Contents

List of Acronyms

BA Bundle Adjustment

BRIEF Binary Robust Independent Elementary Features

DLT Direct Linear Transformation

FAST Features from Accelerated Segment Test

IMU Inertial Measurement Unit

ORB Oriented Fast and Rotated Brief

PTAM Parallel Tracking and Mapping

RANSAC Random Sample Consensus

SIFT Scale Invariant Feature Transform

SLAM Simultaneous Localization and Mapping

SURF Speeded-Up Robust Features

SVD Singular Value Decomposition

UML Unified Modeling Language

1 ORB SLAM Overview

Oriented Fast and Rotated Brief (ORB) Simultaneous Localization and Mapping (SLAM) is a visual SLAM approach which derives its name from the ORB feature descriptor. The ORB feature descriptor was developed by Rublee et al. [1] as "an efficient alternative to Scale Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF)". According to Mur-Artal et al. these features provide "good invariance to changes in viewpoint and illumination" [2] and are cheap to compute. As is obvious from the use of ORB features, ORB SLAM uses a feature-based front-end. The back-end works on a keyframe-based[1] graph-optimization procedure.

Mur-artal et al. utilize different techniques of other approaches like Parallel Tracking and Mapping (PTAM)'s parellelization of mapping and localization with the difference that ORB SLAM uses three parallel threads instead of two as in PTAM. An overview over the algorithm's structure and information flow is given in figure 1.1.

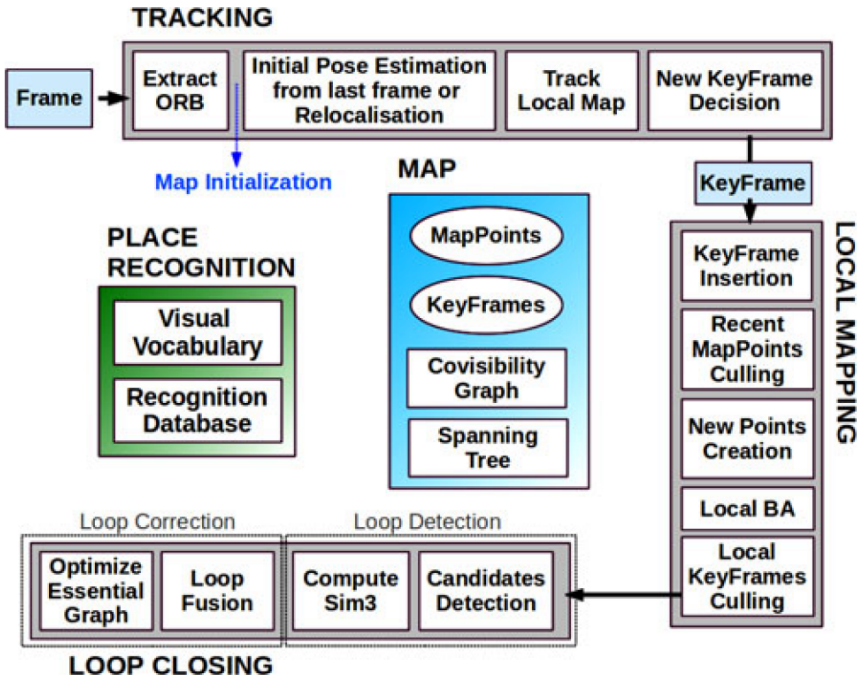


Figure 1.1: Overview of the ORB SLAM process. The white boxes inside a larger grey box are tasks which share a thread. Figure source: [2]

The first step that has to be taken for ORB SLAM to run is to initialize a map. This task is handled automatically and is described in [2]. A map consists of *map points* and *keyframes*. Each map point represents a 3D point in world coordinates and stores information about all the keyframes from which it was observed. Keyframes store information about the

[1] Instead of using every single camera image, keyframe-based approaches only use such images which minimize informational redundancy. By using such "key" frames the computational costs can be greatly reduced.

camera pose, the camera configuration parameters and the extracted ORB features for this frame. Keyframes and map points are used to build a *covisibility graph*. In this graph each keyframe is a node and the nodes are connected through an edge if they share at least 15 map point observations. Additionally a *spanning tree* is created that, starting with the initial keyframe, connects only those keyframes which share the most map point observations. The last graph introduced is the *essential graph*. This graph is a reduced version of the covisibility graph which holds the same nodes but only those edges which have a high covisibility and loop closing edges. This allows for a faster optimization of the global map without much accuracy loss. An overview over all graphs used is given in figure 1.2.

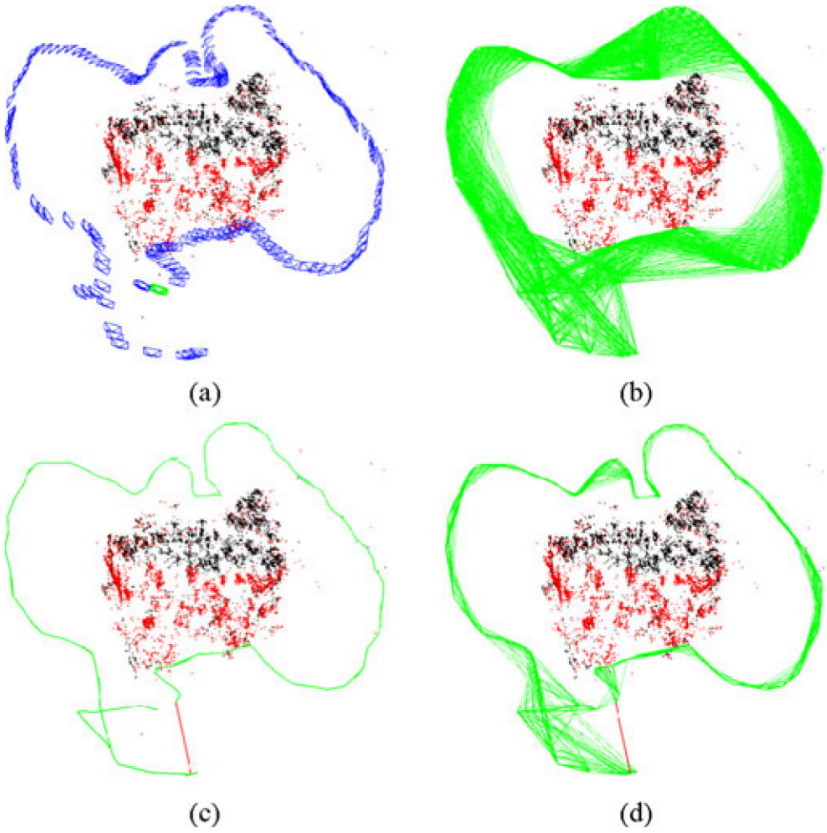


Figure 1.2: An overview over ORB SLAM's graphs. (a) shows keyframes in blue, current camera pose in green and map points in black and red (red are local map points). (b) shows the covisibility graph, (c) the spanning tree and (d) the essential graph. Figure source: [2]

Figure 1.1 also shows the place recognition module which is used for relocalization after tracking loss and loop closures. This module saves representations of seen locations in a database for later matching.

As can be seen in figure 1.1, ORB SLAM works with three parallel threads. The tracking thread is responsible for tracking the movement of the camera. In a first step this thread extracts the ORB features. Then these features are matched with those of the previous

frame. If this works, a constant velocity model is used to estimate the new camera pose. If it fails, a relocalization based on the place recognition database is initialized. Once the initial estimate has been retrieved, the local map is searched for correspondences between image features and map points. The local map consists of keyframes K_1 which share map points with the current frame and those which neighbor K_1 keyframes in the covisibility graph. Doing this the estimated pose can be optimized with minimal computational cost [2]. The algorithm then decides whether the current frame is used as a new keyframe.

The local mapping thread keeps the local map up to date by inserting new keyframes into the covisibility graph and spanning tree. It also tracks whether new map points are being tracked in following keyframes. If not these are removed. By applying a local Bundle Adjustment (BA)[2] the current keyframe, all those keyframes connected to it and all map points seen by those connected keyframes are optimized. In order to keep the computation costs low redundant keyframes are gradually removed from the map.

Once local mapping is done with a keyframe it is passed on to the loop closing thread. Using this keyframe a search for loop closing candidates is started in the place recognition database. These candidates are then evaluated and if the loop is accepted it is incorporated in the covisibility graph. The last step is the optimization of the essential graph.

An interesting extension to ORB SLAM has been proposed in [3]. In this implementation Inertial Measurement Unit (IMU) data is used as an additional information source which was shown to improve the accuracy of the overall process.

[2] An explanation for this technique can be found at 2.7.

2 ORB SLAM In-depth

The following chapter explains the ORB SLAM algorithm. While the basics of it were already presented in 1 this section focuses on a more in-depth explanation of the algorithm's modules and the techniques that are used within them.

ORB SLAM mainly consists of seven separate modules:

1. Feature extraction
2. Data association
3. Initialization
4. Tracking
5. Relocalization
6. Local mapping
7. Loop closing

In order to make it easier to understand how these modules work together and to get a visual overview over ORB SLAM's code, a Unified Modeling Language (UML)-like diagram was created. It can be viewed at A.2. A legend explaining the diagram is given at A.1.

2.1 Feature extraction

The figure at A.3 shows the part of A.2 which is responsible for extracting the image features. As is visible there the first thing ORB SLAM does is it creates an image pyramid. When creating an image pyramid the original image is blurred and subsampled and the resulting image is added as a layer on top of the original image. By doing this a given number of times it results in a pyramid like construct with the original image at the bottom and layers of increasingly smaller images on top of it. Figure 2.1 shows such a pyramid with four layers and a scale factor of 2.

Doing this allows to run a feature extraction on different scale levels. Due to the nature of image features like Features from Accelerated Segment Test (FAST), a feature extraction might miss out on significant features when only looking at one level of scale. Using an image pyramid is a way of making sure that this does not happen. This is an approach that was also taken by Klein et al. [5] in their PTAM algorithm. Instead of using only four levels with a scale factor of 2 like in PTAM, Mur-Artal et al. chose to use eight levels with a scale factor of 1.2 [6].

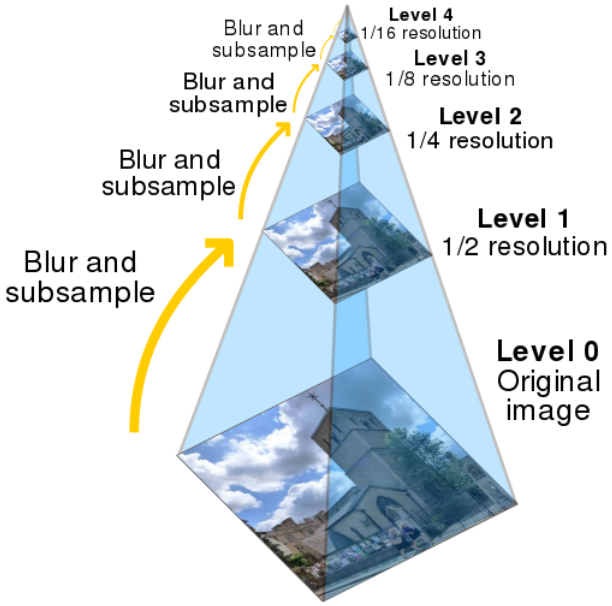


Figure 2.1: An image pyramid with four levels and a scale factor of 2. Figure source: [4]

As a second step ORB SLAM then computes FAST corners on every pyramid level. FAST is a corner detection which was first published by Rosten and Drummond in 2006 [7]. ORB SLAM does not use the machine learned approach described in [7] but uses the OpenCV implementation which is shown in figure 2.2.

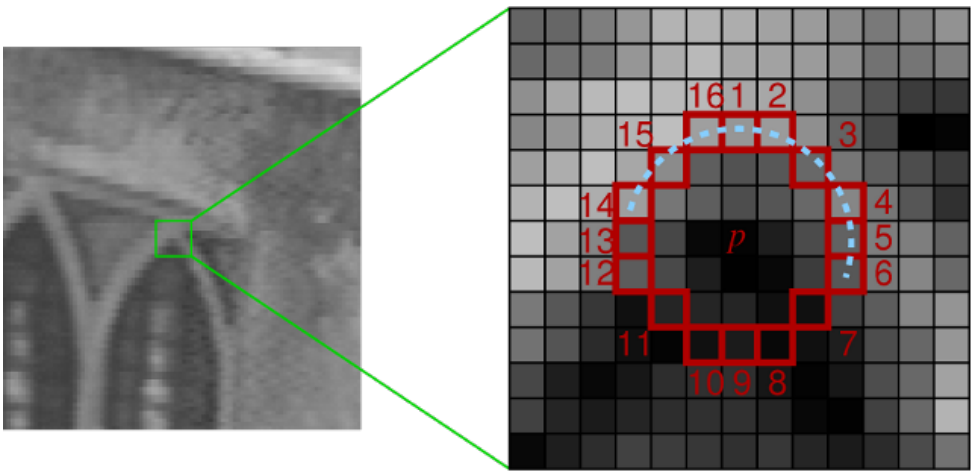


Figure 2.2: Principle of FAST corner detection. For each pixel p in an image a circle with a radius of three pixels around it is considered. A corner is detected as such when at least nine contiguous pixels in this circle are either brighter or darker than p plus a threshold t. By summing up the absolute intensity difference between p and the 16 pixels around it and then keeping only the one with the highest value, duplicate detection of the same corners can be avoided. Figure source: [7]

The advantage of FAST is, that in comparison to other corner detectors like e.g. Harris it is, as its name suggests, a lot faster. However, it is not very robust to noise. A comparison of different methods and their speed is given in [7].

ORB SLAM divides every image in the image pyramid into cells of 30 by 30 pixels and tries to compute FAST corners for each of them. If it does not succeed (cannot detect a single corner) with the initial threshold $t_{init} = 20$ it tries again with a lower threshold $t_{min} = 7$. This way the algorithm makes sure to treat different image regions differently. It then proceeds to distribute the computed corners over each image in the image pyramid. This is based on a maximum desired amount of features n_f which can be configured. To achieve an equal distribution the algorithm will, given it found any corners, divide the image into four equally large cells and then divide these cells again into four equally large subcells. This is repeated until all remaining cells either contain exactly one corner or the amount of cells n_c is equal to n_f. If the latter is the case it will only keep the corner with the highest score for each cell that contains more than one. The rest is simply discarded.

After the detection of FAST corners the next step is to calculate the Oriented Fast and Rotated Brief (ORB) feature descriptor. As its name indicates it is an extension and combination of FAST and Binary Robust Independent Elementary Features (BRIEF). The idea for this descriptor was proposed by Rublee et al. in 2011 [1]. In its original version FAST did not contain any orientation component. ORB adds this by using so-called *intensity centroids* [8]. When defining the moments of an image patch as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \tag{2.1}$$

m_{pq} Image moment
$I(x, y)$ Intensity at position (x, y)

these moments can be used to compute a *centroid* C:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{2.2}$$

The orientation of the corner can then be calculated by:

$$\theta = atan2(m_{01}, m_{10}) \tag{2.3}$$

As mentioned before ORB also improves on the BRIEF descriptor. The BRIEF descriptor was designed by Calonder et al. in 2010 [9]. It is binary string descriptor which allows for comparison using the Hamming distance[3]. This allows for much faster feature comparison than using the L_2 norm as done in other approaches.

[3] The Hamming distance between two strings which are equally long is equal to the amount of corresponding symbols in which they differ. For binary strings this equals the number of ones in the result of a XOR operation on the given strings.

The BRIEF descriptor is defined as a vector of n binary tests:

$$f_n(p) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(p; x_i, y_i) \tag{2.4}$$

f_n feature descriptor
p an image patch of size S x S pixels

with the binary test $\tau(p; x_i, y_i)$:

$$\tau(p; x_i, y_i) = \begin{cases} 1 & if\ p(x) < p(y) \\ 0 & otherwise \end{cases} \tag{2.5}$$

$p(x)$ intensity of a pixel in a smoothed version of p at $x = (u, v)^T$

Figure 2.3 shows an example of how the test points are chosen.

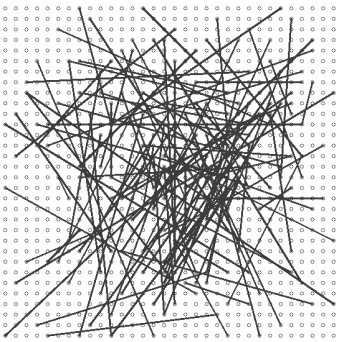


Figure 2.3: A pattern resulting from the way the test points (x_i, y_i) are chosen for the BRIEF feature descriptor. Figure source: [9].

This results in four things which have to be chosen when using BRIEF descriptors:

1. the length of the binary vector n,
2. the patch size S,
3. the spatial arrangement of the chosen test points (x_i, y_i) and
4. the way of smoothing the image patches.

For ORB Rublee et al. chose $n = 256$ and $S = 31$. They also use an integral image approach for smoothing the image patch.

Even though Calonder et al. were able to show that BRIEF achieves better performance than SIFT or SURF, BRIEF suffers from not being able to handle in plane rotation.

For this reason ORB introduces a rotated BRIEF variant (rBRIEF). rBRIEF uses a optimized pattern for picking the test points which was learned from a large set of points by maximizing variance and minimizing correlation. The pattern is shown in figure 2.4.

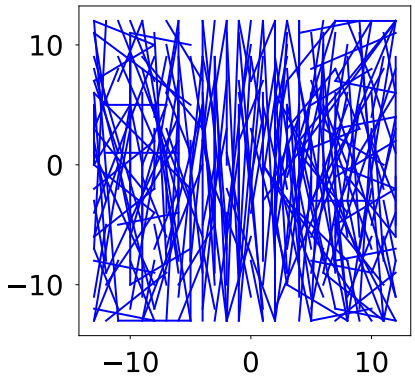


Figure 2.4: A plot of the rBRIEF pattern used by ORB and ORB SLAM.

An exact explanation of how this pattern was created can be found in [9]. This pattern is then rotated to match the orientation which was previously recovered from the FAST corner which allows for ORB to be resistant to rotational changes.

After extracting all features ORB SLAM proceeds to undistort them using the camera calibration parameters the user needs to provide. So ORB SLAM does not actually ever undistort the whole image but only the features after detecting them.

Figure 2.5 shows an exemplary result of ORB SLAM's feature extraction.

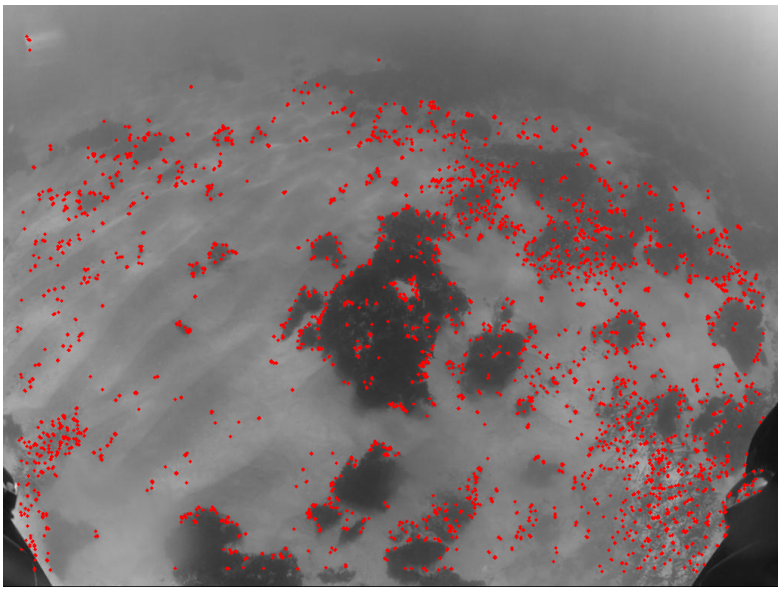


Figure 2.5: ORB SLAM's feature extraction run on an image captured with the BlueROV2. The red dots resemble found features using $n_f = 4000$.

2.2 Data Association

Unlike the other modules feature matching does not happen in one certain place in the code. Different ways of finding correspondence between features in separate frames exist. For initialization for example the 2D features detected in the current frame have to be associated with the 2D features detected in the initial frame. Later when there is a map, 3D map points have to be associated with 2D features in the current frame. In general ORB SLAM uses three different ways of achieving this:

1. search by projection,
2. search by bag of words and
3. search by similarity transformation.

Since which approach is used depends on the individual module, how data association is performed is explained in the section of that module.

2.3 Initialization

Initialization is a large module. An overview over it is given in A.4 and A.5. As is visible in A.4 initialization is only started if there were at least 100 features extracted from the current and the last frame. When the initialization is started the first frame is used as an initial frame against which the following frames will be compared.

In order to find an association between the 2D features seen in the initial frame and the ones seen in the current one, every feature is compared with regard to two properties: its descriptor distance and its orientation. First for each feature in the initial frame all features in a small area (200 x 200 pixels) around this feature in the current frame are collected. In order to avoid having to confirm for every feature whether it lies in the desired area the images are divided into a cell grid with 64 columns and 48 rows. So the features which are recovered from the area around the feature in the initial frame are not actually recovered from an area within 200 x 200 pixels but rather from all cells which lie inside that reach. Then for each feature within that range the descriptor distance to the initial feature is calculated. Two features are considered to be matching with regard to distance when the descriptor distance is less than 50 and the shortest distance found is shorter than 0.9 times the second shortest distance.

If the distance check was ok the orientation of every match is checked. For this all distance matches' orientations are sorted into bins of 12 degrees size. If a match's orientation should not be within the three most common bins it is not considered a match anymore.

Initialization only proceeds if at least 100 matches were found amongst all detected features. If so, the algorithm computes a homography H and a fundamental matrix F in parallel. A homography relates points seen from different viewpoints as shown in figure 2.6. It fulfills the following equation:

$$x_i' = Hx_i \tag{2.6}$$

$x_i \in \mathbb{R}^3$ Image point in first coordinate frame (in homogeneous coordinates)
$x_i' \in \mathbb{R}^3$ Image point in second coordinate frame (in homogeneous coordinates)

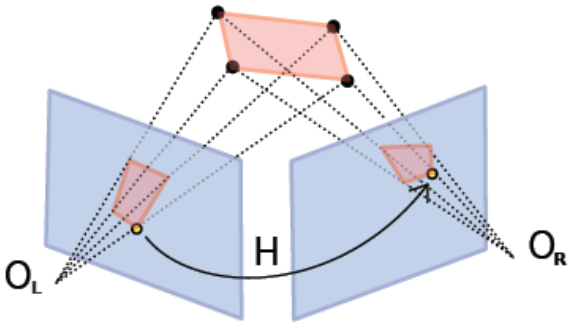


Figure 2.6: A homography relates image points belonging to the projection of commonly seen points in a plane. The homography matrix H directly maps from an image point belonging to the projection in coordinate system O_L to the corresponding image point in coordinate system O_R, if the intrinsic camera parameters are the same. Figure source: [10]

A fundamental matrix also relates corresponding points in stereo images with each other. It does however not rely on the scene being planar. As shown in figure 2.7 the fundamental matrix F describes the epipolar geometry of a scene. More on this topic can be found in [11]. A fundamental matrix F needs to fulfill:

$$x_i'^T F x_i = 0 \tag{2.7}$$

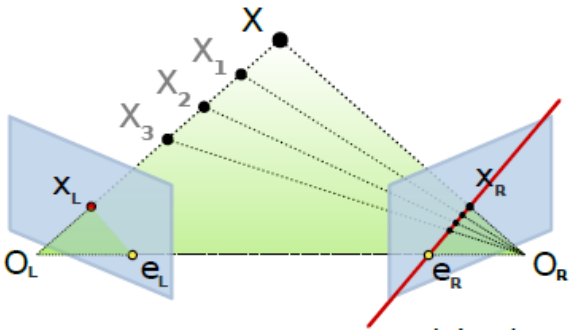


Figure 2.7: The fundamental matrix F constrains where the projections of a commonly seen point has to lie. Given a projection x_L in one image the corresponding point in the other image is constrained to a line Fx_i. Figure source: [10]

Both matrices are estimated in a Random Sample Consensus (RANSAC) algorithm as described in [11] and [2]. By then projecting multiple matches from one frame into the

other using the computed transformations, ORB SLAM calculates a score based on the reprojection error. Whether to use the homography or the fundamental matrix is decided based on a heuristic defined as:

$$R_H = \frac{S_H}{S_H + S_F} \tag{2.8}$$

R_H Decision heuristic
S_H Homography score
S_F Fundamental matrix score

If $R_H > 0.40$ the homography is chosen for pose recovery, otherwise the fundamental matrix. In the case of the homography eight motion hypotheses are recovered from a process explained in [12]. In the fundamental matrix case four hypotheses are reconstructed. How this is done is explained in [11].

In both cases all hypotheses are checked by triangulating all previously determined feature matches and checking whether they lie in front of both cameras, the parallax is big enough and reprojection error is small enough. The triangulation is done using a Direct Linear Transformation (DLT) algorithm as described in [11]. This algorithm needs the two corresponding homogeneous 2D points $x, x' \in \mathbb{R}^3$ and the two camera matrices $P, P' \in \mathbb{R}^{3x4}$. Where:

$$x = PX \tag{2.9}$$
$$x' = P'X \tag{2.10}$$

hold, with $X \in \mathbb{R}^4$ being the corresponding 3D point in homogeneous world coordinates. X can then be determined via:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}, \; x' = \begin{bmatrix} x'_1 \\ x'_2 \\ 1 \end{bmatrix}, \; P = \begin{bmatrix} P_1^T \\ P_2^T \\ P_3^T \end{bmatrix}, \; P' = \begin{bmatrix} P_1'^T \\ P_2'^T \\ P_3'^T \end{bmatrix}, \; A = \begin{bmatrix} x_1 P_3^T - P_1^T \\ x_2 P_3^T - P_2^T \\ x'_1 P_3'^T - P_1'^T \\ x'_2 P_3'^T - P_2'^T \end{bmatrix}$$

$$[U, S, V] = \text{SVD}(A)$$

$$\text{with } U, S, V \in \mathbb{R}^{4x4}, \; V = \begin{bmatrix} V_1^T \\ V_2^T \\ V_3^T \\ V_4^T \end{bmatrix} \text{ and } V_i^T = [v_{i1}, v_{i2}, v_{i3}, v_{i4}]$$

$$X = \frac{1}{v_{44}} V_4 \tag{2.11}$$

Note that the last step in 2.11 has to be performed because the matching points have been normalized before. The minimum parallax is set to one degree and at least 90% of all matches need to be correctly triangulated with a hard cap set to a minimum of 50

points triangulated. There are also a few more constraints depending on whether the homography or the fundamental matrix hypotheses are being used.

In case of using the homography the hypothesis with the second best result (second most matches which are considered to be inliers after checking reprojection error) has to have less than 0.75 times the amount of inliers the best hypothesis has. In the fundamental matrix case no two other hypotheses are allowed to have more than 0.7 times the amount of inliers the best hypothesis has. Using these constraints the initialization module makes sure that the initialization is only accepted when there is a hypothesis which is clearly better than the others and avoids initializing badly.

Once the initialization is accepted an initial map is created using the initial and the current frame as the first two keyframes and all the triangulated matches as initial map points. At this point a first global BA is run on the initial map. If there should not be at least 100 map points in the map after this the initialization will again be discarded and the process repeated.

2.4 Tracking

When the initialization was successful the tracking module is going to be used to estimate the movement of the camera and map the surroundings. This module's code overview is given in A.6 and A.7. As can be seen in A.6 tracking is done in one of two ways depending on whether a motion model exists or not. ORB SLAM uses a constant motion model for its calculations. When tracking was successful in the last frame it will simply assume that the camera will again move the same way for this frame, this allows the algorithm to use a simpler approach for tracking.

Tracking with a motion model:

When using a constant motion model the algorithm can run data association by assuming that the features which were mapped to a map point in the previous image can be found within a certain area in the new image which is determined based on this motion. How large that area is depends on which level of the scale pyramid the feature was extracted from. For a feature found in the base scale it will check an area of 14 by 14 pixels. Should this not bear more than 20 matches with the previous frame it will try again with a window of 28 by 28 pixels. Every feature of the current frame which is accepted during this search will also be associated with the map point the matched feature from the last frame was associated with. If even after the second search less than 15 matches were made the tracking will be retried without using the motion model.

Tracking without a motion model:

Should tracking with a motion model fail or no motion model exist ORB SLAM will take a different approach to tracking. Here instead of projecting features into the current frame, the current frame's features are transformed into a bag of words vector and then compared with features seen in the current keyframe.

For this ORB SLAM uses the DBoW2 library. DBoW2 was developed by Gálves-López and Tardós in 2012. Their approach is described in [13]. In general the bag of (visual) words technique is a way of converting an image in such a way that it is represented by a sparse numerical vector. Using these vectors DBoW2 creates a hierarchical database as depicted in figure 2.8.

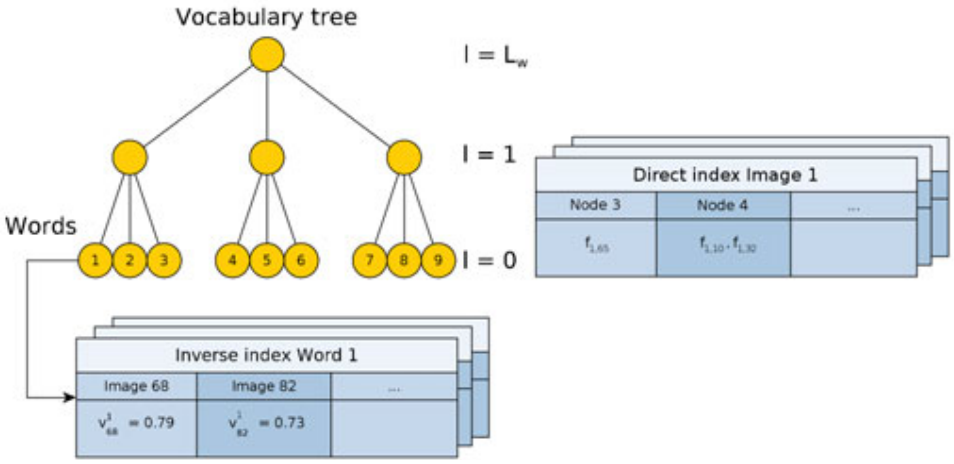


Figure 2.8: A sketch of the database structure formed by DBoW2. The nodes in the tree represent clusters of feature vectors. The root node at level $l = L_w$ holds all stored feature descriptors. To create a new level in the tree the feature descriptors are clustered into k_w clusters using k-median clustering based on their Hamming distance. This is repeated until the whole tree is created. The leaves then represent the W visual words. In DBoW for every word a so called inverse index is saved which is a list of pairs $< I_t, v_t^i >$. I_t being an image in which that word was seen and v_t^i the weight for that word in that image. Furthermore the database also contains a direct index which keeps track of all feature descriptors in a given image. Figure source: [13]

The database described in 2.8 is created in an offline step in advance to the actual algorithm using it. If the training set is sufficiently big the resulting vocabulary can also be used in different scenarios than the training images came from.

When now trying to find images similar to a given query image the feature descriptors extracted from the query image are compared with the tree nodes. For each feature descriptor the tree is traversed from the root to the leaves and a corresponding word is found. The corresponding word is the one for which the Hamming distance between the feature descriptor and the word's centroid is minimal. With this process it is possible to determine which words are present in the query image and how often each word occurs. By then simply creating a bag of words vector $\nu_t \in \mathbb{R}^W$ which holds how often each word was seen in the image, the query image can be compared to other images by using the L_1-score $s(\nu_1, \nu_2)$:

$$s(\nu_1, \nu_2) = 1 - \frac{1}{2} \left| \frac{\nu_1}{|\nu_1|} - \frac{\nu_2}{|\nu_2|} \right| \tag{2.12}$$

For ORB SLAM the database was created with a dataset of roughly 10000 images of both outdoor and indoor scenarios. The database was built with $L_w = 6$ and $k_w = 10$ which results in a total of approximatively one million words [6].

The tracking module can make use of this database by comparing the current frame's features only with those features in the current keyframe that belong to the same node in the database tree at a certain level. For these it then tries to find the feature that has the lowest Hamming distance below a given threshold. Again, when the search results in less than 15 matches tracking will be aborted. If this happens relocalization will be triggered.

Both the tracking approach with motion model and the one without will, given that they were able to find 15 or more matches, use these to optimize the previously assumed pose. This optimization is done using the g^{2}o library [14]. Since after the optimization some matches might not be considered matches anymore, the algorithm will do another check on the amount of matches remaining once the optimization is completed.

If the tracking was successful up onto this point the second part of tracking will be triggered. This part is shown in A.7. In order to keep the processing times minimal ORB SLAM keeps a local map which only contains the set of keyframes K_1 which share map points with the current frame and a set of keyframes K_2 which are direct neighbors of those in K_1 in the covisibility graph. Now that the algorithm has an estimate of the current camera pose as well as some map points tracked in the current frame it will use its local map to further refine the pose estimation. This is done in five steps as described in [2].

For all map points in both K_1 and K_2:

1. Compute the projection into the current frame.
2. Compute the angle between current viewing ray ν and the map point's mean viewing direction n. Discard if $\nu \cdot n < cos(60°)$.
3. Calculate the distance d from map point to camera center. Discard if it is out of the scale invariance region of the map point $d \notin [d_{min}, d_{max}]$.
4. Compute the scale in the frame by d/d_{min}.
5. Compare the representative descriptor D[4] of the map point with the unmatched features in the current frame, at the predicted computed scale and close to its projection in the frame. Associate the map point with the best match.

Once this process was done for all map points in the local map the pose will again be optimized using the newly created matches. If after the optimization less than 30 matches

[4] The associated ORB descriptor whose Hamming distance is minimal with respect to all other associated descriptors from keyframes in which it is observed [2].

between map points and the current image's features remain, tracking will be aborted and relocalization started. If the algorithm has just recently (within the last five frames) relocalized itself the minimum matches necessary are raised to 50.

When tracking succeeds the last decision the tracking module has to make is whether the current frame should become a keyframe. For this to happen four conditions have to be met:

1. The last relocalization must have happened more than 20 frames ago.
2. Local mapping is idling or more than 20 frames have passed since last insertion of a keyframe.
3. The current frame tracks at least 50 points.
4. The current frame tracks less than 90% of the points in the last keyframe.

This point also marks the end of the tracking cycle. Once this part is done the next frame will be loaded and processed.

2.5 Relocalization

When tracking fails completely ORB SLAM falls back to relocalization mode. When in this mode, the algorithm will try to find a keyframe which views the same scene that is visible in the current frame. This also means that no further mapping or localization is done until the algorithm has found a view it already knows. ORB SLAM uses the bag of word approach explained in 2.4 for relocalization. The first step that is taken is to convert the image into the bag of words representation. Then the database is queried to find all keyframes which are similar to the current frame. To find these candidates the following steps are taken:

1. Find all keyframes which share at least one word with the current frame.
2. Find the keyframe that shares the most words w_{max} with the current frame. Discard all keyframes which share less than $0.8 \cdot w_{max}$ with the current frame.
3. Compute the similarity score as in equation 2.12 for all remaining candidates.
4. Find the keyframe with the best score s_{max}. A keyframe's score is calculated by taking its own score and adding to it the score of the 10 keyframes with best covisibility (given they are a part of the candidates as well). Discard all keyframes which have a score lower than $0.75 \cdot s_{max}$.

Should this not yield any candidates the process is stopped and will be retried with the next frame. If there are candidates the algorithm will do a search by using the bag of words approach as done in the case of tracking without a motion model (2.4). Any candidates

that achieve less than 15 matches are directly discarded. For all remaining candidates the algorithm will try to estimate a pose from the matches made. This is done via the EPnP algorithm described in [15]. The EPnP algorithm allows for estimating the pose of a calibrated camera by using n 3D to 2D point correspondences. Using the correspondences made through the previous matching, ORB SLAM runs five iterations with $n = 4$ randomly sampled points and tries to recover a pose from these. A pose is considered to be found when at least 10 points of the previously determined matches demonstrate a small enough reprojection error. If so the algorithm will use the recovered pose and optimize it using all point correspondences. Should there be less than 50 matches after this the algorithm will try to refine the pose up to two times by first finding more matches through projecting map points tracked by the candidate keyframe into the current frame and then optimizing the pose again. The exact way this is done is illustrated in A.8. A candidate keyframe is only accepted for relocalization when at the end of all checks at least 50 matches between the candidate's tracked map points and the current frame's features were found.

2.6 Local Mapping

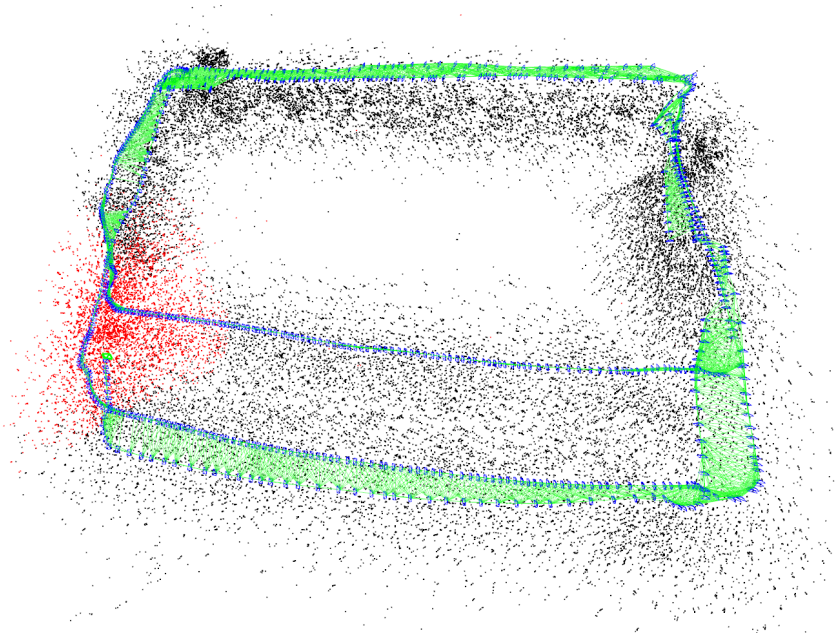


Figure 2.9: A map created and visualized by ORB SLAM. The blue shapes resemble estimated camera poses for keyframes. Green lines connect those keyframes which are connected in the covisibility graph. Black and red dots visualize map points. Map points drawn in red belong to the current local map. The local map is defined by the sets of keyframes K_1 and K_2. K_1 contains all keyframes which share map points with the current frame. K_2 consists of all keyframes which are direct neighbours of a keyframe in K_1 within the covisibility graph.

As described previously ORB SLAM keeps a local map of its current surrounding. The

local mapping module is the part of the algorithm which is responsible for keeping this map up to date by inserting and removing both keyframes and map points and constantly optimizing it. In order to be able to do this permanently this module runs in its own thread.

The general structure of this module is depicted in A.9. The module is run in a loop where it checks every 3 ms whether there are any new keyframes which have to be integrated in the local map. Once a new keyframe arrives the following steps are taken:

1. Integrate the new keyframe into the local map by updating all necessary connections and objects.
2. Check whether map points in the local map can be deleted. A map point will be deleted in two cases:
   (a) Its ratio between how often it should be visible from a frame and how often it was actually matched is lower than 0.25.
   (b) It has been matched less than three times and more than one keyframe has been added since it was first included in the local map.

   If a map point is not deleted within the first two keyframes it will stay in the map and will not be deleted anymore. Note that ORB SLAM does not actually delete map points from memory. It simply marks them as unusable.
3. New map points are created by triangulating feature matches between the new keyframe and the 20 keyframes which share the most map points with the current keyframe. Only feature matches which are not yet related to a certain map point are triangulated.
4. If there are no new keyframes waiting for insertion, search for duplicate map points and fuse them.
5. If there are still no new keyframes for insertion, perform a local BA and afterwards remove such keyframes which share at least 90% of their map points with at least three other keyframes.

Bundle Adjustment (BA) is a technique which is widely used in graph based SLAM algorithms. As described in equation 2.9 the projection from a 3D point X to its corresponding 2D point x in an image can be achieved through the camera matrix P. However in a realistic scenario cameras are subjected to noise so this relation may not always be fulfilled correctly. When now faced with a scenario where a set of cameras with respective camera matrices P^i sees a set of 3D points X_j, as shown in figure 2.10, BA allows to optimize the solution, assuming Gaussian noise.

The goal of BA is to estimate the camera matrices $\hat{P}^i$ and 3D points $\hat{X}_j$ whose corresponding projections $\hat{x}^i_j$ correctly fulfill: $\hat{x}^i_j = \hat{P}^i \hat{X}_j$, while also minimizing the reprojection error between the reprojected point and the measured x^i_j for every frame it is viewed in [11].

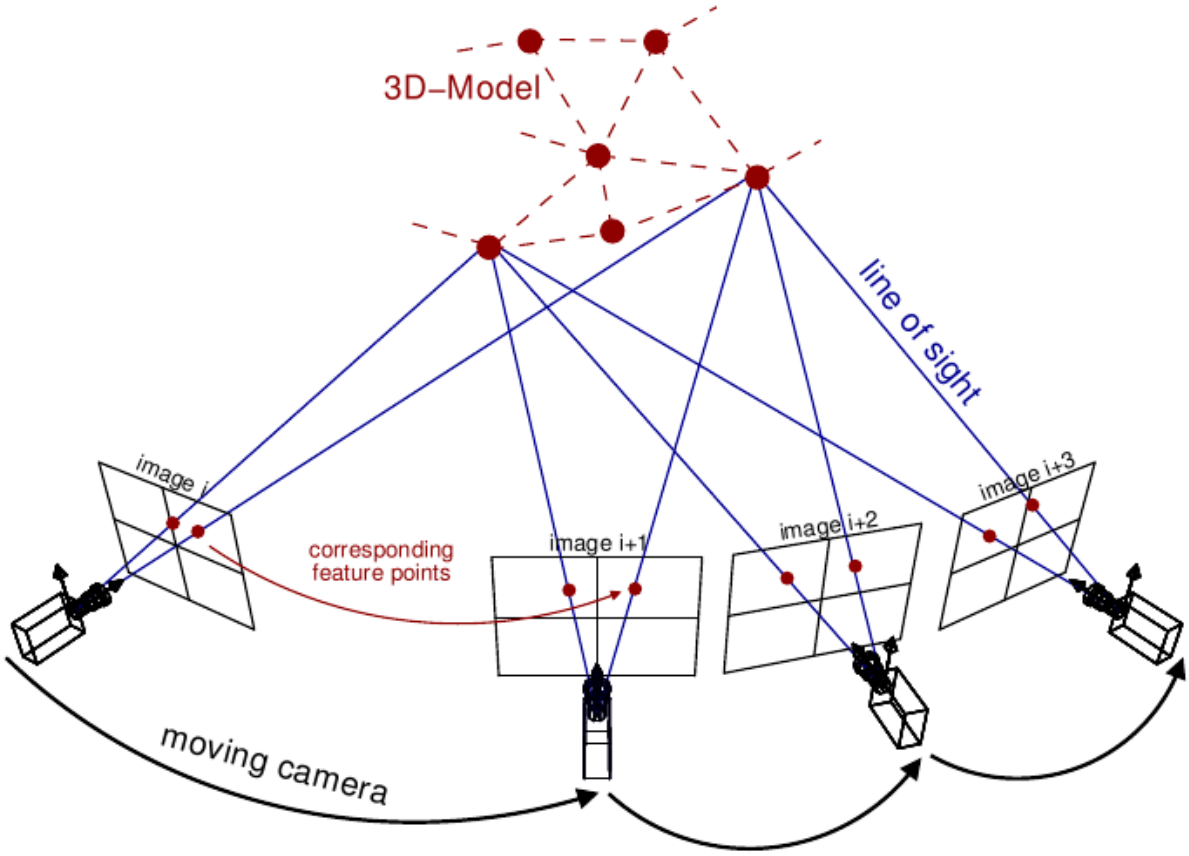


Figure 2.10: Illustration of the situation BA is used in. In theory the lines of sight corresponding with an observation of a 3D point from multiple points of view should cross in that point in 3D space. Given the noise cameras are subject to this is likely not to be the case in a real scenario. Figure source: [16]

This means BA minimizes the reprojection error function:

$$\min_{\hat{P}^i, \hat{X}_j} \sum_{ij} d(\hat{P}^i \hat{X}_j, x^i_j)^2 \tag{2.13}$$

where $d(x, y)$ stands for the image distance between homogeneous points x and y.

Due to the large number of unknowns this can quickly become an unmanageable task. Every camera matrix $\hat{P}^1$ comes with 11 degrees of freedom and every 3D point $\hat{X}_j$ with 3. So for a scenario with m views and n 3D points the algorithm needs to find a solution for $3n + 11m$ parameters.

To account for this problem ORB SLAM uses the g^{2}o library for its graph optimization needs. g^{2}o is able to deal efficiently with large amounts of parameters by restructuring the optimization problem based on its sparse properties. More information on the topic can be found in [14].

2.7 Loop Closing

Much like the local mapping module the loop closing module also runs in its own thread. This is due to the fact that it needs to observe incoming data permanently in order to be able to detect loops. In ORB SLAM loop closing is done in three steps:

1. Detecting a loop (diagram at A.10)
2. Confirming the loop (diagram at A.11)
3. Refining the loop (diagram at A.12)

Loop Detection

In analogy to local mapping ORB SLAM only looks for loops when a new keyframe is available. As is evident in A.9 the loop detection will be notified of a new keyframe as soon as local mapping is done processing it. Once loop closing receives the new keyframe it will start to look for other keyframes connected to it via the covisibility graph. For each of these it will compute a score according to equation 2.12. Given there are at least 10 keyframes in the map and the last loop closure was at least 10 keyframes ago it will pose a query to the keyframe database asking for all keyframes which share at least one word with the current keyframe and have a higher score than the minimum of those just calculated. The keyframes returned by the keyframe database are then used as loop closing *candidates*.

For each of these candidates a *candidate group* is created which contains all keyframes which are connected to this candidate. ORB SLAM then checks these groups for *consistency*. A group is considered to be consistent when it overlaps with a candidate group which was created for the last keyframe that was passed to the loop detection module. Overlapping means that two candidate groups share at least one common keyframe. Once a candidate group has achieved a consistency count of more than two, it is passed on to the second part of the loop closing module. A visualization of the procedure is given in figure 2.11.

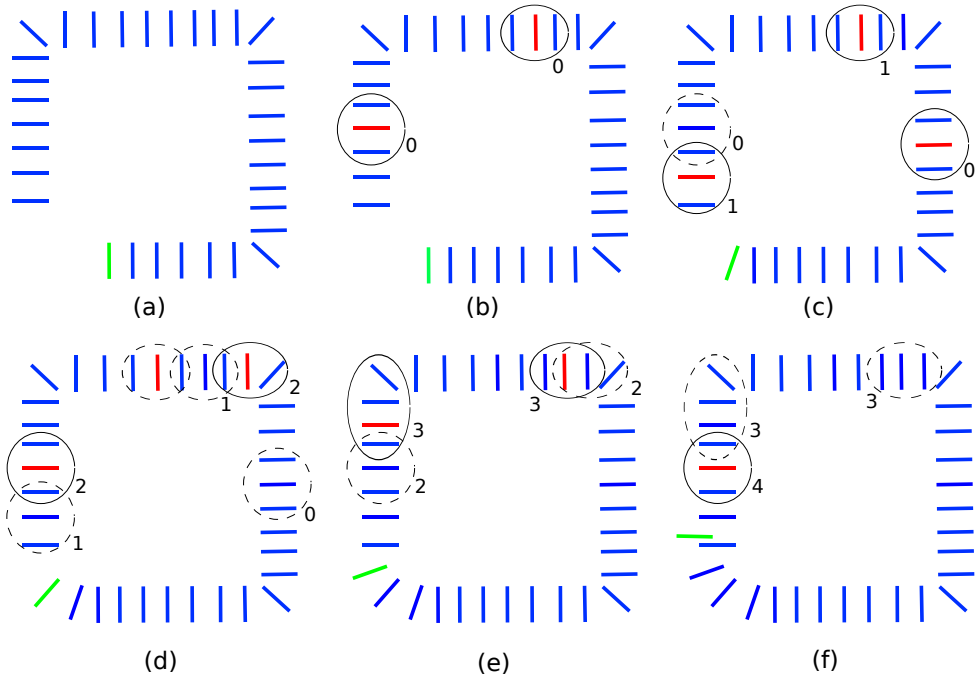


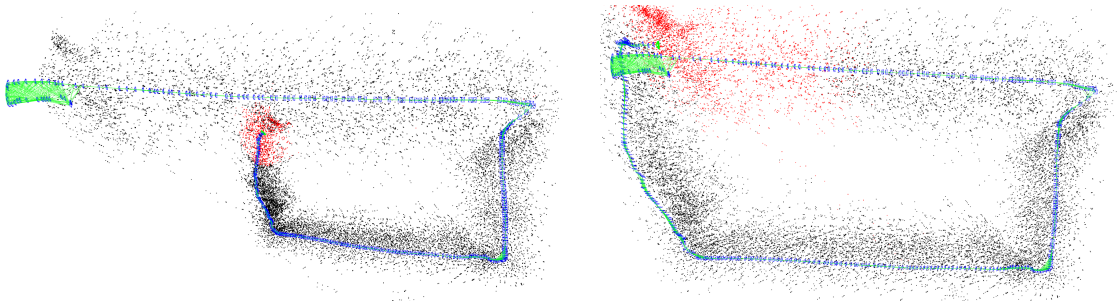
Figure 2.11: All six figures a-f sketch a possible setup of keyframes viewed from above. Blue lines resemble keyframes which are part of the map, the green keyframe is the current keyframe and red keyframes illustrate loop closing candidates. Closed circles represent candidate groups which are going to be saved for the next iteration, the dashed ones are deleted after the current iteration. (a) depicts the initial situation. No keyframes are considered as candidates. (b) shows the creation of two candidate groups and initialization of their consistency counter. In (c) the new group on the left overlaps with the old one and therefore increases its consistency counter. So does the one on top where the candidate keyframe stayed the same. On the right a new candidate group is created. (d) shows that when two new groups overlap the old one (top) only the one tested first goes on to increase its counter. The group on the right is deleted because there was no new group overlapping it. Sketch (e) shows how two groups reach a consistency count of three. At this point they will be used for loop confirmation. (f) illustrates that groups are not deleted once they reach a count of three. If they keep finding overlaps they will be used for confirmation with the next iteration again.

Loop Confirmation

Once a candidate group has accumulated enough consistency its last associated keyframe will be used to evaluate whether it can actually be considered as a loop. In order to check whether the current keyframe and a candidate keyframe correspond, ORB SLAM searches for matches between them in a similar way to what was explained in 2.4 for the case without a motion model. Only that now the algorithm does not look for matches between 2D and 3D points but can match the map points seen in the current keyframe directly to the map points seen in the candidate keyframe by using their saved ORB descriptors.

When successful (at least 20 matches found) ORB SLAM now holds matches between pairs

of map points which correspond to the same 3D point in the real environment. However, internally, due to accumulated drift, these map points will most likely not be very close to each other even though they represent the same real point. Figure 2.12 illustrates this situation.



(a) (b)

Figure 2.12: Result of mapping with ORB SLAM while driving the BlueROV2 in a rectangular path. (a) shows the state of the map created by ORB SLAM just before detecting a loop closure. (b) shows the same map a few frames later than (a), just after a loop has been detected and closed. This figure demonstrates the accumulation of error due to e.g. scale-drift and how loop closing can account for it.

In localization the solution can generally drift with respect to six degrees of freedom. Three rotational degrees of freedom and three translational. For monocular localization however a seventh degree of freedom becomes important as well: scale. Similar to humans that loose their sense of depth when trying to see with only one eye, a SLAM algorithm using a single camera cannot know about the scale of its environment because it doesn't have a reference. Simply put a monocular SLAM algorithm has no idea whether it is looking at life size room or a perfect miniature representation of it. This is why it can only estimate its surrounding up to a factor of scale between internally kept *units* and real distances in meters. Unfortunately this also means that this factor is another degree of freedom which can drift over time, as can be seen in figure 2.12.

For the sake of finding and correcting the drift accumulated over time, ORB SLAM estimates a similarity transformation[5] between the current keyframe and the loop closing candidate keyframe. Such a similarity transformation S can be described as:

$$S = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} \tag{2.14}$$

$S \in Sim(3)$	Similarity transformation
$s \in \mathbb{R}$	Scale factor
$R \in SO(3)$	Rotation matrix
$t \in \mathbb{R}^3$	Translation vector

[5] A matrix which allows conversion between representations of the same mathematical construct (e.g. a transformation matrix) from one base to another.

$Sim(3)$ and $SO(3)$ are so called *Lie groups*, $SO(3)$ represent the group of rotations in 3D space and $Sim(3)$ the group of similarity transformations in 3D space. Every *Lie group* has a related *Lie algebra* which is a space in which the representation is minimal. What this means is easily understood through the following example:

Most of the time rotations are expressed through rotational matrices which are over-parametrized (need nine parameters to describe three degrees of freedom) or as euler angles which suffer from gimble lock or singularities. In $SO(3)$'s corresponding *Lie algebra* $so(3)$ every rotation can be expressed as a linear combination of three so called generators[17]:

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \; G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \; G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.15}$$

such that any rotation can be expressed as:

$$\omega = [\omega_1, \omega_2, \omega_3]^T \in \mathbb{R}^3 \tag{2.16}$$

$$\omega_1 G_1 + \omega_2 G_2 + \omega_3 G_3 \in so(3) \tag{2.17}$$

The same is true for similarity transformations which are then simply a combination of seven generators. An in-depth explanation of the topic is given in [17].

ORB SLAM computes this similarity transformation using the method of Horn described in [18]. The transformation is checked by using it to see how large the error is when projecting the corresponding map points. Should this check produce more than 20 matches the transformation is accepted.

The algorithm then goes on to doing another guided search for more 3D point to 3D point correspondences with the help of the computed similarity transformation and use the found matches to further optimize the transformation. Should at least 20 matches remain after optimization the candidate is accepted for one last test. At this point ORB SLAM performs another search for more matching map points by also projecting map points found in the loop closure candidate's connected keyframes. If after this at least 40 matches were found overall the candidate is accepted and the loop will be closed and refined.

Loop Refinement

At this point the loop has already been accepted. The last step that is done by ORB SLAM is to refine the loop. In order to implement the loop the local mapping module has to be interrupted so the map cannot change while it is being optimized. Then the connections in the covisibility graph are updated such that the current keyframe is connected to those which are at the location around the candidate keyframe. ORB SLAM then proceeds to collect all keyframes which are connected to the current one and uses the computed similarity transformation to correct the pose of all connected keyframes and their map points. This effectively moves all keyframes and map points closely connected with the

current keyframe to where they are going to be after the loop closure. Afterwards the connections in the covisibility graph are updated.

Even though the pose of keyframes and map points around the current keyframe have now been corrected, the problem of correcting all other keyframes in the loop and removing duplicate map points still needs to be solved. For removing duplicate map points the algorithm projects all map points seen by connected keyframes into its image coordinates and tries to match them with features. If it finds a match for a feature that is already linked to a different map point the map point will be considered a duplicate and removed. Otherwise it will be used as a new observation. These matches are the first step of linking the two loop closure sides. Using these the local connections in the covisibility graph are again updated which now also connects keyframes from both sides.

The next step is to propagate the correction over the essential graph (a subgraph of the covisiblity graph, see 1). This way the optimization can focus on the most important edges of the pose-graph and will be much quicker than if it would include all edges kept in the covisibility graph. The optimization is run over Sim(3) constraints as detailed in [2].

As a last step a global bundle adjustment is started in a separate thread which will optimize over the all keyframes and map points while at the same time local mapping is rescheduled to be run again. According to [3] this is a very costly operation which also comes with the problem that the optimized output of the BA has to be merged with the current map once it is done. Since the map keeps on changing while the global BA is running the algorithm has to update all those frames which were not yet part of the BA. This is solved by updating those keyframes which were not updated according to the transformation of their parent keyframe in the spanning tree. Not updated map points will be updated according to the transformation of their reference keyframe.

References

[1] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *Proc. IEEE Int. Conf. Comput. Vis.* (2011), pp. 2564–2571. ISSN: 1550-5499. DOI: 10.1109/ICCV.2011.6126544.

[2] Raul Mur-Artal, J. M M Montiel and Juan D. Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Trans. Robot.* 31.5 (2015), pp. 1147–1163. ISSN: 15523098. DOI: 10.1109/TRO.2015.2463671. arXiv: 1502.00956.

[3] Raul Mur-Artal and Juan D. Tardos. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". In: (2016). ISSN: 15523098. DOI: 10.1109/TRO.2012.2197158. arXiv: 1610.06475.

[4] *Wikipedia: Pyramid(image processing)*. URL: https://goo.gl/dN7RdV (visited on 26/09/2017).

[5] Georg Klein and David Murray. "Parallel tracking and mapping for small AR workspaces". In: *2007 6th IEEE ACM Int. Symp. Mix. Augment. Reality, ISMAR* (2007). DOI: 10.1109/ISMAR.2007.4538852.

[6] Raúl Mur-Artal and Juan D. Tardós. "Fast relocalisation and loop closing in keyframe-based SLAM". In: *Proc. - IEEE Int. Conf. Robot. Autom.* (2014), pp. 846–853. ISSN: 10504729. DOI: 10.1109/ICRA.2014.6906953.

[7] Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection". In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* 3951 LNCS (2006), pp. 430–443. ISSN: 16113349. DOI: 10.1007/11744023_34.

[8] Paul L Rosin. "Measuring Corner Properties". In: *Comput. Vis. Image Underst.* 73.2 (1999), pp. 291–307. ISSN: 1077-3142. DOI: 10.1006/cviu.1998.0719.

[9] Michael Calonder et al. "BRIEF: Binary robust independent elementary features". In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* 6314 LNCS.PART 4 (2010), pp. 778–792. ISSN: 03029743. DOI: 10.1007/978-3-642-15561-1_56.

[10] OpenMVG Authors. *Homography matrix*. URL: http://openmvg.readthedocs.io/en/latest/openMVG/multiview/multiview/ (visited on 29/09/2017).

[11] Andrew Zisserman Richard Hartley. *Multiple View Geometry*. 2003. ISBN: 9780521540513. URL: https://goo.gl/2WmqiC.

[12] O.D. Faugeras and F. Lustman. "Motion and Structure From Motion in a Piecewise Planar Environment". In: *Int. J. Pattern Recognit. Artif. Intell.* 02.03 (1988), pp. 485–508. ISSN: 0218-0014. DOI: 10.1142/S0218001488000285. URL: http://www.worldscientific.com/doi/abs/10.1142/S0218001488000285.

[13] Dorian Galvez-Lopez and Juan D. Tardos. "Bags of Binary Words for Fast Place Recognition in Image Sequences". In: *IEEE Conf. Comput. Vis. Pattern Recognit.* 28.5 (2012), pp. 1188–1197. ISSN: 1552-3098. DOI: 10.1109/TRO.2012.2197158. URL: http://doriangalvez.com/papers/GalvezTRO12.pdf.

[14] Rainer Kümmerle et al. "G2o: A general framework for graph optimization". In: *Proc. - IEEE Int. Conf. Robot. Autom.* June (2011), pp. 3607–3613. ISSN: 10504729. DOI: 10.1109/ICRA.2011.5979949.

[15] Vincent Lepetit, Francesc Moreno-Noguer and Pascal Fua. "EPnP: An accurate O(n) solution to the PnP problem". In: *Int. J. Comput. Vis.* 81.2 (2009), pp. 155–166. ISSN: 09205691. DOI: 10.1007/s11263-008-0152-6.

[16] Chris Sweeney. *Structure from Motion (SfM)*. URL: http://www.theia-sfm.org/sfm.html (visited on 06/10/2017).

[17] Ethan Eade. *Lie Groups for 2D and 3D Transformations*. 2013. URL: http://ethaneade.com/lie.pdf.

[18] Berthold K. P. Horn. "Closed-form solution of absolute orientation using unit quaternions". In: *J. Opt. Soc. Am. A* 4.4 (1987), p. 629. ISSN: 1084-7529. DOI: 10.1364/JOSAA.4.000629. URL: https://www.osapublishing.org/abstract.cfm?URI=josaa-4-4-629.

A Appendix

A.1 ORB SLAM code diagram explanation

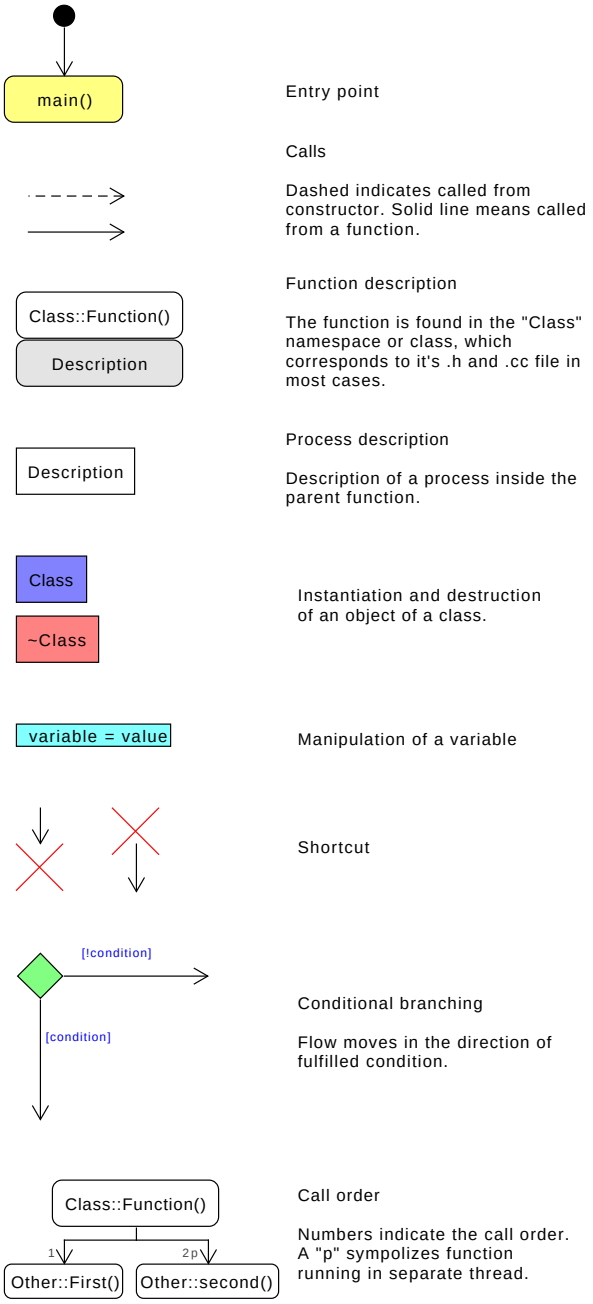


Figure A.1: Legend for diagrams on the following pages.

A.2 ORB SLAM code diagram

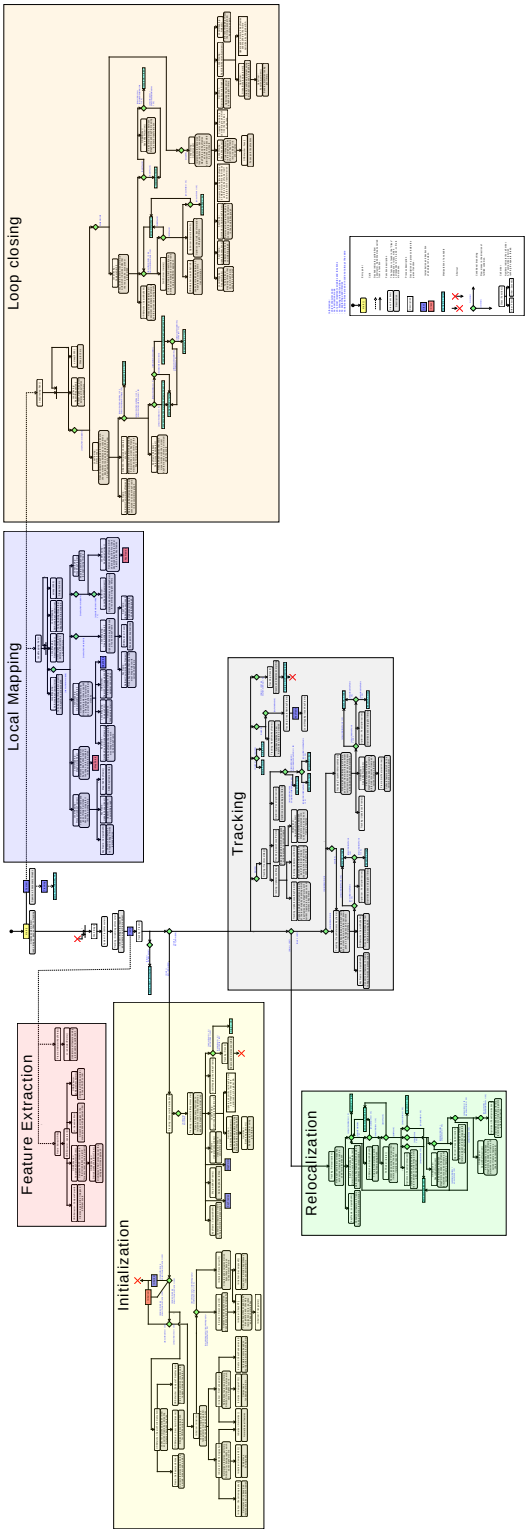


Figure A.2: ORB SLAM code structure diagram. The colored blocks resemble the individual code modules.

For original size visit: `https://raw.githubusercontent.com/kafendt/ORB_SLAM2_Accessible/master/docs/ORB_Code_activity.pdf`

A.3 ORB SLAM feature extraction diagram

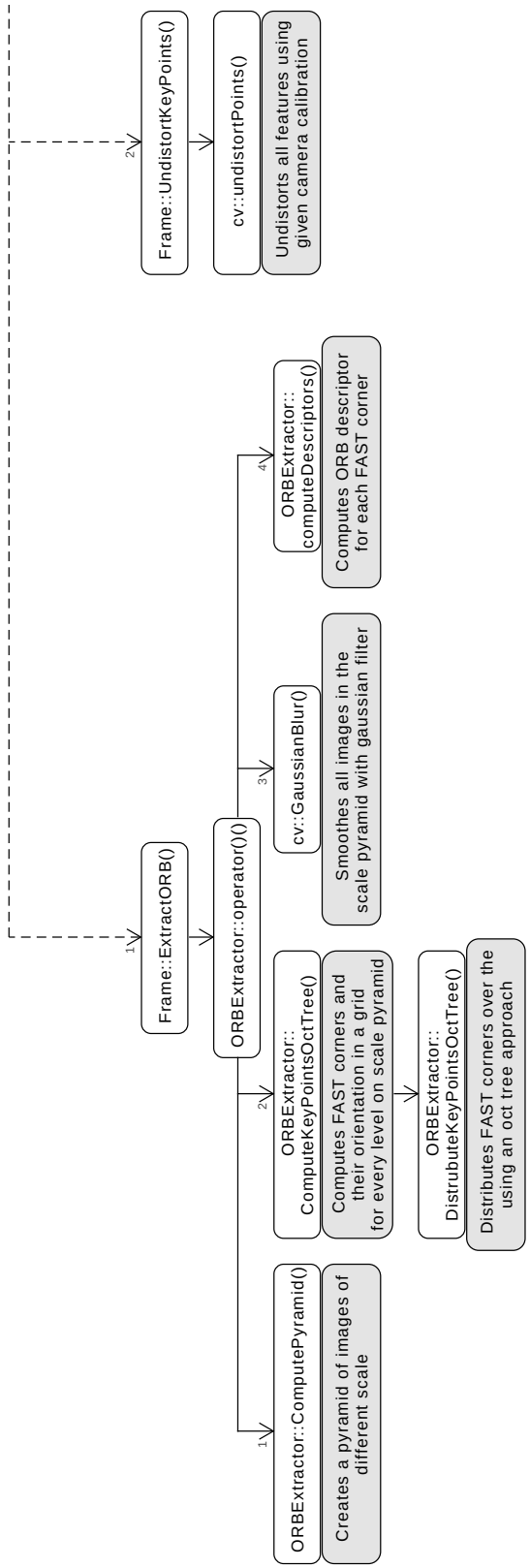


Figure A.3: ORB SLAM feature extraction module diagram

A.4 ORB SLAM initialization diagram (part 1)

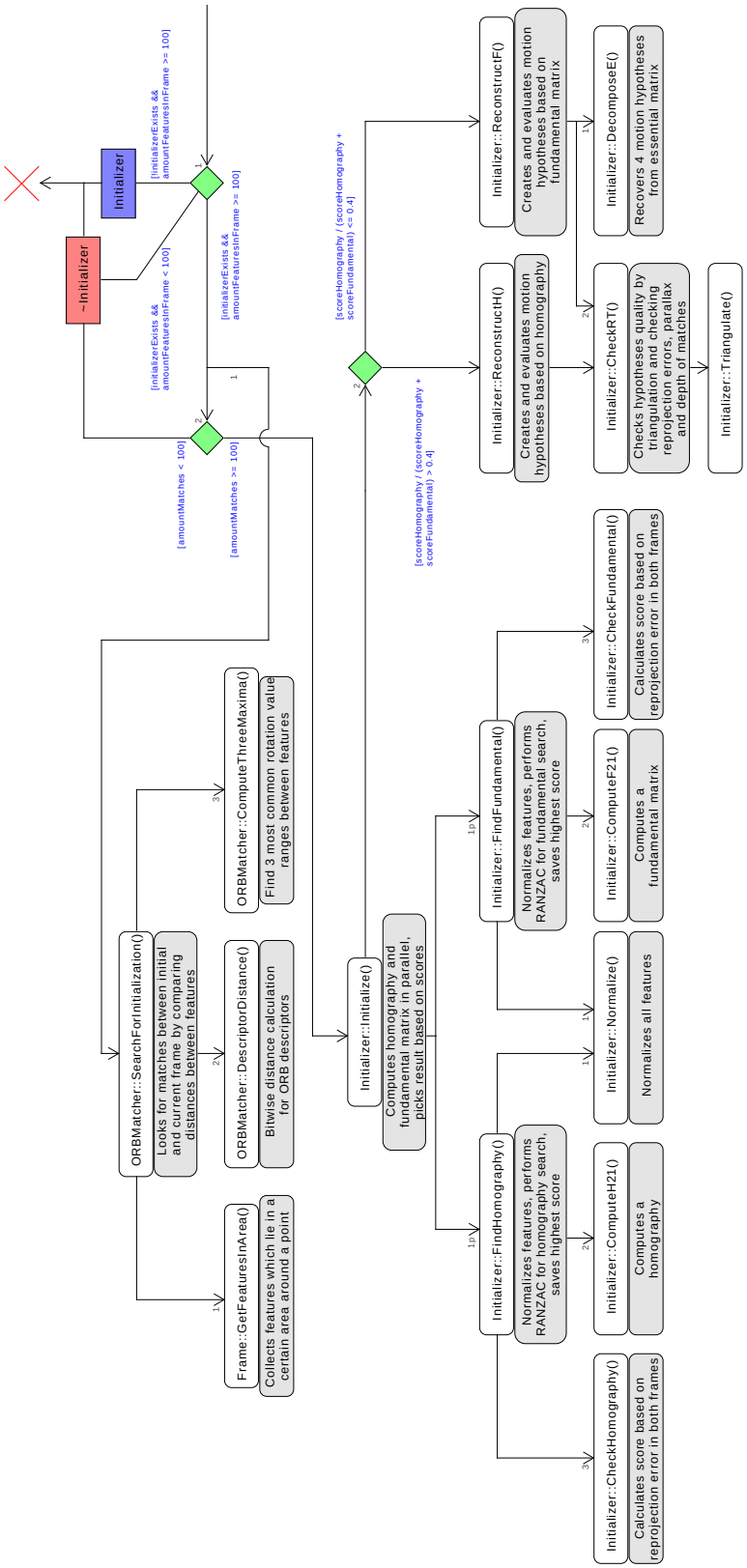


Figure A.4: ORB SLAM initialization module diagram (part 1)

A.5 ORB SLAM initialization diagram (part 2)

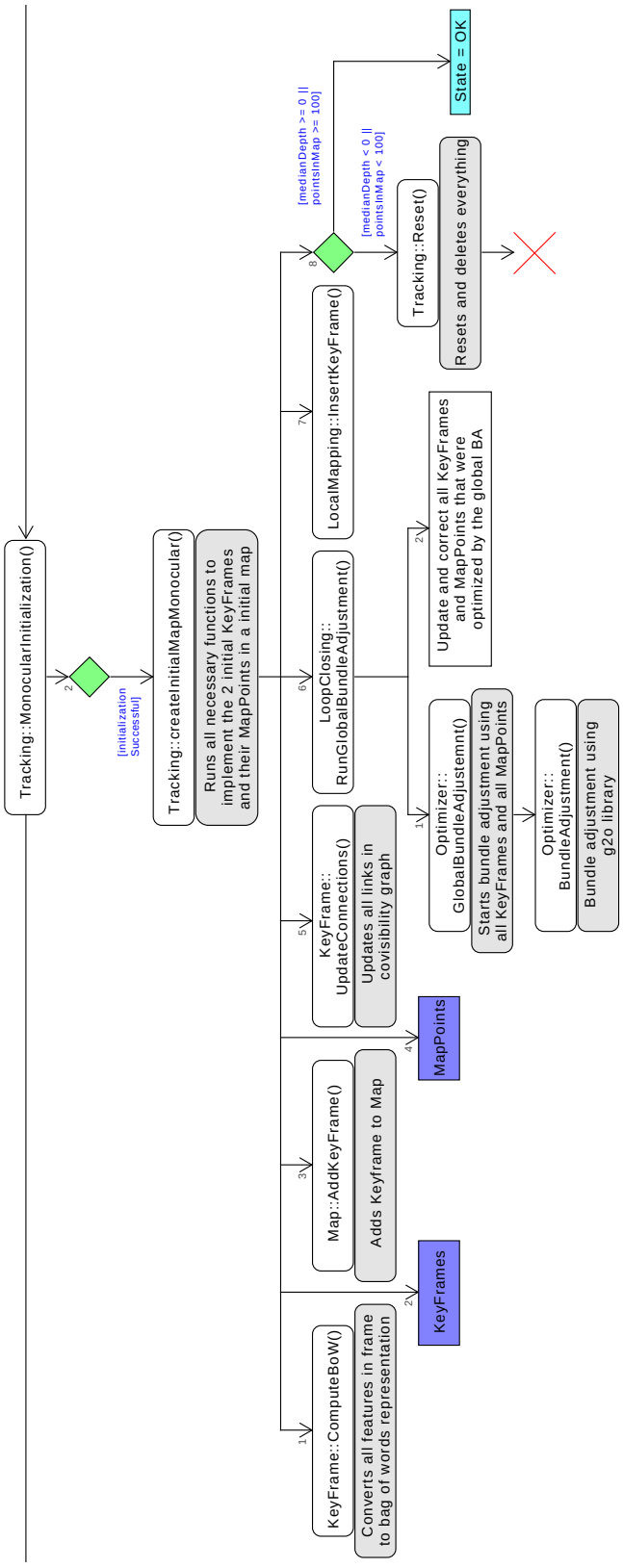


Figure A.5: ORB SLAM initialization module diagram (part 2)

A.6 ORB SLAM tracking diagram (part 1)

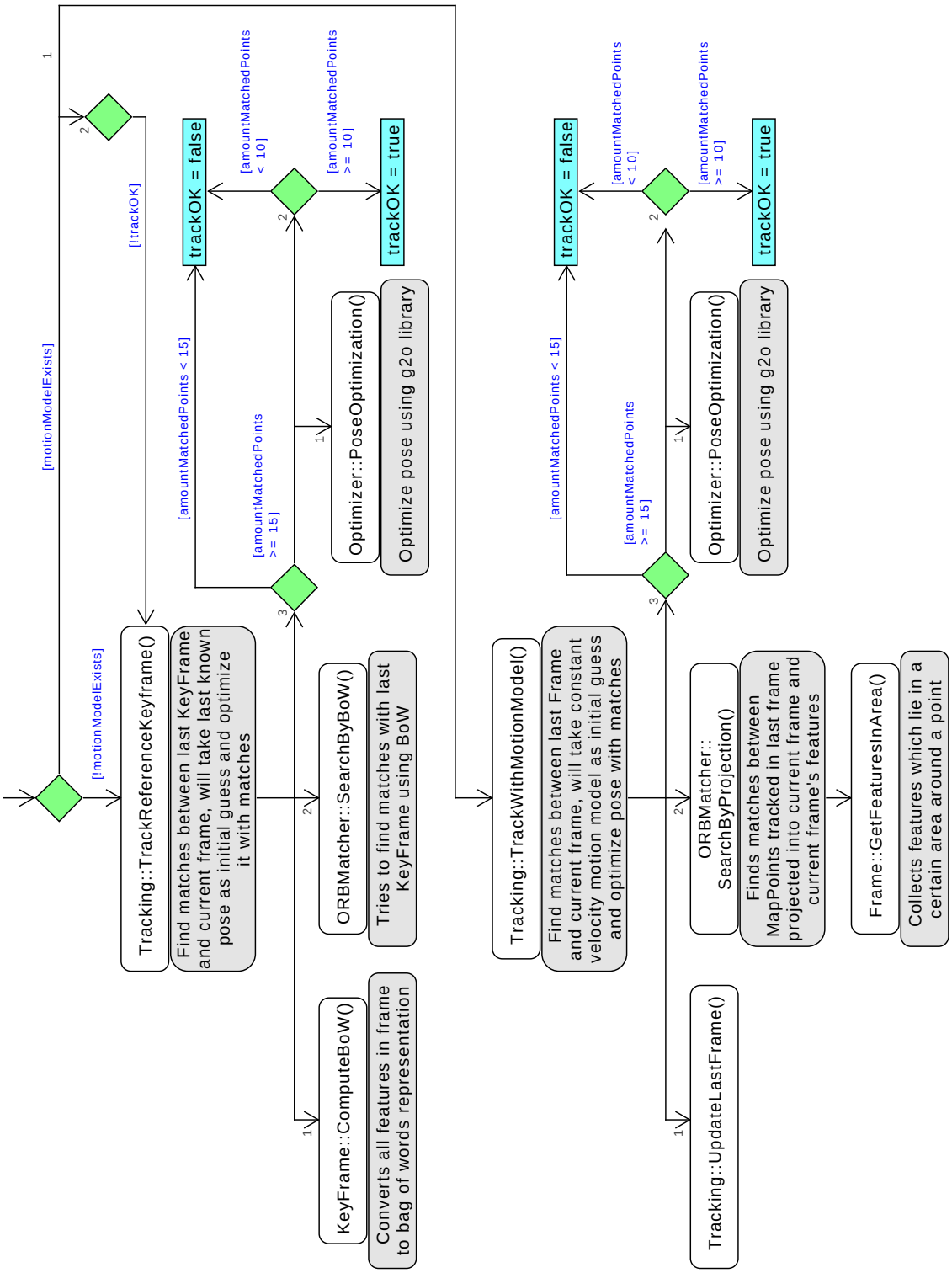


Figure A.6: ORB SLAM tracking module diagram (part 1)

A.7 ORB SLAM tracking diagram (part 2)

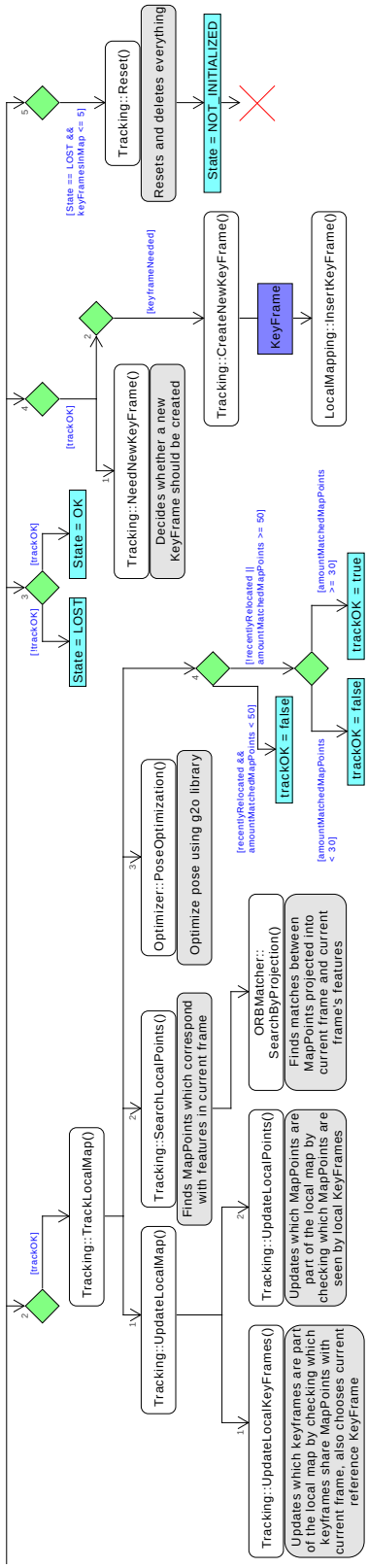


Figure A.7: ORB SLAM tracking module diagram (part 2)

A.8 ORB SLAM relocalization diagram

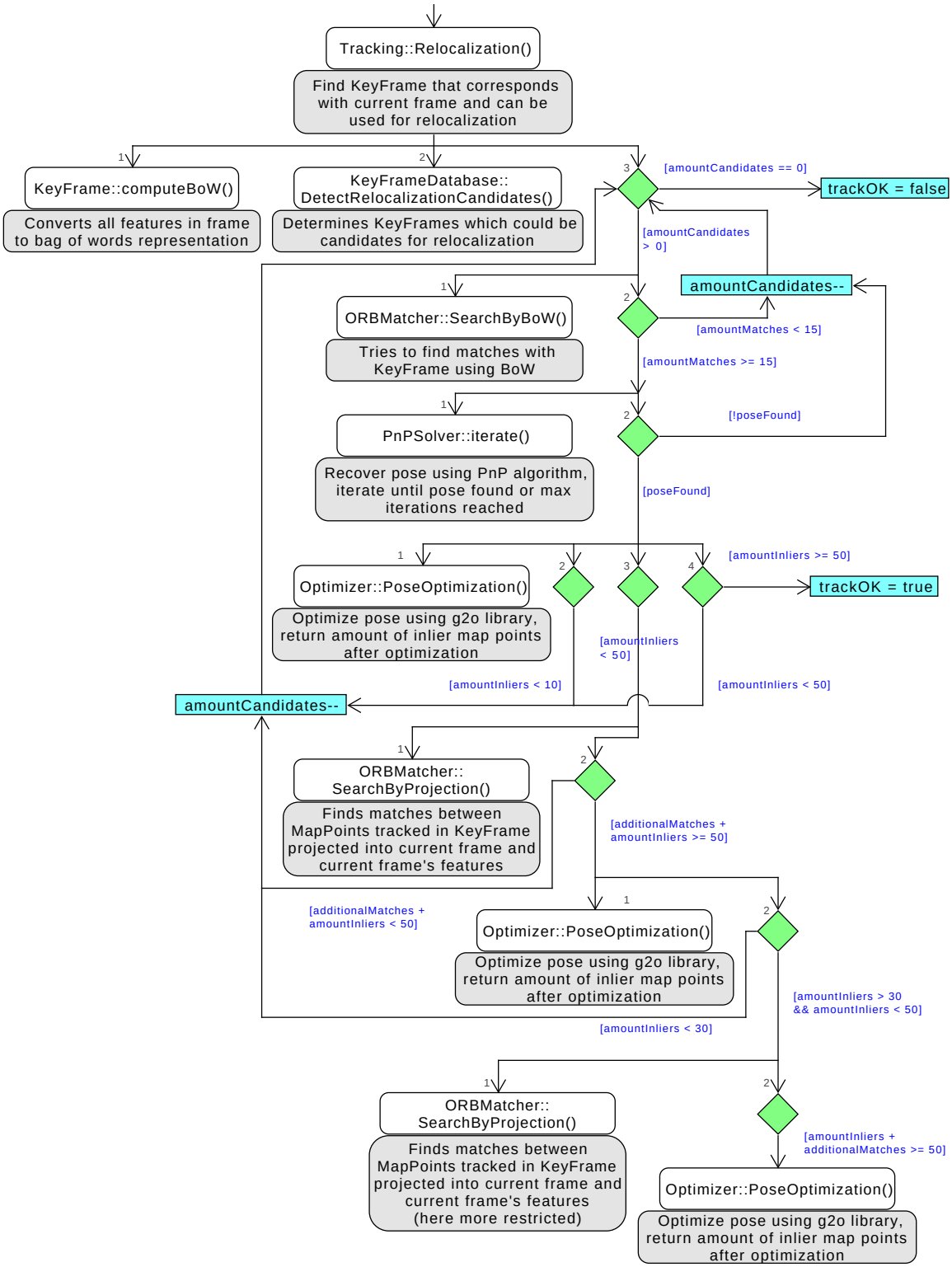


Figure A.8: ORB SLAM relocalization module diagram

A.9 ORB SLAM local mapping diagram

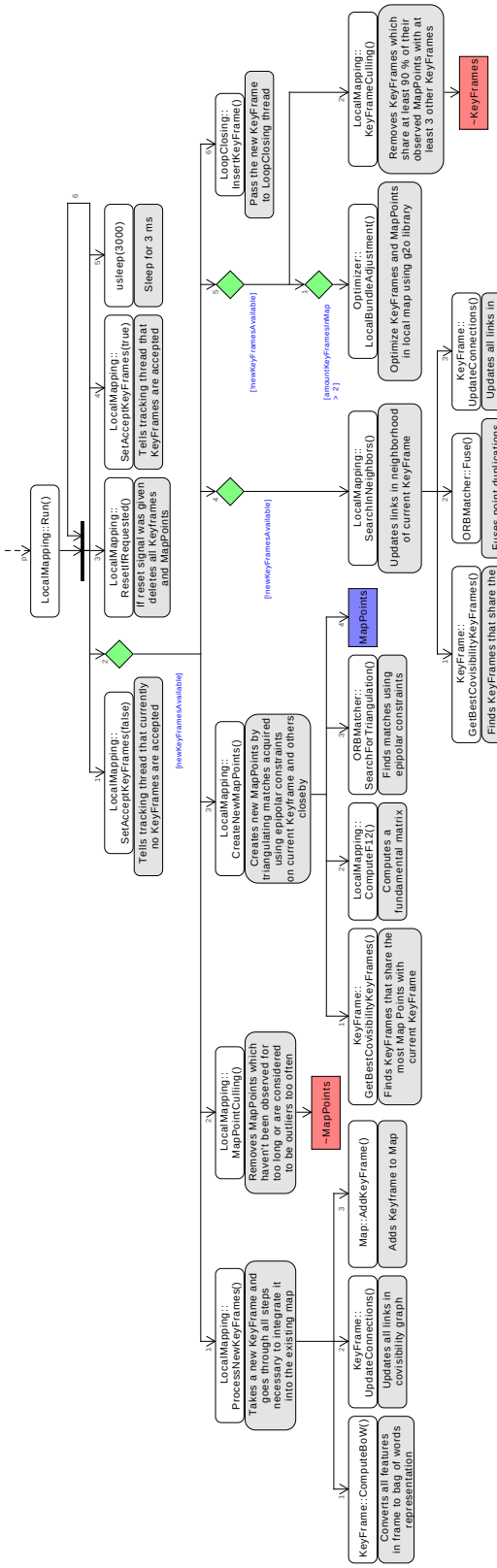


Figure A.9: ORB SLAM local mapping module diagram

A.10 ORB SLAM loop closing diagram (part 1)

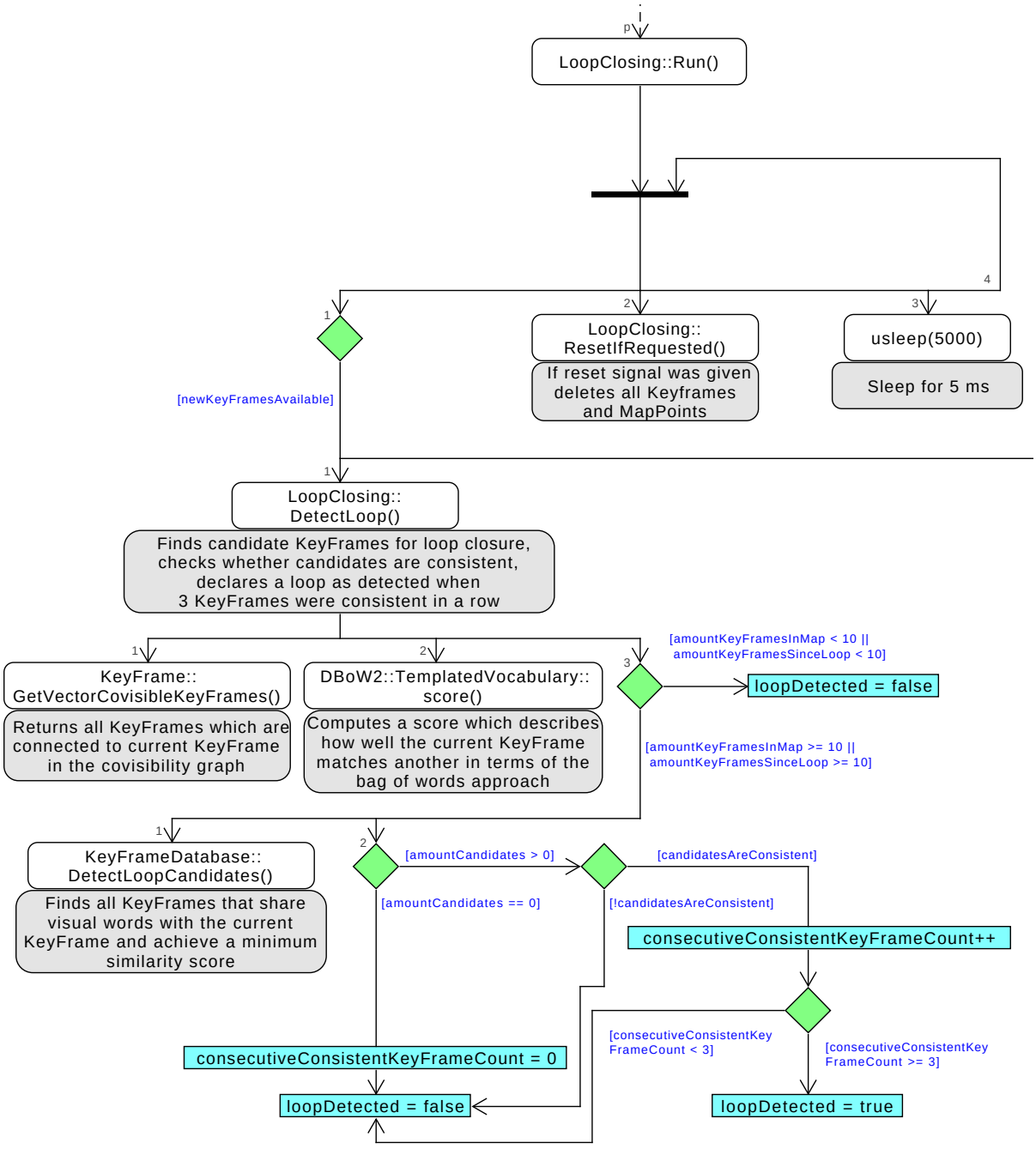


Figure A.10: ORB SLAM loop closing module diagram (part 1)

A.11 ORB SLAM loop closing diagram (part 2)

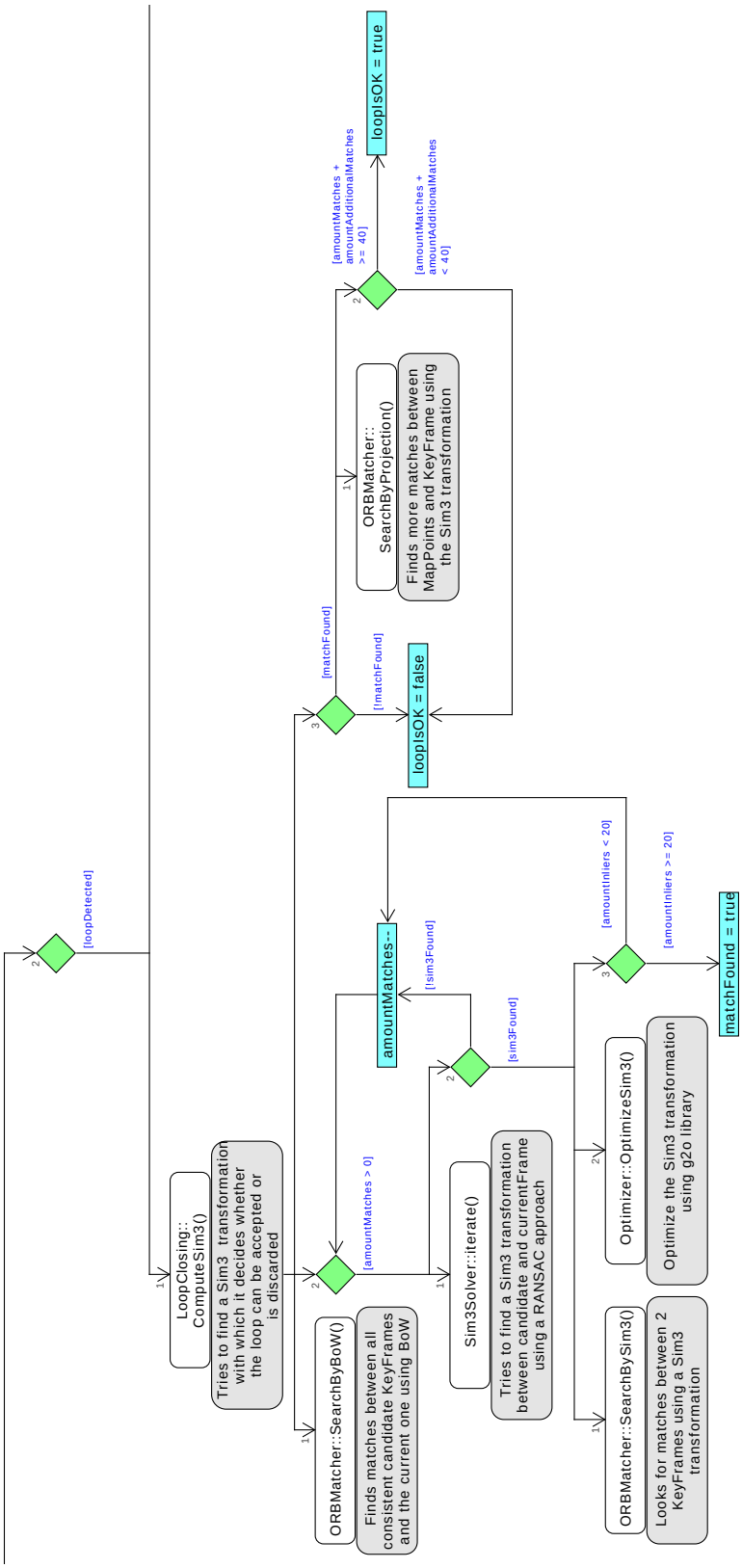


Figure A.11: ORB SLAM loop closing module diagram (part 2)

A.12 ORB SLAM loop closing diagram (part 3)

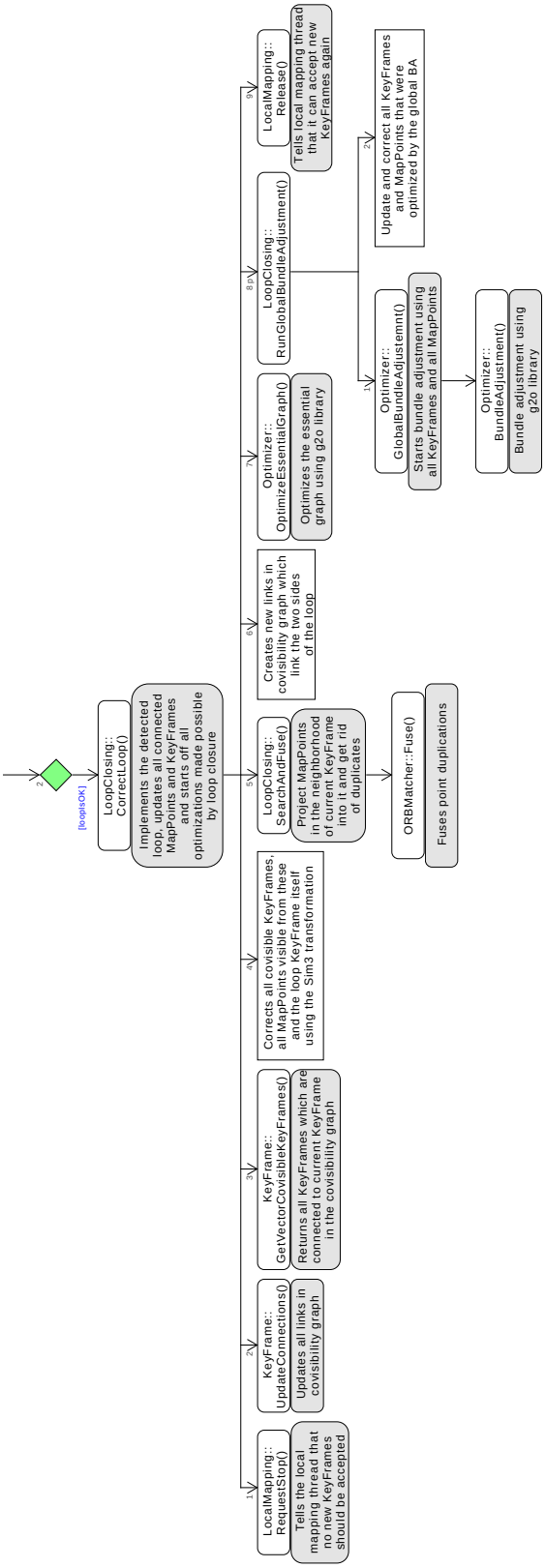


Figure A.12: ORB SLAM loop closing module diagram (part 3)