Vision-based Navigation Project: ORB-SLAM

John Flynn and Parika Goel

Technical University of Munich,Germany

July 2, 2019

I. Introduction

A common challenge in the field of robotics is to reason about the pose of a robot and to construct a map of its surroundings using sensor measurements. This could be an RC car that estimates its position on a race track by integrating its wheel sensors or an undersea vehicle that attempts to map an unexplored shipwreck. However an RC car will quickly accumulate error in its position without a map of the race track and an undersea vehicle can't reconstruct a ship's layout without knowing its current position. The problems of mapping an environment and localizing oneself in it are inherently coupled and the techniques to solve them comprise the field of Simultaneous Localization and Mapping (SLAM).

SLAM is the simultaneous estimation of state (such as position and orientation), and the construction of a map of the environment (such as a 3D point cloud) [2]. A key feature of SLAM is that it attempts to maintain global consistency in its map, whereby revisited places in the real world are also recognized as revisited positions in the map. This is in contrast to odometry which treats sequential observations as a long corridor that never reconnects to itself.

In this project we implemented a real-time stereo-camera SLAM algorithm to demonstrate the key components contained in many simple vision-based SLAM systems and to illustrate its benefits over visual odometry. We use sparse indirect methods to find feature correspondences and focus on algorithmic simplicity while ensuring real-time operation. We borrow concepts introduced in [3] including a covisibility graph to store spacially-related keyframes and landmarks (see section II.C), a tracking thread to localize the camera, a local mapping thread to maintain well-determined landmarks and a loop closing thread to associate spatially related keyframes separated far apart in time.

Using our custom SLAM implementation we show that it is possible to run sparse indirect SLAM in real time on a commodity computer over a long period of time without decreasing performance. We also show how SLAM improves upon visual odometry in the mapping of a 3D environment. Finally, we provide some visual insight into the effectiveness of our SLAM algorithm.

II. Background

To build a functional SLAM system we need a few image processing techniques to extract features from camera frames and use those features to localize the camera and construct a 3D map. Since our SLAM implementation is a direct extension of visual odometry, we describe the operation of visual odometry in this section and then describe extensions to SLAM in the proceeding sections.

We first introduce a camera calibration technique to ensure our 3D-2D projection model fits well with the chosen camera. Next we explain a robust method to identify good features (known as keypoints) in camera frames. Then we describe a matching scheme to match keypoints in different frames which can be used to triangulate landmarks. Finally, we give an overview of how visual odometry uses keyframes and landmarks to localize the camera.

A. Camera Calibration

To accurately project 3D points from the world to the 2D image plane we define a projection model for our camera and determine the calibration parameters for that model. In this implementation we experimented with the Pinhole Camera Model, Extended Unified Camera Model, Kannala-Brandt Camera Model and Double Sphere Camera Model [6].

To calibrate the camera we load a video of an april grid that contains a checker board of squares and corners, the position of the april grid corners in 3D space and the set of precomputed keypoints associated with the april grid corners. Using our camera model, we then project the 3D positions of the april grid corners to the image plane and find the difference between the projected and loaded keypoints to compute the projection error. Finally, we minimize the projection error over all corners using Ceres to obtain optimal calibration intrinsics for our camera model.

In this implementation we found that the Double Sphere Camera Model yields the smallest projection error after calibration while having nearly equal computational efficiency as the other presented camera models.

B. Keypoints, ORB Descriptors and Keypoint Matching

Keypoints are the coordinates of identifiable features on the image plane and ORB descriptors[5] are 1D "descriptions" of these points. ORB descriptors are especially useful in our application because unlike many other descriptors, they are rotationally invariant. Matching keypoints in different frames allows us to triangulate landmark positions in 3D space.

We detect keypoints using the Shi-Tomasi algorithm[1] in OpenCV. Then for each keypoint we calculate an ORB descriptor using the procedure described in [5]. We can then find keypoint matches between two different camera frames by computing the Hamming distance between all keypoint descriptors in either frame. If two descriptors are close in Hamming space and there are other descriptors that are almost just a close (that is, they are not outliers) then their associated keypoints are considered matches.

C. Keyframes and Landmarks

Landmarks are 3D points in the world frame that compose a map of the environment. Keyframes are camera frames that are used to triangulate landmark positions. Keyframes establish connections between the projections of their associated landmarks and also between the positions of landmarks associated with other related keyframes.

D. Odometry

Visual odometry utilizes the above concepts to estimate the pose of the camera and to build a locally consistent map of the environment. Specifically, visual odometry like the one we extended operates using the following general steps.

1. Tracking: Keypoints are calculated for the current frame. Then all visible landmarks are projected into the current frame and matched to the computed keypoints. If a sufficient number of landmarks are matched with keypoints then the pose of the camera can be estimated using PNP. If not then local mapping is triggered.

2. Local Mapping: A keyframe is created and keypoints between stereo camera frames are matched and triangulated to create new landmarks. If enough landmarks exist then tracking continues.

3. When a keyframe expires (e.g. a sufficient number of newer keyframes were created) its landmarks are no longer used for tracking.

The main drawback to this odometry algorithm is that it cannot utilize connections between temporally distant but similar keyframes. We will introduce loop closing to reuse old observations. Additionally, it only stores keyframes temporally instead of utilizing spatial locality. We will introduce the covisibility graph to establish spatial relationships.

III. System Overview

The developed SLAM system (see Figure 1) works in three parts, namely tracking, local mapping and loop closing. Each part shares a common data structure known as the covisibiltiy graph.

The tracking thread is responsible for localizing the camera with every frame and deciding when to insert a new keyframe.

The local mapping thread processes every new keyframe. It then performs local bundle adjustment, the aim of which is to create an optimal reconstruction in the local neighborhood of the current frame. The unmatched keypoints in the new keyframe are matched with the keypoints in the connected keyframes in covisibility graph to triangulate new landmarks. A culling policy is applied in local mapping to retain only high quality keyframes and landmarks.

The loop closing thread searches for loops with every new keyframe. If a loop is detected, a transformation matrix is computed that gives the drift accumulated in the loop. Using the transformation matrix, all the keyframe poses in the loop are corrected and the duplicated landmarks are fused. Then a pose graph optimization is performed over all the keyframes and connections in the Essential Graph, a sparser subgraph of the covisibility graph maintained in the SLAM system. Finally, global bundle adjustment is performed over all the keyframes and landmarks in the map. We have followed the techniques and processes mentioned in the ORB-SLAM papers [3] and [4] with modifications. Further we will describe all the steps mentioned above in detail.

IV. Covisibility Graph and Essential Graph

In our implementation we introduce the covisibiltiy graph to contain all keyframes. Intuitively, keyframes that share many landmark observations are often spatially close to each other and have highly correlated poses. To utilize this insight, each edge in the graph is weighted by the number of shared landmark observations between two keyframes (with a minimum of 15 for a connection to be made). This allows us to efficiently find similar keyframes in a so-called "neighborhood" which is essential for several operations from landmark projection to pose optimization.

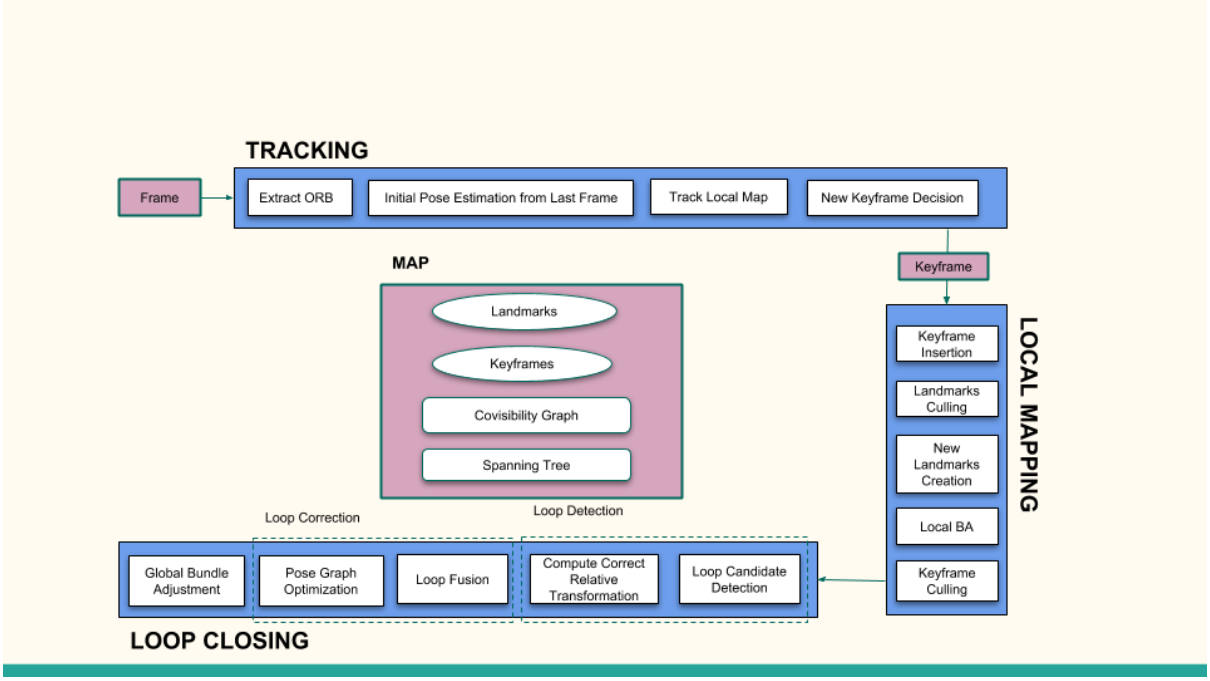


Figure 1: SLAM system overview, showing all the steps performed by the tracking, local mapping and loop closing threads. The main components of the map are also shown.

When a new keyframe is created it aggregates a list of the observing keyframes from all landmarks that it observes. The number of shared landmarks between each keyframe in the list is counted and a new edge is created if there are ≥ 15 shared landmarks. Each keyframe is then notified of the creation of this edge and the connection is completed.

The covisibility graph provides efficient access to the neighborhood of a given keyframe. We save a list of neighbors within each keyframe and provide accessor methods to get these neighbors and to get the landmarks of all its neighbors.

A subset of the connections in the covisibiltiy graph are used to construct an essential graph. The essential graph consists of the spanning tree of the covisibility graph (see Figure 2), loop closure edges and strongly connected edges (≥ 100 shared landmarks). It is intended to contain only the most important connections between keyframes so that optimization over keyframe and landmark positions can be done much more efficiently, while still being almost as accurate as an optimization over the entire covisibility graph.

V. Tracking

The tracking thread of our SLAM system performs the localization of the camera with every new frame. The decision to insert a new keyframe is also made by this thread. In this section, we describe in detail the steps of the tracking thread that are performed for every frame.

A. Finding Keypoint-Landmark Matches

In this step, matches between the detected keypoints in the current frame and the landmarks in the current map are found. First the landmarks in the local neighborhood are found

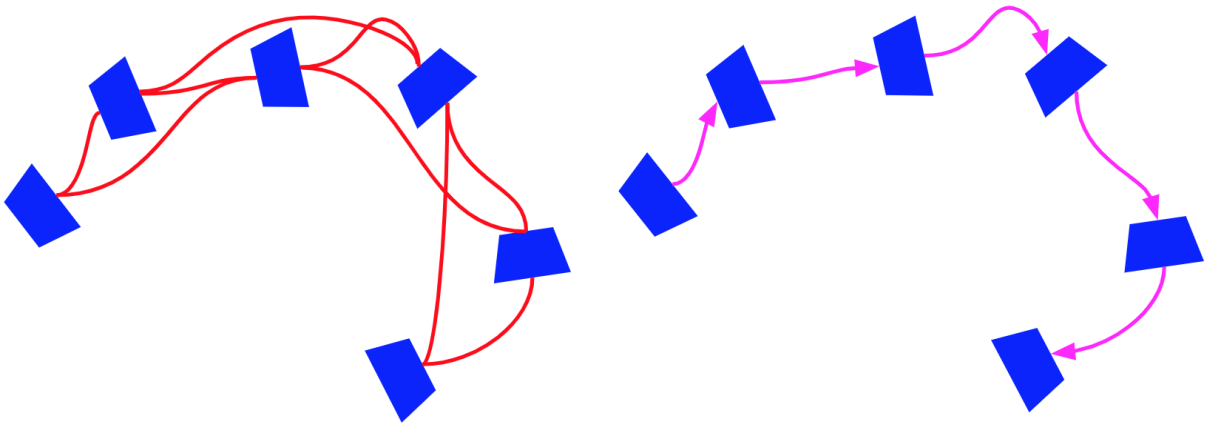


Figure 2: Illustration of the Covisibility Graph (left) and its spanning tree (right). The Essential Graph consists primarily of the spanning tree in addition to loop closure edges and edges with strong connections. Notice that while the spanning tree is sparser, it still connects all keyframes.

by querying the covisibility graph. These landmarks are then projected into the current frame. Then matches between these projected landmarks and the detected keypoints are found using the method described in section II.B, where each landmark descriptor is the descriptor among all its observers that minimizes the Hamming distance to the detected keypoint descriptor.

B. Camera Localization

In this step, the camera pose of the current frame is optimized using a set of keypoint-landmark matches. We create an Absolute Pose Sac Problem in a RANSAC scheme using opengv to find inlier keypoint-landmark matches and an estimate of the camera pose. Next we optimize the camera pose using this set of inliers and then filter the list of inliers using this optimized pose. This reduces the likelihood that outlier keypoint-landmark matches were involved in the calculation of the camera pose.

C. Keyframe Creation

The last step in the tracking thread is to decide whether or not the current frame should be a keyframe. The decision to create keyframe is based on the fact that if the SLAM system is not able to find enough keypoint-landmark matches for the current frame (i.e there are too few inlier matches), then current frame is taken as a keyframe. If there are too few inlier matches from the camera localization step and if the local mapping process is not already active then tracking signals the creation of a new keyframe and triggers local mapping.

VI. Local Mapping

In this section, we describe the steps performed by local mapping thread for every new keyframe K_c.

A. Keyframe Insertion

In this step, local mapping thread first updates the covisibility graph by adding new node for the new keyframe K_c and the edges are updated as per the shared landmarks with other

keyframes. Then it adds the keyframe into the map.

B. Landmarks Culling

We use a point culling policy to make sure that the landmarks which do not provide a lot of information about the environment are removed from the system which makes the system more robust. At every new keyframe, local mapping thread checks all the landmarks in the current map and removes the ones which are observed by less than or equal to 2 keyframes. This policy makes our map contain very few outliers.

C. Creation of New Landmarks

New landmarks are created by matching the keypoints of the current keyframe stereo pair and triangulating the ones for which landmarks do not exist. All the keypoint matches that do not fulfill the epipolar constraint are discarded. We use relative_pose and triangulation APIs of opengv libraries to triangulate new landmarks. The newly triangulated landmarks are then projected onto all connected (neighboring) keyframes K_n in the covisiblity graph. Projected landmarks are matched with the keypoints of the connected keyframes K_n and those keyframes are added as observations to each matched landmark. Each keyframe's connections in the covisibility graph are then updated to reflect the addition of new landmarks (e.g. the creation of new edges or strengthening existing ones).

D. Local Bundle Adjustment

In the local mapping thread, local bundle adjustment is performed to optimize the current keyframe K_c, all the connected keyframes K_n in the covisibility graph and all the landmarks that are observed by all of the above mentioned local keyframes. The keyframes that also observe these landmarks but are not the neighboring keyframes K_n of the current keyframe are also added in the local BA but are kept fixed. We then use Ceres to minimize the reprojection error between the projected landmarks and the detected keypoints in each image.

E. Keyframe Culling

Local mapping thread detects redundant local keyframes and removes them from the system. A keyframe is considered to be redundant if 90% of the landmarks observed by that keyframe are also observed by at least three other keyframes. This policy enables the SLAM system to have a lifelong operation as the number of keyframes within a given physical area will not grow unbound. This helps limit the number of residuals created when running bundle adjustment.

VII. Loop Closing

The third thread in our SLAM system is loop closing. The loop closing thread processes the last keyframe K_c created by the local mapping thread. This thread tries to detect if there are any loops between K_c and older keyframes and then close them. The steps taken by loop closing thread are discussed in detail below.

A. Detecting Loop Candidates

In this step, the loop closing thread tries to detect a keyframe K_l that is a suitable candidate to close a loop. To find a loop candidate, we follow the steps below:

1. Consider the list of all keyframes called loop candidates. There must be at least 10 keyframes and at least 10 keyframes must have passed since the last loop detection.

2. Neighboring keyframes and neighbors of neighboring keyframes are removed from the list of loop candidates.

3. Keyframes that are not within a threshold euclidean radius of current frame are removed from the list of loop candidates.

4. Keypoint descriptors of each of the remaining keyframes are matched with the keypoint descriptors of the current keyframe. Keyframes with less inlier matches than a specified threshold are discarded.

The result is a list of loop closure candidates.

B. Computing the Transformation Matrix

In our SLAM system the map can drift in six degrees of freedom: three translations and three rotations. Scale drift is not a concern because we explicitly model the stereo camera in all optimizations. That is, the transform between the left and right cameras is always fixed to a constant value.

To close the loop, we need to estimate the true transformation matrix from current keyframe K_c to loop keyframe K_l. This will give us the corrected pose for K_c and inform us about the amount of drift accumulated in the loop.

For each loop candidate keyframe K_l we get its landmark observations, transformed into the reference frame of K_l. Then we unproject the keypoints in the current keyframe K_c and create an Absolute Pose Sac Problem in a RANSAC scheme with the unprojected keypoints as bearing vectors and the transformed landmarks as points. This gives us an estimate of the transformation $\boldsymbol{T}_{lc}$ from the candidate keyframe to the current keyframe and a set of inlier keypoint-landmark matches. Candidate keyframes with the number of inlier matches below a specified threshold are discarded and the candidate keyframe with the most inlier matches is accepted as a loop closure.

C. Loop Fusion

From the last step, we have a loop keyframe K_l and corresponding transformation matrix $\boldsymbol{T}_{lc}$. Now we need to correct the loop. The first step in loop correction is to propagate the transformation matrix through the covisible keyframes. We first correct the pose of the current keyframe $\boldsymbol{T}_{wc}$ using the pose of the loop keyframe $\boldsymbol{T}_{wl}$ and the transformation matrix $\boldsymbol{T}_{lc}$ such that $\boldsymbol{T}'_{wc} = \boldsymbol{T}_{wl}\boldsymbol{T}_{lc}$. Then we correct the pose of all the covisible keyframes using the old pose of the neighboring keyframe $\boldsymbol{T}_{neighbor,wc}$, old and corrected pose of current keyframe $\boldsymbol{T}_{wc}$ and $\boldsymbol{T}'_{wc}$, that is $\boldsymbol{T}'_{neighbor,wc} = \boldsymbol{T}'_{wc}\boldsymbol{T}_{wc}^{-1}\boldsymbol{T}_{neighbor,wc}$.

Next duplicated landmarks are fused and new edges are added in the covisibility graph that will close the loop. For each keypoint-landmark inlier match from the computation of the transformation matrix $\boldsymbol{T}_{lc}$, if the keypoint (feature) id matches landmarks observed by both K_c and K_l then those landmarks are fused. Both keyframes are then instructed to update their weights and edges in the covisibility graph by recounting their landmark observations. Since K_c and K_l share fused landmarks, a new edge in the graph will always be created between them.

D. Pose Graph Optimization

Once a loop has been closed it is desirable to distribute the drift throughout the loop. This is sometimes possible with Global BA but the use of reprojection error as residuals in the optimization often results in local minima that are far from a global minimum. An alternative idea is to only create residuals on the transformations between pairs of connected keyframes and then update landmark positions once the optimization is complete. This is a computationally expensive operation if used on the full covisibility graph but the essential graph provides a sparse yet descriptive subset of the covisible connections.

We create one residual for each edge $e_{i,j} = log_{SE(3)}(\boldsymbol{T}_{ji}\boldsymbol{T}_{wi}^{-1}\boldsymbol{T}_{wj})$ in the essential graph. Here $\boldsymbol{T}_{ji}$ is the transform between keyframes K_i and K_j, $\boldsymbol{T}_{wi}$ and $\boldsymbol{T}_{wj}$ are the transforms from keyframes K_i and K_j to the world, respectively and $log_{SE(3)}()$ is the mapping to the Lie algebra. $\boldsymbol{T}_{ji}$ is specified in the cost functor and the optimization is run over $\boldsymbol{T}_{wj}$ and $\boldsymbol{T}_{wi}$. Intuitively, $e_{i,j}$ allows the pose of each keyframe in world coordinates to deviate with a penalty enforced by the transform to each of its neighbors.

To run the optimization we first fix the loop closure keyframe pose K_l. Then we add weight to the residual associated with the loop closure edge so that changing the loop closure transformation is expensive. Residuals are created using the spanning tree edges, strong connections and loop closure edges, taking care not to optimize the same connection twice (for example, an edge can be a strong edge and a spanning tree edge). Once the optimization finishes, we iterate over all landmarks, obtain the corrected transformation from one of their observers (our implementation uses the first one in the list) and update each landmark's position.

E. Global Bundle Adjustment

This is the final step in the loop closing thread. To make our reconstruction more optimal, global BA is performed over the all keyframes and landmarks. While pose graph optimization corrects much of the drift created in the loop, global BA finds a slightly more optimal solution by accounting for the reprojection error. This works well because pose graph optimization initializes global BA closer to a globally optimal solution so that global BA is less likely to fall into a poor local minima.

Parameter blocks are added for all the keyframes and the landmarks and the initial keyframe pose is fixed. In addition, the transform between the stereo camera frames is fixed to prevent scale drift. We use Ceres library to perform the bundle adjustment (same as done for local BA in local mapping thread).

VIII. Results

A. Execution on the EuRoC V1 Dataset

We tested our implementation on the EuRoC V1 Dataset. Figure 3 shows keyframes, the covisibility graph, spanning tree and a loop closure edge for a loop in the dataset. In the left image, the CV graph explains the covisible neighborhood quite well as each keyframe shares a red edge with other keyframes that face approximately the same direction, therefore observing similar landmarks. In the center image the spanning tree connects the keyframes temporally. This is also expected as each keyframe in a simple loop is likely to share the most landmark observations with its adjacent keyframes. In the third image we note that the loop closure edge (yellow) connects distant parts of the spanning tree. Intuitively, this is what we should expect

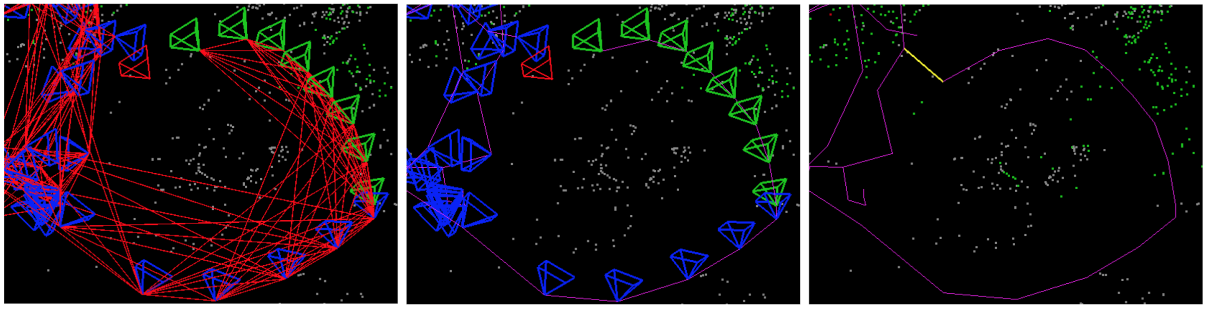


Figure 3: Visualization of the covisibility graph and keyframes before loop closing (left), the spanning tree and keyframes before loop closing (center) and the spanning tree with a loop closure edge (right).

since the spanning tree roughly traces keyframes through time and connects their strongest edges. In general, the larger the loop in a spanning tree, the more drift can be corrected by the inclusion of a loop edge.

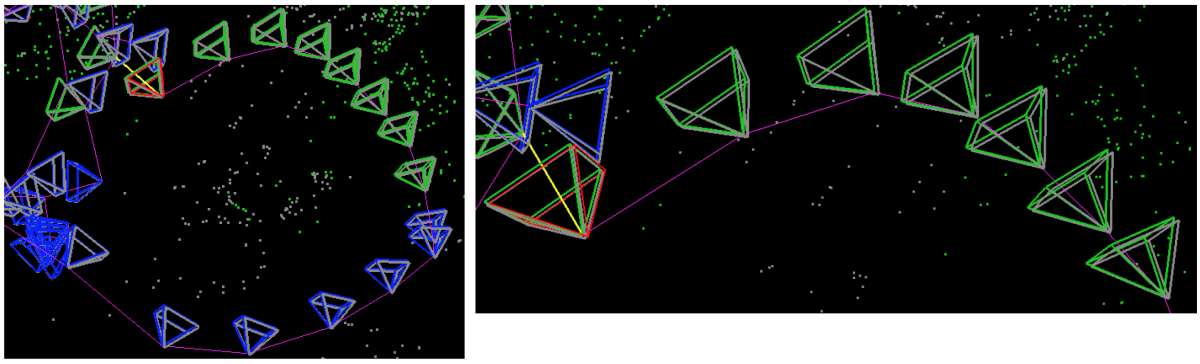


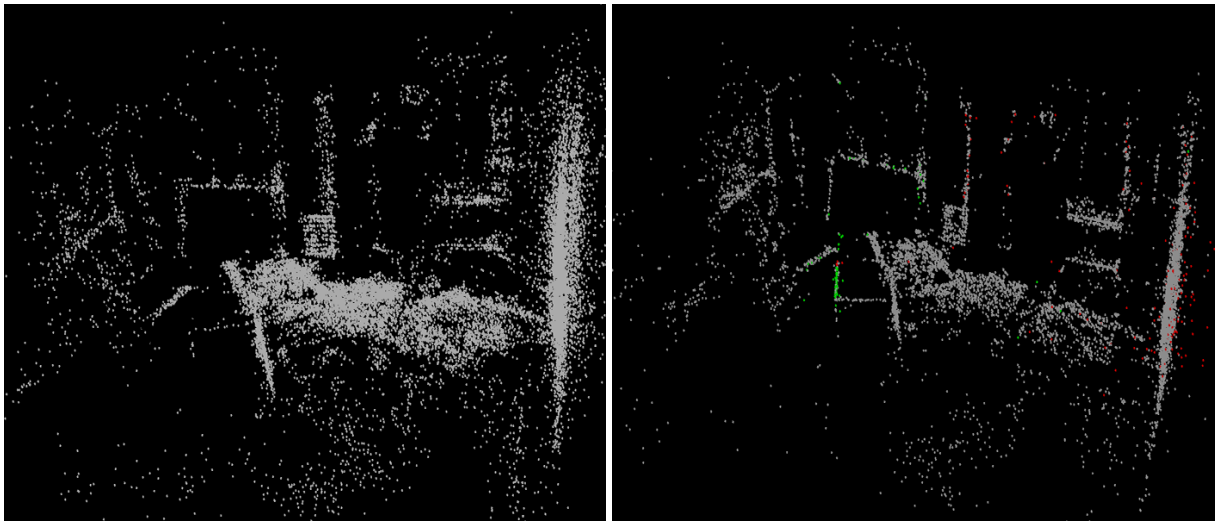
Figure 4: Visualization of a loop closure before optimization (gray) and after (blue and green). Left is a view of the entire loop and right is a close-up of the top of the loop.

Figure 4 illustrates a loop closure before and after optimization. While the drift is minimal since the loop is small, it is still clear that the drift is spread out over all keyframes. The keyframes closer to the current keyframe have a slightly larger correction which we should expect since longer deviations from the loop closure keyframe naturally result in the accumulation of larger drift. The current keyframe is only mildly corrected as the residual for its associated loop closure edge is weighted strongly in the pose graph optimization to reflect our certainty in the loop transformation matrix $\boldsymbol{T}_{lc}$.

B. Comparison with Visual Odometry

Figure 5a and figure 5b illustrate the difference between visual odometry and our approach to visual SLAM. Two main observations can be made about these maps: First, the point cloud for SLAM is much sparser than for odometry. Second, the edges in the SLAM point cloud are much more pronounced with fewer "free floating" points.

The reduction in the number of points for visual SLAM is due to the culling of low-quality keyframes and landmarks. Unlike odometry, our SLAM algorithm forms spatial associations



(a) Point cloud generated using Odometry (b) Point cloud generated using SLAM

between keyframes (and therefore landmarks) using the CV graph which allows poorly-explained landmarks (that is, landmarks with few keyframe observers) to be removed. The result is that only well-explained landmarks (that is, with many observers) remain in the map.

While the culling of landmarks also explains the sharpness of edges (since these are the locations of high quality landmarks) another factor is global consistency enforced by loop closing. In odometry, as the camera explores the room it generates redundant landmarks in places in previously explored in the past. Loop closing however finds connections between keyframes that are distant in time, corrects for drift and fuses redundant landmarks. This means that error is not accumulated the longer the camera moves through the room and future landmark observations refine the point clouds that describe the environment.

IX. Future Work

While our SLAM implementation works well on simple datasets and illustrates the functional components of SLAM quite well, there are still many improvements that could make it faster and more robust. Here we discuss a few next steps to improve upon our system.

A. Multithreading

Most SLAM systems have many operations that do not directly depend each other and are capable of running in paralell. In this implementation the tracking, local mapping and loop closing threads are all designed to run on separate threads and communicate with each other using mutexes and joins. However, designing a multithreaded system by which data (such as the covisibility graph) and operations (such as the creation of a new keyframe) are made thread safe is difficult. For example, to execute a Global BA the covisibility graph must be copied, optimized on a different thread and then stitched back into the original covisibility graph on the main thread.

At the moment, we run these processes on a single thread to verify the correctness of our implementation. However future versions of our program could focus on a consistent thread-safe program architecture to run these processes independently.

B. Bag of Words

Sometimes it is desired to search for a keyframe that is similar to another keyframe. For example, loop closing searches for a loop closure keyframe K_l that shares many observations with the current keyframe K_c. To find a matching keyframe K_l one must search the list of all keyframes and compare each one. Our implementation uses a radius search around the current keyframe K_c as a simplifying heuristic. While this nicely illustrates the mechanisms of loop closing, it is still computationally expensive and isn't guaranteed to find the best match.

A faster approach to this problem is to introduce a Bag of Words (BoW) model. In BoW each keyframe is converted into a vector of visual words where each word is a descriptor of a patch of the keyframe image. Then whenever a similar keyframe K_l to the current keyframe K_c is desired, the keyframe K_c is simply compiled into a BoW representation, queried in the BoW model and a list of keyframes ordered by the number of shared words is returned. K_l can then be chosen from that list. The benefits of this system are that querying the BoW model is fast and the similarity metric is based directly on the visual content of each keyframe image.

C. Global Relocalization

In some cases it is possible that when tracking is lost there are not enough remaining landmarks to estimate the current pose of the camera. Without an estimate of the camera pose it is difficult to accurately triangulate new landmarks. Our implementation makes the naive assumption that the camera simply isn't moving and continuously triggers the local mapping thread to create more keyframes until enough landmarks are triangulated.

An alternative solution to this problem is to search all keyframes for one that might observe similar features as the current camera frame. These keyframes could be found using Bag of Words or a brute force search whereby a keyframe with many similar keypoint descriptors is selected. Then a camera pose could be computed by finding keypoint-landmark matches between the current frame and the landmarks of the similar keyframe.

X. Conclusion

In this project we created a functional visual SLAM system to illustrate the operation of foundational components used in many SLAM systems and to demonstrate its benefits over visual odometry. We modified an existing visual odometry algorithm using ideas borrowed from [3] including the covisibility graph and the separation of tracking, local mapping and loop closure threads to create our SLAM system.

Our two greatest tasks in this project were the creation and integration of the covisibility graph and the design of a loop closure thread that didn't use a Bag of Words model. We constructed a CV graph capable of inferring spatial locality between keyframes and used this to run many different algorithms more efficiently in a visual neighborhood. For loop closing we devised our own method to find loop closure candidates and implemented two powerful optimization methods to correct the drift: Pose Graph Optimization and Global Bundle Adjustment.

Finally, we found through experimentation and visual inspection that our SLAM system operates intuitively and outperforms visual odometry. Our covisibility graph and essential graph link keyframes in a predictable way and provide an efficient platform for optimization. Point

clouds generated by our system look sharper and sparser than those from visual odometry. Our SLAM system operates as expected on simple datasets, serves as a good learning tool and provides an excellent basis for a practical SLAM application.

Bibliography

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

[2] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. *CoRR*, abs/1606.05830, 2016.

[3] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *CoRR*, abs/1502.00956, 2015.

[4] Raul Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *CoRR*, abs/1610.06475, 2016.

[5] 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, Nov 2011.

[6] Vladyslav C. Usenko, Nikolaus Demmel, and Daniel Cremers. The double sphere camera model. *CoRR*, abs/1807.08957, 2018.