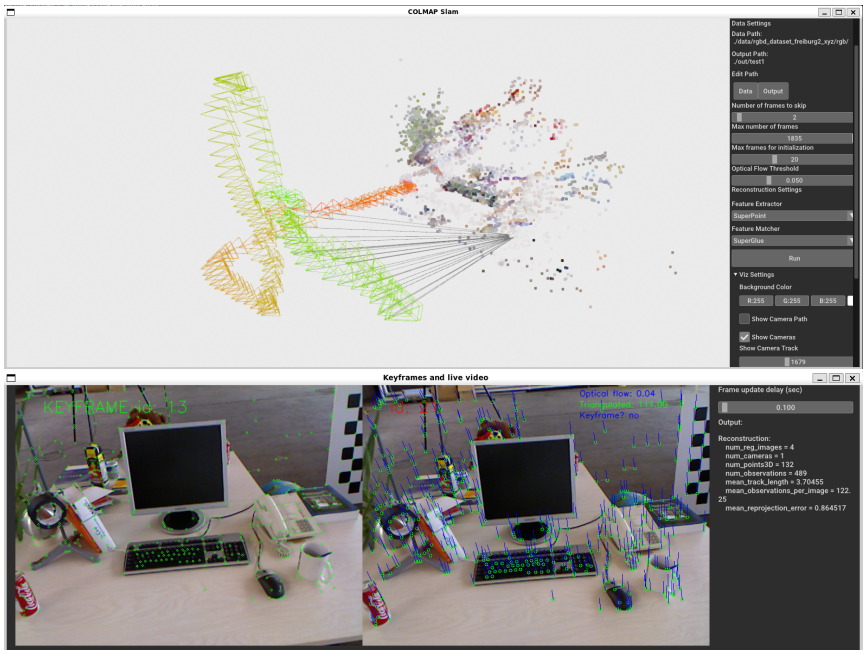


Figure 2. Top: Reconstruction view of the main window showing the Frieburg desk data. Bottom: Frame viewer window with the last registered keyframe (left), current frame (center) and summary (right)

## 3.7. Visualization

### 3.7.1 Reconstruction View

A majority of the primary viewer window, top of Figure 2, shows the reconstructed points from the pipeline and the estimated camera path through those points. This rendering, as with the GUI, is created using Open3D [17]. We first use COLMAP to color the points based on their images and render the points as a point cloud in the viewer. For the cameras, we visualize a frustum based on the ratio of the image's dimension that is scaled by a user-defined scale value. Each camera's frustum is colored based on recency, with the first being green and the last being red. Additionally, we show a line through the cameras, again colored from green to red, to show the path the camera follows.

### 3.7.2 Frame View

The application spawns a second window, bottom of Figure 2, to visualize the frames as they are being processed. In this view we show the last registered keyframe along with its ID on the left side. In the center, as each frame is being processed and undergoes keyframe selection it is visualized with the optical flow lines (in blue) as well as the optical flow score and correspondence count. If these surpass the required threshold the image is registered as the next keyframe and the main viewer is updated with the new points and camera. In order to visualize this processing, a per keyframe callback is used to pass the image updates back to the main render thread from the pipeline thread. On the right side of this panel we show the summary of the reconstruction up until that point, this allows us to keep track of changes during each registration.

### 3.7.3 Settings

On the right side of the reconstruction window is a sidebar with various settings. At the top of the side bar is an option to export or load an old reconstruction for visualization and analysis. Below this are the pre-reconstruction settings shown in Figure 3 on the left. Amongst these settings are options to change the data/output directories, data processing parameters, and the feature matchers and extractors.

As some datasets have either a high frame rate or very minimal motion between frames we included a frame skip slider. This allows the user to only load every $s$ frames in the raw data. Although optical flow will dynamically check for motion between frames while the pipeline runs, this option helps decrease memory consumption for long datasets. The second slider also helps reduce memory and excess image processing on larger datasets by allowing the user to only process a subsection of the data. The slider sets the maximum number of frames to process, which is again helpful if the user has issues with memory consumption. The max value of this slider is automatically updated based on the dataset selected and the number of frames to skip.

For the reconstruction itself, we have a slider to change the number of images to consider as part of initialization.


Figure 3. Left: Sidebar of the reconstruction window showing various pre-reconstruction settings. Right: Sidebar with the visualization settings.

As mentioned in Section 3.3 a good map initialization is crucial for a successful reconstruction and triangulation. Below this setting is a slider for the optical flow threshold. Increasing this value increases the amount of motion needed before an image can be considered as a keyframe. Next are drop down menus to select the feature extractor and feature matcher for the pipeline, presently SuperPoint and ORB for the extractor and SuperGlue and Nearest Neighbors for the matcher. Once the final configuration of the pipeline is set, the "Run" button can be pressed which starts the pipeline in a separate thread.

Once a reconstruction is complete, the user can adjust the visualization settings as shown in Figure 3 on the right. The first of these settings is to change the background color of the point visualizer for better contrast. Below this are two toggle boxes to enable or disable the camera path and the cameras themselves. For analysis of the reconstruction we also included a track slider. The value in the slider corresponds to the id of a 3D point, by selecting a value greater than -1, the visualizer will draw a line from the selected point to all cameras that tracked this point. There are also sliders to scale the camera frustum and the size of the 3D points as well as a "Reset Camera" button which realigns the viewer camera based on the reconstruction data. The last two settings in the visualization settings allows clipping of 3D data to points, cameras, and tracks that are between a certain start and end image. This again allows for analysis on when certain points are introduced and how the reconstruction has changed over time.

# 4. Experiments

This section is divided into 3 parts; in 4.1 we justify the choice of datasets and explain the criteria for choice, in 4.2 we perform evaluations on some of the datasets and comparisons to state-of-art systems, in 4.3 we provide an ablation study which deals with optimizing hyperparameters used in reconstruction settings finally, in 4.4 we provide some reconstruction examples.

## 4.1. Data sets

For the choice of datasets a few factors were taken into consideration:

- Easy evaluation - availability of metrics for our pipeline to compare to state-of-the-art SLAM systems
- Variability of motion - minor movement and dynamic sequences recorded while walking or driving
- Variability of setting - indoor and outdoor sequences

Consequentially, 5 representative datasets were chosen, 3 indoor static sequences and 2 outdoor dynamic sequences filmed while driving. For the static sequences an online evaluation tool with ground truth is provided so that direct assessment of our method and respective comparison with state-of-art was a straightforward task. Indoor sequences *fr1_rgb*, *fr2_rgb* and *fr1_desk* were chosen from a collection of RGB-D datasets[2], and include the intrinsic camera parameters. For the 2 outdoor sequences, images were extracted from the videos *test_kitti984.mp4* and *test_ohio.mp4* [3].

## 4.2. Evaluation metric

In table 1, our system is compared to other SLAM systems in terms of RMSE error between estimated poses and ground truth. The following set of hyperparameters was used: Number of frames to skip : 2, Max frames for initialization : 20, Optical flow threshold: 0.05.

| RMSE [cm] | fr1_xyz | fr2_xyz | fr1_desk |
|---|---|---|---|
| ORB-SLAM | **0.90** | 0.30 | **1.69** |
| PTAM | 1.15 | **0.20** | - |
| RGB-D SLAM | 1.34 | 2.61 | 2.58 |
| Ours | 10.93 | 0.26 | 19.57 |

Table 1. Comparison of our system to state-of-the-art systems

[2]https://vision.in.tum.de/data/datasets/rgbd-dataset
[3]https://github.com/geohot/twitchslam/tree/master/videos

| | ORB + NN | SuperPoint + SuperGlue |
|---|---|---|
| Initialization [sec] | **1.3** | 4.7 |
| Frame registration [sec] | **0.47** | 1.08 |
| 3D points per image | 306 | **337** |
| Reprojection error | 0.7550 | **0.6512** |

Table 2. Comparison of performance for different extractor/matcher. The Initialization is per frame.

Furthermore, a comparison of system performance with different extractors/matchers was conducted. In table 2 results are shown for ORB+NN and then SuperPoint+SuperGlue respectively. Additionally, the execution time of specific steps is provided in order to highlight impact of feature processing on their execution time and reconstruction quality. As SuperPoint+SuperGlue outperformed the ORB system it is used for evaluation and shown in Figure 4 for *fr2_rgb*.

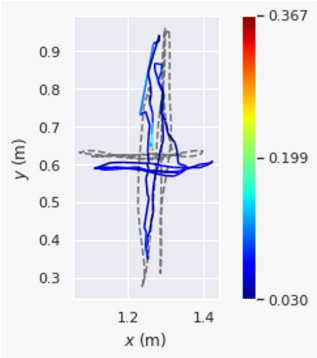


Figure 4. Evaluation of *fr2_rgb*

### 4.3. Ablation study

Analysis of the affect of hyperparameters changing on reconstruction quality in terms of RMSE and reprojection error was conducted. This was run with *fr2_rgb* (first 1 minute, 2000 frames) as it performed best. In Table 3 the results are presented, where in each row the value of 1 parameter was being changed while others were kept the same as Section 4.2 (frame skip: 2, initialization size: 20, flow threshold: 0.05).

| Frames skip | 2 | 5 | 10 |
|---|---|---|---|
| | **0.095** | 0.098 | 0.15 |
| Frames for init | 10 | 20 | 40 |
| | 0.115 | **0.095** | 0.096 |
| OF threshold | 0.02 | 0.05 | 0.1 |
| | 0.11 | **0.0955** | 0.097 |

Table 3. RMSE[cm] for varying frame skips, initialization size and optical flow

### 4.4. Reconstruction examples

Apart from numerical results, 2 samples of dynamic reconstructions (Figure 5) are provided. For these sequences ground truths weren't available, thus we can only visually evaluate its veracity. Videos of the reconstructions for *kitti984.mp4*[4], *fr1_desk*[5] and *fr2_xyz*[6] are available online.

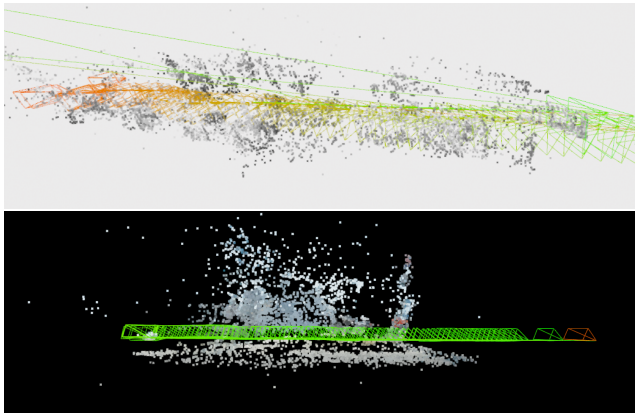

Figure 5. Reconstruction of *test_kitti984.mp4* (top) and *test_ohio.mp4* (bottom).

## 5. Conclusion

This work aimed at restructuring a state-of-the-art SfM method into an efficient and robust SLAM pipeline which was partially achieved. Core COLMAP functionalities are utilized as the foundation of our system, while a few novelties are incorporated into the usual SLAM pipeline to improve robustness and overall accuracy. Still, on evaluated datasets it is obvious that our system is not robust enough as it matches or under performs existing SLAM systems, and can generate poor reconstructions. However, this paper presents a good starting point as well as a development tool that provides plenty of visualization options for debugging and thorough analysis.

## 6. Contributions of team members

- Markus: Map Initialization; Registration, Triangulation and Bundle Adjustment; Main Loop of pipeline.
- Alan: Keyframe selection and registration; Optical Flow; Visualization; General development and testing
- Vukasin: Development of general pipeline; Keyframe selection and registration; Datasets selection and evaluation; Ablation study.
- Maximum: Open3D visualization; Threading for reconstruction; Spatial evaluation.

[4]https://youtu.be/EojVGsfVN7s
[5]https://youtu.be/Nt3mH8UF_9g
[6]https://youtu.be/fbRbJeR3u8M

## References


[1] S. Agarwal, K. Mierle, and T. C. S. Team. Ceres Solver, 3 2022.

[2] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010.

[3] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[4] C. Campos, R. Elvira, J. J. Gomez, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[5] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018.

[6] M. Dusmanu, P.-E. Sarlin, and J. Lambert. colmap/pycolmap: Python bindings for colmap, 2022.

[7] M. I. Lourakis and A. A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):1–30, 2009.

[8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[9] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[10] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[11] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[13] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *CVPR*, 2019.

[14] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. SuperGlue: Learning feature matching with graph neural networks. In *CVPR*, 2020.

[15] P.-E. Sarlin and P. Lindenberger. cvg/pyceres: Factor graphs with ceres in python, 2022.

[16] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

[17] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.