A review on ORB-SLAM-2 paper

Sally Robotics · Follow
8 min read · Sep 7, 2020

-By Kanishk Vishwakarma, SLAM Researcher @ Sally Robotics.

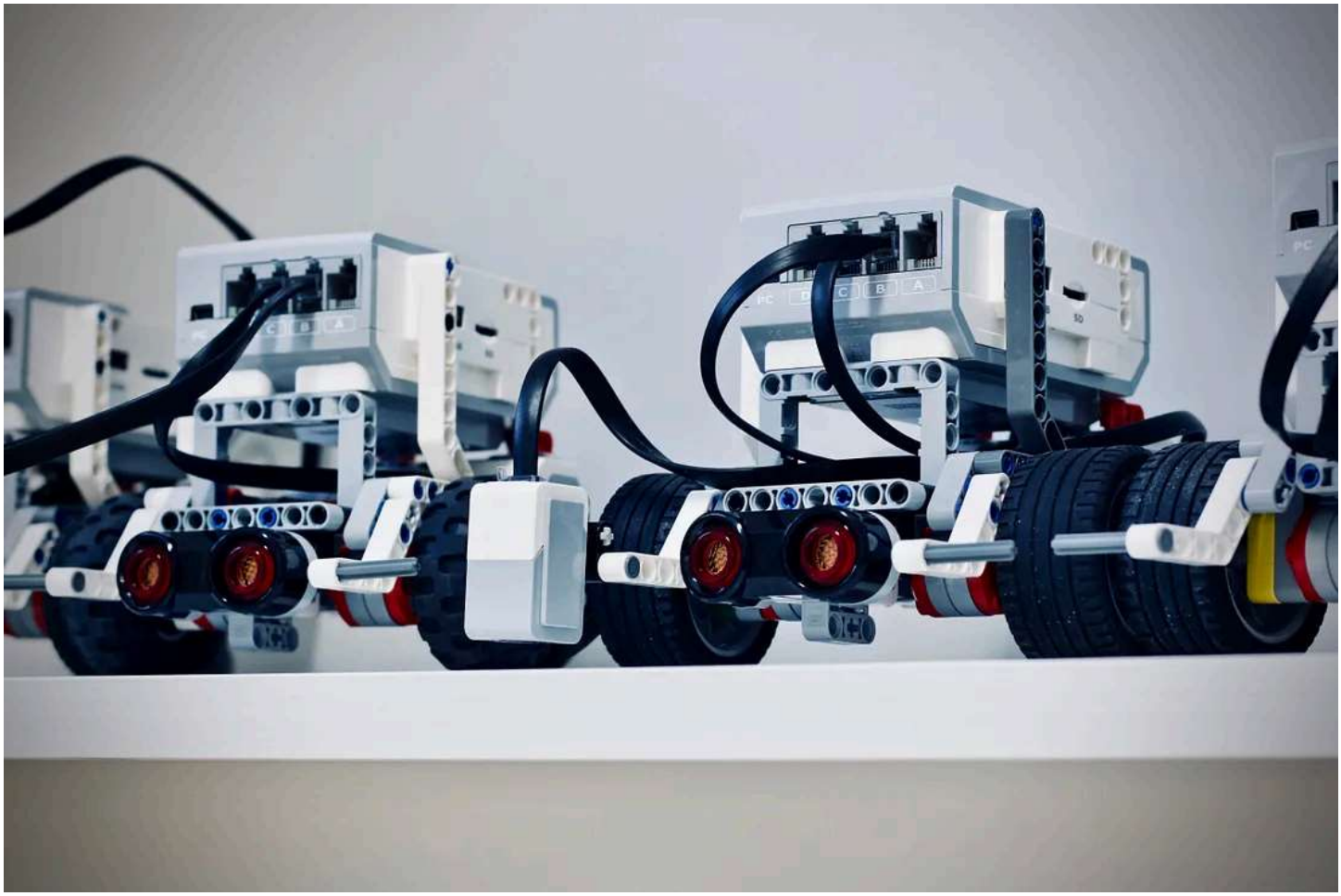
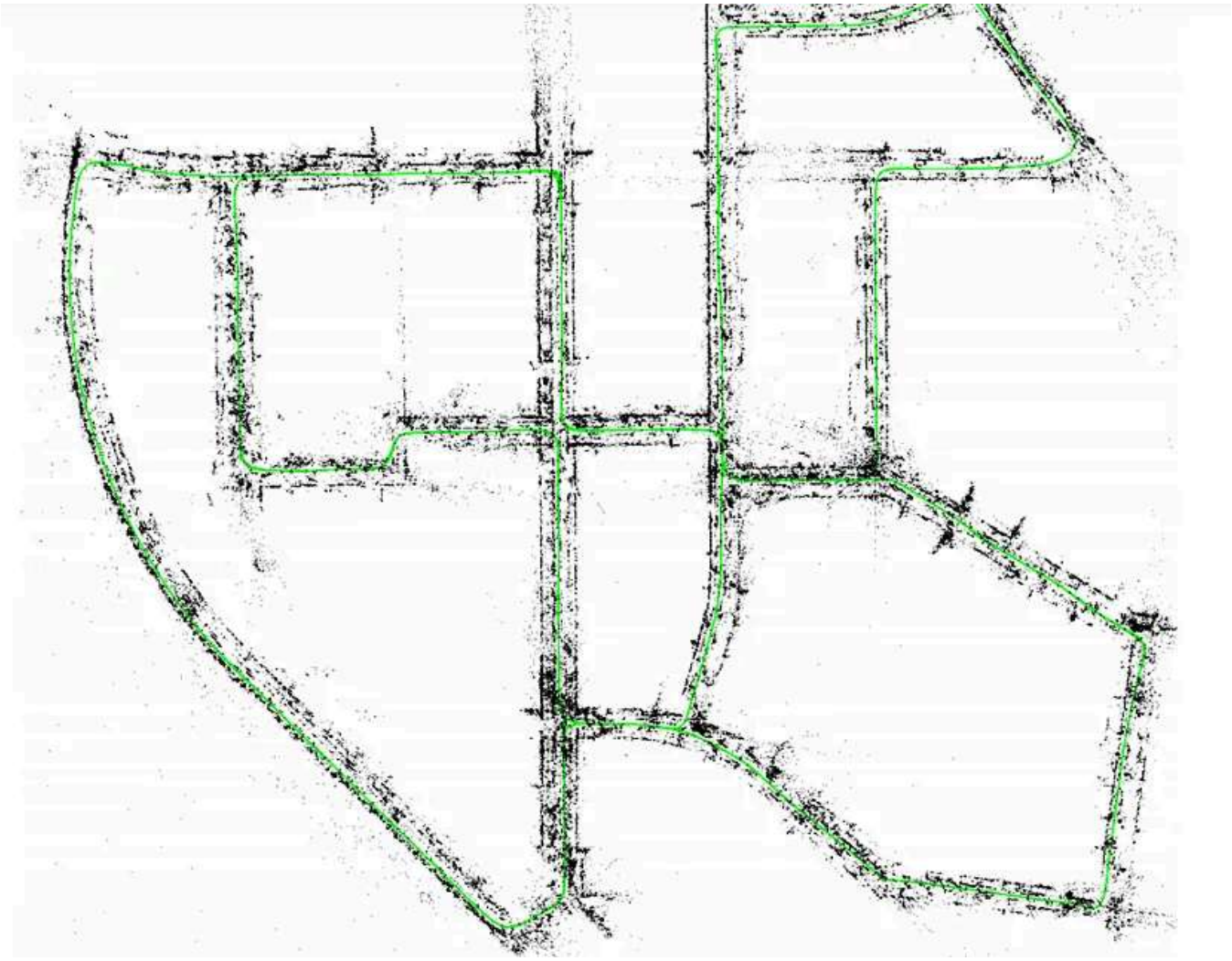


Photo by Jelleke Vanooteghem on Unsplash

Introduction

This paper starts with explaining SLAM problems and eventually solving each of them, as we see in the course of this article. Certain problems like depth error from a monocular camera, losing tracking because of aggressive camera motion & quite common problems like scale drift, and their solutions are explained pretty well. Let's first dig into how this algorithm works.

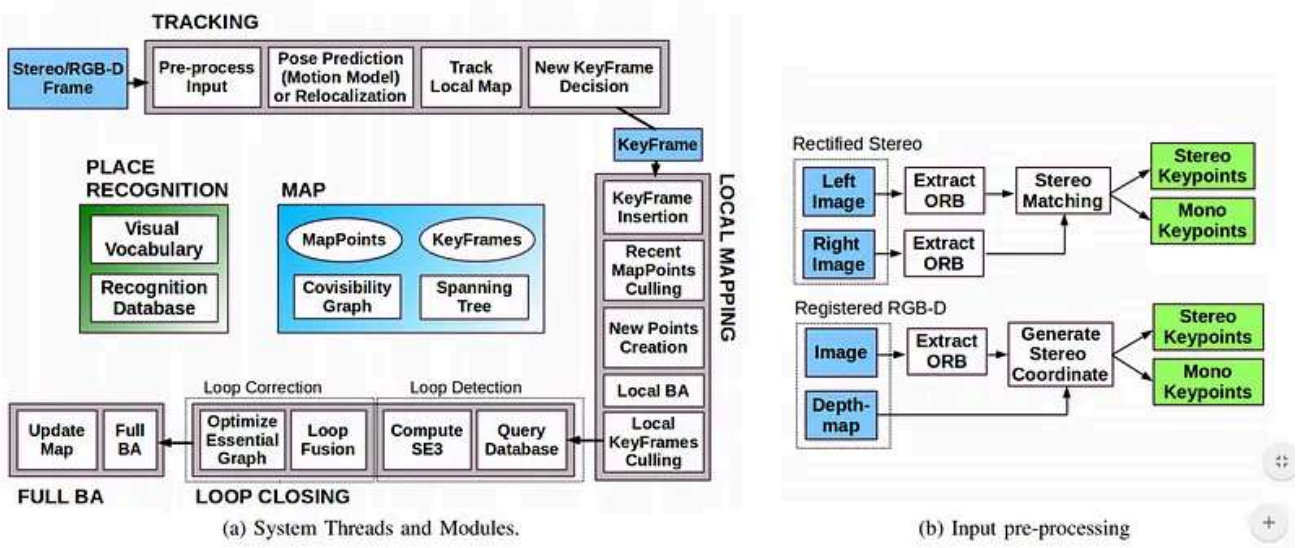


ORB-SLAM 2 implementation on the popular KITTI dataset

The Algorithm!

ORB-SLAM2 works on three tasks working simultaneously: tracking, local mapping & loop closing. In its tracking part, ORB-SLAM2 does frame-by-frame feature matching and compares them with a local map to find the exact camera location in real-time. It does a motion-only bundle adjustment so as to minimize error in placing each feature in its correct position, also called as minimizing reprojection error. Then comes the local mapping part. ORB-SLAM2 makes local maps and optimizes them using algorithms like ICP (Iterative Closest Point) and performs a local Bundle Adjustment so as to compute the most probable position of the camera. Finally, it uses pose-graph optimization to correct the accumulated drift and

perform a loop closure. It's necessary to perform Bundle Adjustment once after loop closure, so that robot is at the most probable location in the newly corrected map. After the addition of a keyframe to the map or performing a loop closure, ORB-SLAM2 can start a new thread that performs a Bundle adjustment on the full map so the location of each keyframe and points in it get a fine-tuned location value.



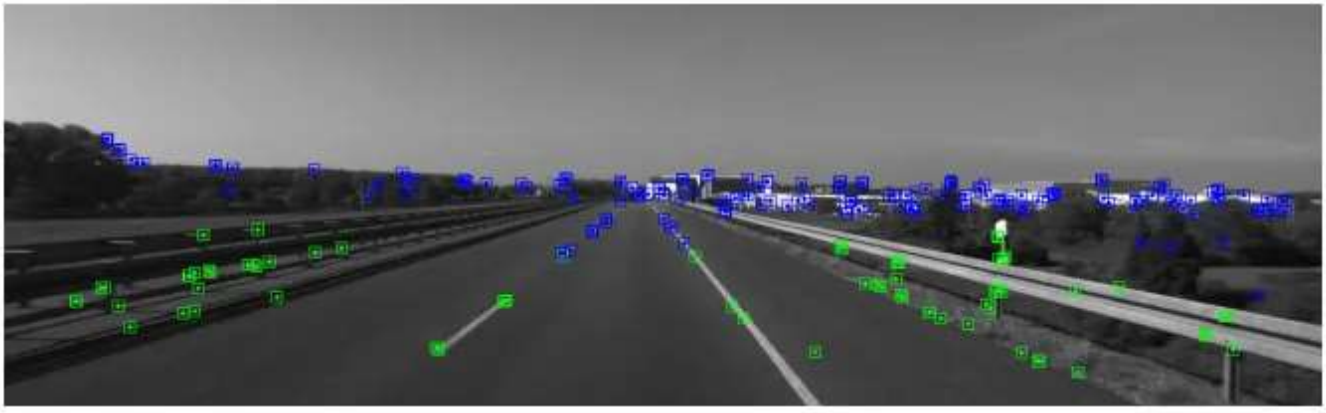
Graphical summary of what actual algorithm looks like

In its III-A section explaining monocular feature extraction, we get to know that this algorithm relies only on features and discards the rest of the image. Although as a feature-based SLAM method, it's meant to focus only on features than the whole picture, discarding the rest of the image (parts not containing features) is not a nice move, as we can see Deep Learning and many other SLAM methods using all the image without discarding anything which could be used to improve the SLAM method in some way or the other.

Features

This paper explains Stereo points (points which were found in the image taken by the other camera in a stereo system) and Monocular points (points which couldn't be found in the image taken by the other camera in a stereo system) quite intuitively. It's a really nice strategy to keep monocular points and using them to estimate translation and rotation. Also, this paper explains a simple mathematical formula for estimating the depth of stereo points and doesn't include any kind of higher mathematics which may increase the length of this overview paper unnecessarily. It tells that close points can be used in both calculating rotation and translation and they can be triangulated easily. But the calculation of translation is a severely error-prone task if using far points. That's why it triangulates them only

when the algorithm has a sufficient number of frames containing those far points; only then one can think of calculating a practically approximate location of those far feature points.



Far features are blue while close ones are green

The system bootstrapping part tells how RGB-D cameras are used in reducing initialization time, but we know that initialization time is already quite less and it doesn't matter whether the algorithm initializes immediately, or takes a few milliseconds, as long as we don't want it to initialize while at a stop. If the vehicle is standing still and we need it to initialize the algorithm without moving, we need RGB-D cameras, otherwise not. And oh, not to forget self-driving race cars, timing matters a lot in races.

The Math Part

In part III.C of this paper, the use of Bundle adjustment in ORB-SLAM2 is explained pretty well. It's divided into three categories, Motion only Bundle Adjustment, Local Bundle Adjustment & Full Bundle Adjustment. In motion only bundle adjustment, rotation & translation are optimized using the location of mapped features and the rotation and translation they gave when compared with the previous frame (much like Iterative Closest Point). In local bundle adjustment, instead of optimizing the camera's rotation and translation, we optimize the location of Keypoints and their points. In full bundle adjustment, we optimize all the keypoints and their points, keeping the first marked keyframe, to avoid the drift of the map itself. The mathematics behind how ORB-SLAM2 performs bundle adjustments is not much overwhelming and is understandable, provided the reader knows how to transform 3D points using rotations and translation of camera, what's Huber loss function, and how to do 3D differential calculus (partial derivatives).

Loop Closure

Proceeding to III-D now comes the most interesting part: Loop closure. Loop closure in ORB-SLAM2 is performed in two consecutive steps, the first one checks if a loop is detected or not, the second one uses pose-graph optimization to merge it into the map if a loop is detected. This section clearly mentions that scale drift is too large when running ORB-SLAM2 with a monocular camera. You'll need to look for similarities and scale changes quite frequently and this increases workload. With stereo cameras, scale drift is too small to pay any heed, and map drift is too small that it can be corrected just using rigid body transformations like rotation and translation during pose-graph optimization. To fine-tune the location of points in the map, a full bundle adjustment is performed right after post-graph optimization is performed. As a full bundle adjustment takes quite some time to complete, ORB-SLAM2 processes it in a separate thread so that other parts of the algorithm (tracking, mapping, and making loops) continue working. Now think for yourself, what happens if my latest Full Bundle Adjustment isn't completed yet and I run into a new loop? This should come pretty intuitively to the reader that we need to prioritize the loop closure over Full Bundle Adjustment, as a full bundle adjustment is used to just fine-tune the location of points in the map, which can be done in the future, but once a loop closure is lost, it's lost forever and the complete map will be messed up (See table IV for more information on time taken by different parts of the algorithm under different scenarios). So obviously we need to pause full bundle adjustment for the sake of loop closure so that it gets merged with the old map and after merging, we re-initialize the full bundle adjustment. Loop closure is explained pretty well in this paper and it's recommended that you peek into their monocular paper [3].

Reading III.E section of this paper proves that ORB-SLAM2 authors have thought about inserting new keyframes quite seriously. This new concept of keyframe insertion uses another concept of 'close' and 'far' feature points. If the depth of a feature is less than 40 times the stereo baseline of cameras (distance between focus of two stereo cameras) (see III.A section), then the feature is classified as a 'close' feature and if its depth is greater than 40 times, then it's termed as a 'far' feature. Guess what would be more for better performance of the algorithm, the number of 'close' features, or the number of 'far' features? If 'close' features are more than localization processes better and those features are triangulated better. Authors' experiments show that if the number of previously tracked close feature points drops below 100, then for the sufficiently good working of the algorithm, there should be at least 70 new close feature points in this new frame. If it's not the case,

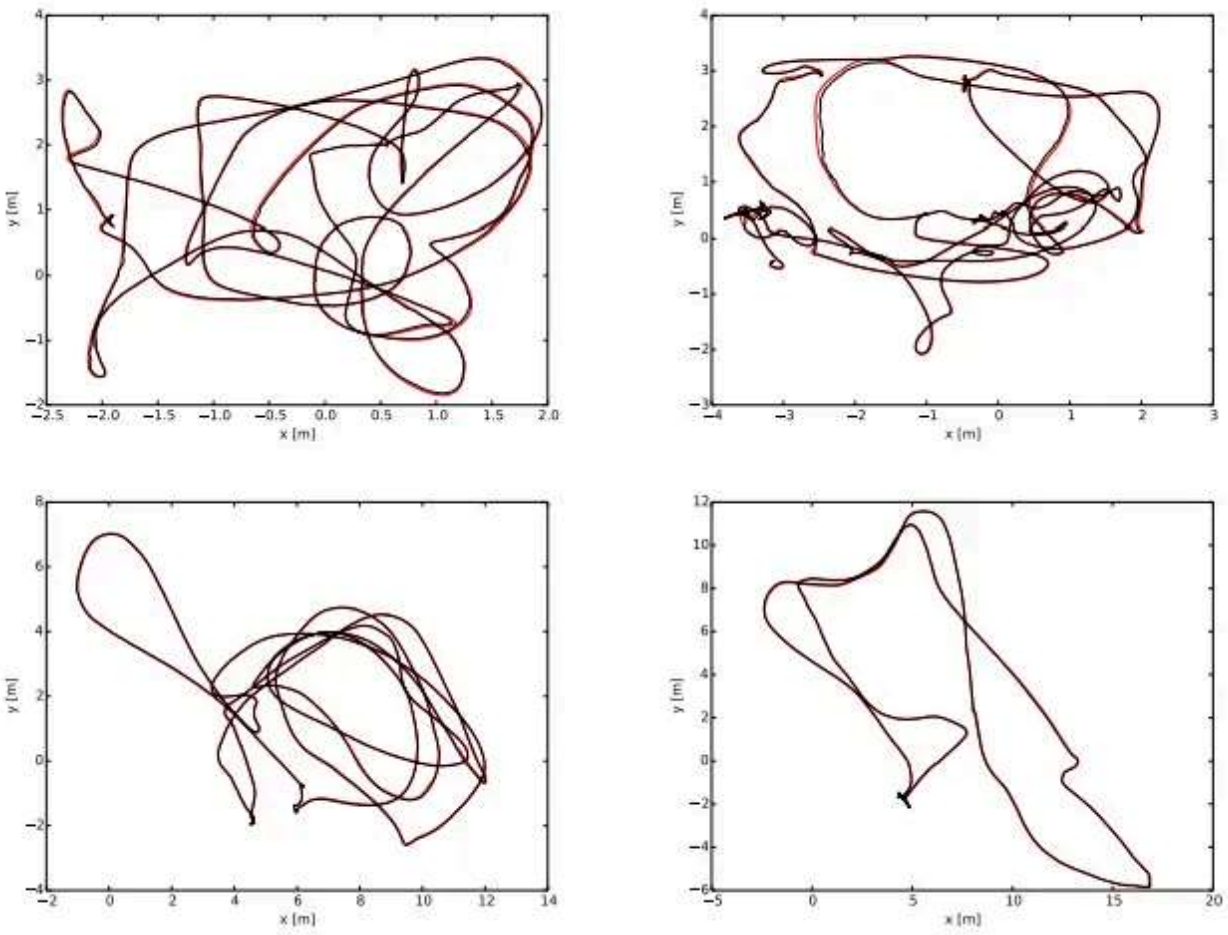
then time for a new Keyframe. ORB-SLAM2 follows a policy to make as many keyframes as possible so that it can get better localization and map and also has an option to delete redundant keyframes, if necessary.

Localization

Coming to the last part of the algorithm, III.F discusses the most important aspect in autonomous robotics, Localization. Unlike LSD-SLAM, ORB-SLAM2 shuts down local mapping and loop closing threads and the camera is free to move and localize itself in a given map or surrounding. In this mode of localization, the tracking leverages visual odometry matches and matches to map points. Visual odometry matches are matches between ORB in the current frame and 3D points created in the previous frame from the stereo/depth information. Visual odometry points can produce drift, that's why map points are incorporated too.

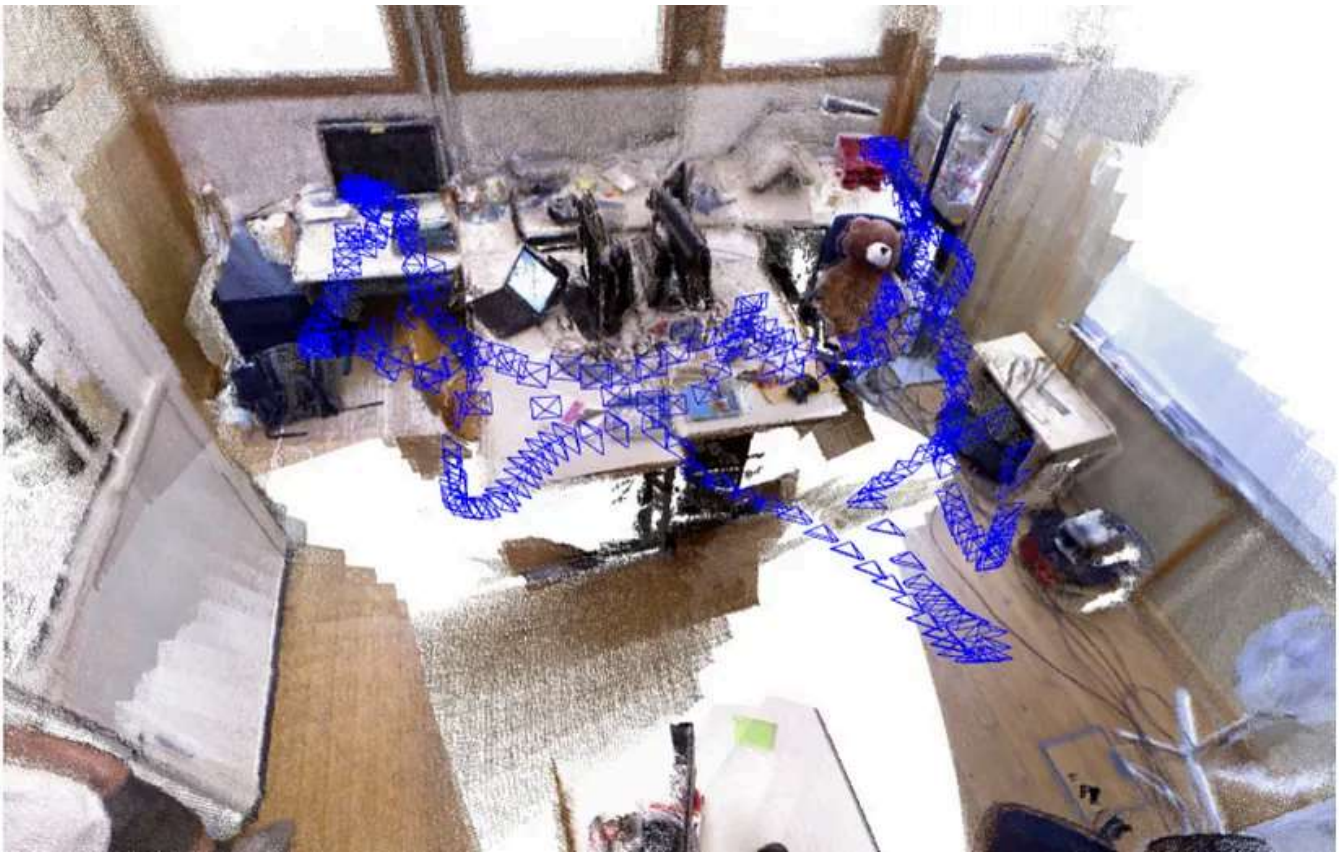
Results

That was pretty much it for how this paper explained the working of ORB-SLAM2. Now comes the evaluation part. Intel Core i7–4790 desktop computer with 16Gb RAM is used for ORB-SLAM2. This algorithm is compared to other state-of-the-art SLAM algorithms (ORB-SLAM (the older one, not ORB-SLAM2), LSD-SLAM, Elastic Fusion, Kintinuous, DVO SLAM & RGB-D SLAM) in 3 popular datasets (KITTI, EuRoC & TUM-RGB-D datasets) and to be honest I'm pretty impressed with the results. Let's see them dataset by dataset.



ORB-SLAM 2 on the TUM-RGB-D dataset. Black is the estimated trajectory, while red is ground truth. Pretty, isn't it?

Firstly the KITTI dataset. According to the authors, ORB-SLAM2 is able to perform all the loop closures except KITTI sequence 9, where the amount of frames in the last isn't enough for ORB-SLAM to perform loop closure. Table 1 shows absolute translation root mean squared error, average relative translation error & average relative rotational error compared between ORB-SLAM2 & LSD-SLAM. Dark numbers indicate low error than its counterpart algorithm and clearly it's ORB-SLAM2 'holding more bold numbers'. In the EuRoC dataset, ORB-SLAM2 beats LSD-SLAM face-on as translation RMSEs are less than half of what LSD-SLAM produces. No words for the TUM-RGB-D dataset, ORB-SLAM2 works like magic in it, see for yourself. ORB-SLAM2 also beats all the popular algorithms single-handedly as evident from table III. Most of the algorithms require high-end GPUs and some of them even require server-client architecture to function properly on certain robots. ORB-SLAM is also a winner in this sphere, as it doesn't even require a GPU and can be operated quite efficiently on CPUs found mostly inside modern laptops.



ORB-SLAM 2 on TUM-RGB-D office dataset

Without any doubt, this paper clearly writes it on paper that ORB-SLAM2 is the best algorithm out there and has proved it. Let's conclude this article with some useful references.

> *This particular blog is dedicated to the original ORB-SLAM2 paper which can be easily found here: https://www.researchgate.net/publication/271823237_ORB-SLAM_a_versatile_and_accurate_monocular_SLAM_system,*
>
> *and a detailed one here: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7219438*

References:

1. ORB-SLAM website: https://webdiis.unizar.es/~raulmur/orbslam/
2. ORB-SLAM paper: https://www.researchgate.net/publication/271823237_ORB-SLAM_a_versatile_and_accurate_monocular_SLAM_system
3. ORB-SLAM in-depth: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7219438
4. ORB-SLAM GitHub repository: https://github.com/raulmur/ORB_SLAM2

5. Inverse depth parametrization: https://en.wikipedia.org/wiki/Inverse_depth_parametrization
6. Pose graph optimization: https://censi.science/pub/research/2013-mole2d-slides.pdf
7. Bundle Adjustment: https://arxiv.org/pdf/1912.03858.pdf and https://www.coursera.org/lecture/robotics-perception/bundle-adjustment-i-oDj0o
8. ICP (Iterative Closest Point): https://en.wikipedia.org/wiki/Iterative_closest_point
9. Huber loss function: https://en.wikipedia.org/wiki/Huber_loss

Slam | Robotics | Computer Vision | Research | Paper