Retail Store Visual Structure-from-Motion

Valter André Ribeiro dos Santos Piedade
Instituto Superior Técnico, Lisboa
valter.piedade@tecnico.ulisboa.pt

***Abstract*—3D mapping technologies have received increasing attention over the last few years. Among these technologies are Simultaneous Localization and Mapping (SLAM) and Structure-from-Motion (SfM) systems. One of the many applications of these systems is found in retail stores where robots are used to, *e.g.* , autonomously navigate through the store to stock the shelves or detect missing products. In this thesis a SfM system to work in a retail store was implemented from scratch in C++, using the structure of the well-known ORB-SLAM2, proposed in [1], as a baseline for the developed method. The system receives images from a moving stereo camera and uses them to estimate visual odometry and to build a map of the traveled region. The pipeline is modular so that various feature detectors and several methods of visual odometry estimation can be used. Two hybrid methods for visual odometry estimation are further proposed in this thesis. One of them aims at exploiting the typical geometric structure of retail stores. Results of the developed system were obtained using acquired datasets from a retail store and the KITTI dataset, and show some ablation studies for each module of the pipeline.**

***Index Terms*—Structure-from-Motion, visual odometry, mapping, keyframe, keypoint.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) and Structure-from-Motion (SfM) have received increasing attention over the last years due to their applications in several industries like robotics, autonomous vehicles, mapping, security, and more.

One particular use of these two types of systems can occur in a retail store. This use has been increasingly adopted to have robots that can, *e.g.* , store, pick up, or detect products on the shelves. It is, therefore, crucial to have a correct map of the store so that the robot knows its location in the environment so that it can efficiently perform its function.

The work carried out in this dissertation is included in a research project that aims at developing and implementing a vision-based SLAM module for an autonomous mobile robot in a retail store environment. The research project working plan is divided into three stages. Stage I is the development and implementation of algorithms for the localization of an autonomous mobile robot working in the salesroom of a retail store. Stage II is the development of algorithms for map update and simultaneous location and mapping for the autonomous mobile robot described above, using the same set of sensors as in Stage I. Lastly, Stage III are tests, adaptation, and integration of the system on the robot platform built as part of the project.

The work presented in this dissertation fits in Stages I and II of the work plan and is an SfM system that differs from the SLAM system proposed for the project by not having, mainly, a re-localization module nor being optimized for real-time. The developed system is capable of, based on images captured by a stereo camera, estimating the movement of the camera along its trajectory, while creating a map of the region based on the 3D points computed along the path and adjusting their positions using bundle adjustment. The system was also developed to be modular in several aspects, such as the type of image features to extract and odometry estimation methods to use.

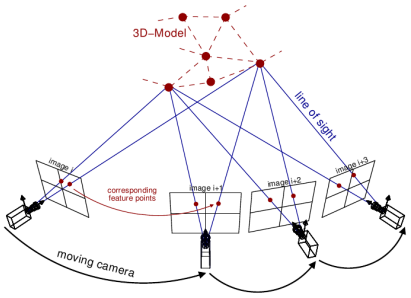


Fig. 1: Operating scheme of a Structure from Motion system. Source: theia-sfm.org.

Since the system will operate mainly in a retail store environment, which is characterized by a geometric structure with many orthogonal components, this geometric structure can be exploited for better environment-specific results by, *e.g.* , assuming a Manhattan World structure.

The goals to be achieved with this thesis are:

- Implement a Structure-from-Motion pipeline in C++, based on a known pipeline;
- Integrate the use of several types of keypoint extractors and several conventional odometry estimation methods;
- Create an odometry estimation method that takes advantage of the structured retail store environment.

The desired result is to obtain a map of the environment in which the robot has traveled, together with its trajectory.

II. Background

SLAM systems have the goal of estimating the position of a robot and its trajectory, while simultaneously building a map of the traveled region. To achieve this, sensors such as cameras, LiDAR, sonar, GPS, IMU, among others are used. Although it can be used offline, the main focus of this type of system is its use in real-time. SfM systems, on the other hand, aim to reconstruct the 3D environment from a series of 2D images taken from different positions, as illustrated in Figure 1. Unlike SLAM systems, these systems work offline.

In some situations, SfM and SLAM systems work similarly, namely when both use cameras and when the different camera positions in the region are captured by a robot moving on a particular trajectory. In this case, the estimation of the robot's trajectory is necessary for the construction of the map. Two steps can then be considered in the operation of these systems, odometry estimation, and mapping.

A. Visual Odometry

Odometry estimation, or ego-motion, consists of estimating the motion of the camera. This movement is described by a rotation and a translation typically between consecutive frames, which is used to convert the referential of each received image into a global referential common to the whole map. Several approaches have been developed to solve the problem of visual odometry. These approaches either use 2D points, 3D points, or a combination of both, and can be divided into featureless, feature-based, or deep learning methods.

1) Featureless Registration: Featureless methods perform 3D registration without resorting to keypoints extracted from the images. For this type of method, the Iterative Closest Point (ICP), [2], and its variants are the standard approach. These methods create correspondences between points based on their proximity. They have, however, some disadvantages, the main one being the convergence in local minimums, therefore requiring a good initial estimation.

2) Feature-based Registration: Feature-based registration methods require the use of visual features extracted from images and their correspondence between different images. This type of method is divided into several steps. The first is to extract keypoints from the two images. These points usually correspond to edges, corners, blobs, or ridges. Next, it is necessary to find matches between the keypoints of the two images. To do so, descriptors are extracted for each detected keypoint. The descriptors characterize the region in the image from which the keypoint was extracted and it is by comparing the descriptors that the correspondences between the points are found. The correspondences indicate that the same point is seen from two different perspectives. The last step is to compute the rotation and translation between the two cameras, based on the points they have in common.

For feature detection, several methods can be used, such as SIFT, SURF, ORB, and AKAZE, proposed in [3]–[6].

Based on the keypoints detected in consecutive images, correspondences between them are detected. Many keypoint matching methods use brute force in this process and then use an outlier removal method to eliminate outliers and estimate the best transformation based on the inliers. One popular solution to perform outlier removal is RANSAC, proposed in [7], which iteratively randomly selects sets of points and classifies them into outliers and inliers depending on whether they fit the desired model. The trade-offs of this method are that it does not guarantee the optimal solution and fails in case there are too many outliers.

Having the correspondences between points, the Orthogonal Procrustes method ([8]) or the solution proposed by Umeyama in [9] can be used.

3) Deep Learning Registration: Due to the recent increase in studies involving machine learning, namely the study of CNN (Convolutional Neural Networks), several signs of progress have been achieved in the area of odometry estimation. These new learning-based methods can surpass or complement the classical methods and achieve state-of-the-art results. The versatility of neural networks enabled the creation of deep learning methods that solve the problems of feature extraction ([10]), feature matching ([11]), registration of point clouds ([12]), among others.

B. Mapping

The 3D mapping problem consists of joining several sets of point clouds acquired at different times and in different positions, either by several static sensors looking over the same region or by sensors moving along a path, thus creating a map of a region. To create a map in which all these point clouds are correctly aligned, it is necessary to transform them all to the same referential using the positions calculated in the odometry estimation. The odometry estimation problem is thus part of the mapping problem. Various methods allow these transformations to be computed not only in the case where there is movement but also in cases where it is intended to reconstruct a map from several static sensors that have different perspectives from the same region. The methods to be analyzed are Bundle Adjustment and Rotation Averaging.

Bundle Adjustment (BA) is an essential component in solving SfM problems, as shown in [13]. It is capable of making an optimal visual reconstruction of a 3D structure and estimating the position of the camera or its calibration parameters using feature points and their correspondences. To obtain an optimal solution, the problem is defined as an optimization problem that minimizes the difference between a given point and its 3D projection on the image plane. This minimization is usually done using the l_2 norm, and the problem is typically solved by Levenberg–Marquardt's algorithm. This method, however, requires a good initialization of its parameters and has a long execution time for large data sets. To ensure a correct mapping using bundle adjustment, it is also necessary to consider loop closure. This way, the map is coherent even when it returns to a known position.

Rotation averaging is a common alternative to bundle adjustment that has multiple strands as explained in the survey [14]. One of the strands is the problem of single rotation averaging, in which the same rotation $\mathbf{R}$ is calculated with data from various measurements, the final result being the average of the estimated rotations. If there are noise measurements that cause wrong rotations, some of the effects of these rotations will be removed when averaging.

C. Baseline Visual SLAM System

The system used as the baseline for the proposed SfM system was the ORB-SLAM2, proposed in [1]. ORB-SLAM2 is a real-time SLAM system based on ORB features and is divided into four modules: tracking, local mapping, loop closure, and re-localization. The pipeline is shown in Figure 2. Tracking estimates the position of the camera along

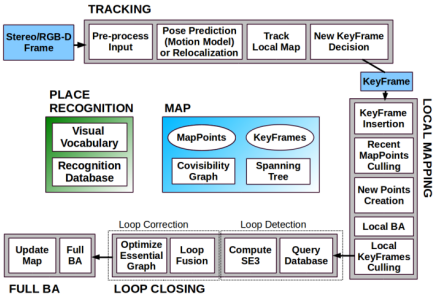


Fig. 2: ORB-SLAM2 framework. Source: [1].

its trajectory using correspondences between keypoints of consecutive frames. Local mapping builds the map along the trajectory, and locally optimizes the position of the keyframes to reduce the odometry estimation error. Loop closing reduces the error of the entire map by performing a full BA when the camera returns to a previously mapped position. The relocation operates alternately to the construction of the map and works as an extra module that is only used after the map has been built. For that, it uses Bag of Words to create an image database which is used to efficiently compare new images with already processed images.

The main processes of this system like pose estimation, local BA optimization, and full BA optimization are performed using graph optimization. In the case of pose estimation, points with correspondences between consecutive frames are inserted in the graph with fixed positions and only the positions of the frames are optimized, from which the movement between frames is estimated. In the case of BA, the positions of keyframes and 3D points are optimized based on the points that the keyframes have in common. For the local BA, only the keyframes and their respective points close to the current keyframe are used, while in the full BA all keyframes and 3D points are used. These procedures have been shown to yield good results, however, they require correct tracking of points over several frames.

III. 2-STEP VISUAL ODOMETY

As an alternative to the methods presented in Section II-A2 for odometry estimation, two hybrid methods are proposed. The methods are hybrids since they combine two different methods, one for rotation estimation and one for translation estimation. Each hybrid method has its method for rotation estimation but they share the method for translation estimation.

A. Rotation Estimation

The two proposed methods for estimating the rotation are presented in this subsection. The first method uses 2D-2D correspondences to estimate the essential matrix. By decomposing this matrix using SVD the rotation and translation at less than a scale factor are obtained. The second method uses lines detected in the image, instead of keypoints. From the detected lines, three vanishing points are estimated with which the orientation of the camera is obtained. This method aims to exploit the structured environment of retail stores.

1) Essential Matrix Estimation: The first method used for estimating the rotation between two consecutive frames uses the essential matrix. The essential matrix represents the geometrical relationship between matching points of two images. Its estimation uses the detected keypoints (2D points of the image coordinates), instead of 3D points. The desired rotation can be estimated by decomposition of the essential matrix.

The estimation of the essential matrix requires two sets of points that have correspondences with each other. The relation between the sets of points is given by

$$\mathbf{x'}^T \mathbf{E} \mathbf{x''} = 0, \tag{1}$$

as used in [15], where $\mathbf{x'}$ is the set of points of the current frame, $\mathbf{x''}$ is the set of points of the previous frame, and $\mathbf{E}$ is the essential matrix. (1) defines the coplanarity constraint between the two sets of points.

The essential matrix is then computed from (1) based on the five-point algorithm solver described in [16], that uses a RANSAC framework to remove outliers from the estimation. The matrix obtained from this algorithm has the following format

$$\mathbf{E} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}. \tag{2}$$

Having the essential matrix, the rotation and translation are obtained by decomposing it using SVD according to

$$\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are 3×3 orthogonal matrices with the singular vectors of $\mathbf{E}$ and $\mathbf{S}$ is a 3×3 diagonal matrix with the singular values of $\mathbf{E}$. With the matrices obtained from the decomposition, the rotation matrix $\mathbf{R}$ and the translation vector $[\mathbf{t}]_\times$ are computed by

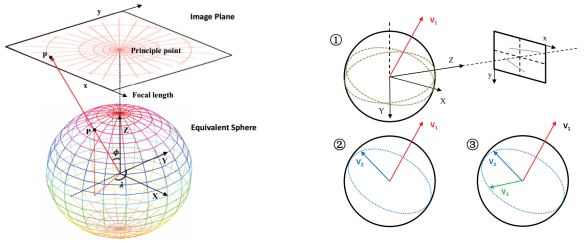
$$\mathbf{R} = \mathbf{U} \begin{bmatrix} 0 & \pm1 & 0 \\ \pm1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{V}^T \tag{4}$$

$$[\mathbf{t}]_\times = \mathbf{U} \begin{bmatrix} 0 & \pm1 & 0 \\ \pm1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{S}\mathbf{V}^T. \tag{5}$$

This process gives four solutions, but only one of them places the points in front of both cameras, being that the desired solution. Only the rotation matrix $\mathbf{R}$ is used since the translation is obtained at less than a scale factor, so only information about its direction is available.

2) Vanishing Points Estimation: The second method used to estimate a rotation matrix between two images of consecutive frames requires the estimation of vanishing points. For this, the method presented in [17] was used. This method uses lines detected in the images to estimate vanishing points, assuming the environment is under the Manhattan World assumption. This assumption states that all surfaces in the environment are aligned along with three dominant directions. Each dominant direction is thus described by one vanishing point. This method only works with images taken by pinhole cameras.

In order to be efficient for real-time applications, a polar grid is created by expanding the unit vectors on the equivalent



(a) (b)

Fig. 3: Vanishing points estimation: (a) Relationship between the image plane and the equivalent sphere; (b) Procedure of generating orthogonal vanishing points. Source: [17].

sphere to intersect the image plane. This will be used to store the response of each line segments. Figure 3(a) shows the relationship between the image plane and the equivalent sphere. A point $(x, y)^T$ on the image is converted to a 3D point $\mathbf{P} = (X, Y, Z)^T$ on the equivalent sphere according to

$$\begin{cases} X = x - x_0 \\ Y = y - y_0 \\ Z = f \end{cases}, \tag{6}$$

where (x_0, y_0) are the principal point and f is the focal length. The longitude and latitude is further computed using

$$\begin{cases} \phi = \text{acos}(Z/\sqrt{X^2 + Y^2 + Z^2}) \\ \lambda = \text{atan2}(X, Y) + \pi \end{cases}, \tag{7}$$

The first vanishing point $\mathbf{v}_1 = (X_1, Y_1, Z_1)^T$ is computed iteratively by randomly choosing two line segments and computing their intersection point (step 1 of Figure 3(b)). Since the three vanishing points are orthogonal, the second vanishing point will belong to the great circle of $\mathbf{v}_1$ (step 2 of Figure 3(b)). The circle is divided into fractions of 360° (1% accuracy), and for each fraction a vanishing point $\mathbf{v}_2 = (X_2, Y_2, Z_2)$ is computed according to

$$\begin{cases} X_2 = \sin(\phi)\ \sin(\lambda) \\ Y_2 = \sin(\phi)\ \cos(\lambda) \\ Z_2 = \cos(\phi) \end{cases}, \tag{8}$$

and

$$X_1\ X_2 + Y_1\ Y_2 + Z_1\ Z_2 = 0, \tag{9}$$

where ϕ is the latitude and λ is the longitude. Finally, the third vanishing point $\mathbf{v}_3$ is the cross product of $\mathbf{v}_1$ and $\mathbf{v}_2$

$$\mathbf{v}_3 = \mathbf{v}_1 \times \mathbf{v}_2, \tag{10}$$

since the three vanishing points must be orthogonal to each other (step 3 of Figure 3(b)).

Several hypotheses are thus produced for the set of the three vanishing points, so it is necessary to validate and choose the best set. The validation is done by computing the response of the detected line segments to each hypothesis, and the vanishing points that produce the best response are selected.

Once the vanishing points are estimated, they are combined to get the rotation matrix that describes the rotation of the image. With this approach, however, it is not possible to estimate the translation.

B. Translation Estimation

Using the rotations obtained with each of the methods in the previous subsection, a translation estimation method is proposed that combines RANSAC for outlier removal and the least-squares method for translation estimation.

Having correspondences between points, to estimate the rotation and translation that align a point ${}^{\text{l}}\mathbf{p} = [{}^{\text{l}}x, {}^{\text{l}}y, {}^{\text{l}}z]^T$ in the last frame and a point ${}^{\text{c}}\mathbf{p} = [{}^{\text{c}}x, {}^{\text{c}}y, {}^{\text{c}}z]^T$ in the current frame, at least 3 sets of matching points are required. Since the rotation has already been estimated, instead of needing a minimum of 3 sets of matching points, only 1 set is needed. A point is then chosen randomly, with which the translation is computed. Having the rotation and translation, the number of inliers is computed based on the squared error e between the point in the current frame and the point in the previous frame transformed to the current frame as follows

$$e = \|{}^{\text{c}}\mathbf{p} - (\mathbf{R} \cdot {}^{\text{l}}\mathbf{p} + \mathbf{t})\|^2. \tag{11}$$

The choice between inliers and outliers is established based on a predefined threshold value.

This process of randomly choosing a point, computing the error according to (11), and counting the number of inliers is done iteratively over a fixed number of iterations. In the end, the inliers of the iteration that produced the most inliers are obtained.

Using the inlier points obtained, the translation is estimated using the least squares method. The residual r_i of a point $i \in \{0, 1...N\}$ is defined as the difference between the value of the point i in the current frame ${}^{\text{c}}\mathbf{p}_i = [{}^{\text{c}}x_i, {}^{\text{c}}y_i, {}^{\text{c}}z_i]^T$ with the value of the point i in the previous frame ${}^{\text{l}}\mathbf{p}_i = [{}^{\text{l}}x_i, {}^{\text{l}}y_i, {}^{\text{l}}z_i]^T$ transformed to the referential of the current frame, according to

$$r_i = {}^{\text{c}}\mathbf{p}_i - (\mathbf{R} \cdot {}^{l}\mathbf{p}_i + \mathbf{t}), \tag{12}$$

where $\mathbf{R}$ is the previously estimated 3×3 relative rotation matrix and $\mathbf{t}$ is a 3×1 column vector corresponding to the desired translation.

The goal is to obtain the optimal value of the parameter $\mathbf{t} = [t_x, t_y, t_z]^T$ that minimizes the square sum of the residuals S, according to

$$S = \sum_{i=1}^{N} \|r_i\|^2 = \sum_{i=1}^{N} \|{}^{\text{c}}\mathbf{p}_i - (\mathbf{R} \cdot {}^{\text{l}}\mathbf{p}_i + \mathbf{t})\|^2. \tag{13}$$

Solving this problem results in

$$\frac{\partial S}{\partial \mathbf{t}} = 0 \Rightarrow \mathbf{t} = \frac{\sum_{i=1}^{N} {}^{\text{c}}\mathbf{p}_i - \mathbf{R} \cdot {}^{\text{l}}\mathbf{p}_i}{N}. \tag{14}$$

IV. Structure from Motion Pipeline

The developed Structure from Motion (SfM) system is based on the ORB-SLAM2 pipeline, being divided into three modules: Tracking, Local Mapping, and Loop Closure. Based on a sequence of images received, Tracking extracts important features from the images, finds matches between consecutive frames, and estimates the position of the camera along with

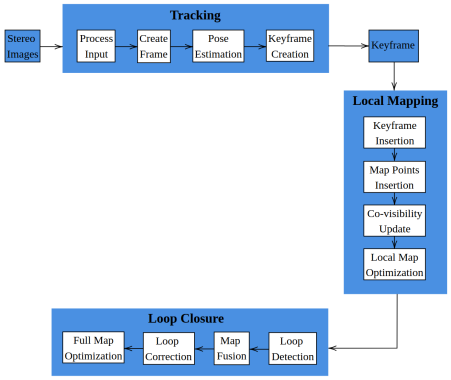


Fig. 4: Developed SfM system pipeline.

its movement (visual odometry). It also creates keyframes (detection of important camera positions based on the available features over time). Local Mapping aims to optimize the position of new keyframes and their respective map points, taking into account the existing keyframes (and respective map points) close to the new keyframe. The final module, Loop Closure, optimizes the entire map when a loop is detected. That is when the camera returns to a position already known on the map. Figure 4 shows how the three modules are combined to create the SfM system.

A. Tracking

The Tracking module starts by receiving a pair of images from a stereo camera. The images can come from either a fisheye or a pinhole camera and are assumed to be rectified and their calibration is also assumed to be available. For every pair of stereo images received, a frame is created. In each frame, keypoints are detected and descriptors are extracted using various alternatives for the type of features (*e.g.* SIFT, ORB, or AKAZE features). With the descriptors, the keypoints of the left and right images are matched, which allows the estimation of 3D points. Using the computed 3D points or the 2D keypoints with matches in consecutive frames, it is estimated the relative motion between the consecutive frames, using 3D-3D or 2D-2D correspondences. The motion estimated is accumulated over time to obtain the camera movement from the current frame to the world reference frame. The final step of Tracking is creating keyframes. Keyframes represent similar sets of frames and prevent repeated information from being inserted into the map. New keyframes have the same information as the frame that created it and are created according to the conditions defined in Section IV-A4.

1) Process Image Input: The Tracking module input is the pairs of images taken by a stereo camera. Images from two different types of cameras can be processed: pinhole and fisheye. Both types of images can be processed directly, however in fisheye images, the distortion can be removed thus converting the image to the pinhole model. Although this remapping removes the distortion, the edges of the image get motion blur, which mainly affects feature detection in those areas.

The images also need to be rectified, *i.e.* the images from the two cameras need to be parallel. As this is not always the case it is necessary to estimate a transformation that remaps the right image so that corresponding points have the same y-coordinate in both images. This must be the case since the method of computing 3D points and the optimizations made in the Local Mapping and Loop closure modules depends on this characteristic.

2) Create Frame: Frames describe each stereo image pair received. It contains all the information extracted from the images. This information is keypoints, descriptors, correspondences between keypoints and 3D points.

Keypoints are points on the image that are differentiated either by color, intensity, texture, among others. Their differentiation is important to allow an easier match with other similar keypoints. Descriptors are calculated for each detected keypoint. The descriptors contain information about the region around the keypoint in the compact form of a vector and are used to determine whether two keypoints are similar. Three different types of features were implemented: ORB, SIFT, and AKAZE. To reduce the computation time for keypoint detection, a mask for the image is created. This mask corresponds to a binary matrix with the dimension of the image and indicates in which regions of the image to look for keypoints. The mask is constructed by selecting a circular region around all keypoints that have a stereo match. This mask is reconstructed whenever a new keyframe is created.

The matching of keypoints between the stereo image pair is done based on the descriptors extracted for each keypoint of each image. The matching is done using a brute-force descriptor matcher that searches for k best matches of the right image for each descriptor of the left image. The best matches are those with the smallest distance, which are computed using the l_2 norm. The default value used for k is 2. For values of $k \geq 2$ the two keypoints of the right image with the smallest distance to the keypoint of the left image may have close distances. In this case, Lowe's ratio test is applied. To further eliminate bad matches, the fundamental matrix is estimated. This estimation uses a RANSAC algorithm, which allows the exclusion of matches that are classified as outliers in the matrix estimation.

The use of stereo cameras allows 3D points to be easily computed using only the images of a single frame. Since the images are rectified, both belong to the same plane, and there is only a translation on the X axis between the left and right images, which corresponds to the baseline of the camera. The baseline value is a known input parameter since the cameras are calibrated.

A 3D point $\mathbf{X} = [x, y, z]^T$ in the left camera reference is estimated using

$$d = x_{\text{left}} - x_{\text{right}} \tag{15}$$

$$z = \frac{b \cdot f_x}{d}, \tag{16}$$

$$x = \frac{x_{\text{left}} \cdot z}{f_x} \tag{17}$$

$$y = \frac{y_{\text{left}} \cdot z}{f_y}, \tag{18}$$

where d is the disparity between the cameras, b is the stereo baseline and $f = (f_x, f_y)$ is the focal length of the left camera.

3) Pose Estimation: The pose estimation sub-module aims at estimating the trajectory of the camera along its path in the world reference frame. The world reference is defined as being the reference of the first keyframe.

The movement of the current frame i in the world reference frame ${}^{W}\mathbf{T}_i$ is computed as the accumulation of the relative transformations between consecutive frames according to

$$ {}^{W}\mathbf{T}_i = {}^{W}\mathbf{T}_1 \cdot {}^{1}\mathbf{T}_2 \cdot \ldots \cdot {}^{i-1}\mathbf{T}_i, \tag{19}$$

where i is the current frame.

The relative transformations between frames ${}^{i-1}\mathbf{T}_i$ can be estimated using different methods. The two hybrid methods proposed can be used, as well as other known methods such as Umeyama. Since these methods use 2D-2D or 3D-3D correspondences between frames, it is necessary to find matches between the keypoints of the left images of the consecutive frames. Only keypoints for which there is an estimated 3D point, are used.

4) New Keyframe Decision and Creation: The last task done in the Tracking module is the decision of creating new keyframes, and their creation when that decision is favorable. A keyframe is defined as a frame that represents similar frames of a certain region. The importance of creating keyframes is related to the complexity of the map. It is necessary to ensure that frames with new features of the environment will be given to the map, at the same time that frames that will only add features already in the map are only used to estimate the trajectory.

Two conditions were defined to decide when a new keyframe is created:

1) The current keyframe tracks less than 30% of the points of the current keyframe.
2) Each keyframe can only represent, at maximum, 30 frames.

The first condition ensures that new information about the environment is not lost by grouping frames that represent different areas of the environment and the second condition guarantees that the complexity of the map is kept reduced while creating enough keyframes to guarantee a correct mapping.

B. Local Mapping

The Local Mapping module is responsible for creating and managing the map. The map consists of the keyframes created in the Tracking module and the corresponding 3D points (map points). When creating or adding data to the map, the module is responsible for adding keyframes and map points. However, since different keyframes can see the same map points, it is necessary to evaluate when to insert new map points or update existing map points. After the new data has been inserted into the map the positions of the current keyframe, the keyframes connected to it, and the map points belonging to those keyframes are optimized. This optimization aims to reduce the impact of trajectory estimation errors on the Tracking module.

1) Map: The map has two types of data, keyframes and map points. Keyframes represent sets of similar consecutive frames and map points are the 3D points seen by the keyframes. The map thus has the keyframes created and the map points associated with each keyframe. Map points can belong to several keyframes and keyframes have connections between each other based on the map points they have in common, creating a co-visibility graph.

2) Map Creation and Update: The first step in Local Mapping is to add new keyframes to the existing keyframe map. This is a simple process since each keyframe is unique, so there is no risk of having two repeated keyframes in the map.

The second step is to insert the new keyframe map points into the map. Unlike the keyframes, for the map points, it is necessary to check if those points already exist on the map (seen by other keyframes) or if they are new points. For that, matching is made between the map points seen by the new keyframe and by the previous keyframe. Based on the matches, the points that did not have a match are considered new and are inserted in the map. The ones that had a match are not inserted, and the new observations for those points are added to the respective map points.

Finally, co-visibility connections are made between the keyframes. These connections are established based on the number of map points that the keyframes have in common.

3) Local Map Optimization: To correct some of the drift that is inevitably obtained in this type of system, due to the continuous integration of odometry estimation errors, a local BA is performed. This optimization will adjust the position of the current keyframe, the position of the keyframes connected to it according to the co-visibility graph, and the 3D points of all these keyframes, based on the points seen by each keyframe (observations). The observations correspond to the coordinates of the keypoints that originated the 3D points.

The optimization is done using the Levenberg–Marquardt method and aims to minimize the reprojection error between 3D points seen by multiple keyframes, according to observations $\mathbf{o} = [u_L, v_L, u_R]^T$, where (u_L, v_L) are the keypoint coordinates of the left image and u_R is the keypoint horizontal coordinate of the right image.

Let $\mathcal{K} = \{\mathbf{k_1}, ..., \mathbf{k_n}\}$ be the set of keyframes that includes the current keyframe and the keyframes connected to it by co-visibility connections, and defining $\mathcal{X}_{\mathbf{k_i}} = \{\mathbf{x_1^i}, ..., \mathbf{x_m^i}\}$ as the set of 3D points that are seen by the keyframe $\mathbf{k}_i \in \mathcal{K}$, the cost function to minimize is

$$ \sum_{i=1}^{n}\sum_{j=1}^{m} \rho(\|\mathbf{o}_j^i - \pi(\mathbf{R}_{\mathbf{k}_i}\mathbf{x}_j^i + \mathbf{t}_{\mathbf{k}_i})\|^2), \tag{20}$$

where ρ is the robust Huber cost function, $\mathbf{R}_k \in SO(3)$ and $\mathbf{t}_k \in \mathbb{R}^3$ are the orientation and position of the keyframe k, respectively, and π is the function that projects the 3D points onto the image, according to

$$ \pi\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \\ f_x \frac{X-b}{Z} + c_x \end{bmatrix}, \tag{21}$$

where $f = (f_x, f_y)$ is the left camera focal length, $c = (c_x, c_y)$ is the left camera principal point and b is the stereo camera baseline.

C. Loop Closure

Loop closure aims to detect when the camera returns to a previously visited position that is already mapped. Throughout the motion, it is common for the trajectory error to increase mainly due to the accumulation of several small motion estimation errors. By returning to an already known position it is possible to eliminate some of that error so that it does not grow infinitely.

1) Loop Detection: The first step in correcting the loop closure error is to detect when the loop is closed. To do so, two procedures are performed. The first detects loop candidates and the second checks whether any of the candidates are valid.

The first step of loop candidate detection uses Bag of Words (BoW) place recognition. Whenever a new keyframe is created in the Tracking module, the left stereo image of the created keyframe is converted into a BoW word. This word is created based on the keyframes and descriptors detected in the image and acts as a frequency histogram of the features in the image. After creating the word, it is inserted into a database that associates the word with the ID of its keyframe. In the loop closure module, a comparison of the word created for the current keyframe is made with the words in the database to compute a similarity score between the keyframes using the l_1 norm. The values of the scores obtained are between $[0, 1]$ and the keyframes that have a score greater than 0.9 times the maximum score among the keyframes connected to the current keyframe are considered candidates for loop closure.

The second step is to validate the detected candidates. Although BoW is a good solution that allows fast matches between all keyframes, perceptual aliasing can occur. To avoid such situations, a validation of the geometric consistency between the candidate and the current keyframes is performed through feature matching. For each candidate, a match is made between the descriptors of the candidate and current keyframe keypoints. The candidates are only valid if at least 30% of the points of the current keyframe have matches in the candidate keyframe.

Since the optimizations performed to correct loop closure can be computationally high, new loops are only detected 10 keyframes after the last loop was detected.

2) Map Fusion: After validating the detection of a new loop closure, it is necessary to pass that information to the map. To do so, an update to the map point observations of the points that the current keyframe and the loop keyframe have in common must be done. It is also necessary to create co-visibility connections between these keyframes and possibly other keyframes that share observations for the same points.

Similar to the process done in Section IV-B2, the update of the observations of the map points that the current keyframe and the loop keyframe have in common is done based on the descriptor matching done in the previous section to validate the loop. Only the points seen by both keyframes are thus updated. As the points are updated, the number of points that the current keyframe has in common with other keyframes are counted. This count is necessary since other keyframes may observe the same points that the current keyframe and the loop keyframe have in common. Based on the number of points the keyframes have in common new co-visibility connections are established.

3) Loop Correction: Before performing the final step of optimizing the total map, an adjustment is made to the positions of the keyframes to distribute the loop closure error over the entire trajectory. Two ways of performing the error distribution have been implemented. The first is keyframe optimization and the second is transformation averaging. In both, it is necessary to first estimate the correct position of the current keyframe. Initially, an attempt was made to estimate odometry between the current keyframe and the keyframe that completes the loop with the same method used in the Tracking module, however, the estimation was not correct. Therefore, the position of the current keyframe was estimated by doing an optimization identical to the one in Section IV-B3. Note that after the fusion of the map the set of keyframes connected to the current one now has the keyframe where the loop closure was detected. The difference with the BA performed in Section IV-B3 is that in this case the optimized positions are not kept, but are only used to estimate the relative transformation between the keyframes forming the loop.

The difference with the local BA is that the positions of the keyframes and map points are not optimized, obtaining only the optimized position of the current keyframe.

Having the correct position for the current frame, one of two methods is applied: keyframe optimization or transformation averaging. The keyframe optimization corresponds to the optimization of a pose-graph that has as vertices the position of the keyframes and as edges between the keyframes the relative position between them. There are only edges between keyframes connected by the co-visibility graph. The relative position between keyframes is computed based on the positions of the keyframes in the world referential, except for the edge that connects the loop where the previously estimated position is used.

Let $\mathcal{K} = \{\mathbf{K}_1, ..., \mathbf{K}_n\}$ be the set with all keyframes positions and using the Levenberg–Marquardt method, the goal is to minimize the position error between keyframes according to the cost function

$$\sum_{i=0, j=0, i \neq j}^{n} \|\mathbf{K}_i - {}^{i}\mathbf{T}_j \mathbf{K}_j\|_{\text{frob}}, \tag{22}$$

where $\mathbf{K}_i$ and $\mathbf{K}_j$ correspond to the position of the keyframes i and j, respectively, and ${}^{i}\mathbf{T}_j$ is the relative position from keyframe j to keyframe i. $\|.\|_{\text{frob}}$ denotes the matrix's Frobenius norm.

The second method implemented, transformation averaging, corresponds to the distribution of the average loop error over the keyframes. Based on the computed correct position for the current keyframe and its position with accumulated error, the loop error is estimated. From this error, the translation error and the rotation error are obtained separately and divided by the total number of keyframes. The obtained value is assumed

Dataset	Distance [m]
STORE1	21.91
STORE2	34.23
KITTI00	217.06
KITTI07	694.70

TABLE I: Distance traveled in each dataset.

to be the average accumulated error between each keyframe and is therefore distributed over all the keyframes.

4) Full Map Optimization: To complete the loop closure, all keyframes and map points are optimized. This is done with a BA similar to the one in Section IV-B3, the difference being that all keyframes and map points are used instead of just the current keyframe and those connected to it with co-visibility connections and their 3D points.

V. Experiments and Results

The experimental results performed have been divided into three sections: Tracking, Local Mapping, and Loop Closure. The first section presents the results of using different feature types and different visual odometry estimation methods, using only the Tracking module. The second section compares the Tracking results before integration of the Local Mapping module with the results after integration. Finally, the third section compares the results obtained with the Tracking module only, with Tracking and Local Mapping modules, and with Tracking, Local Mapping, and Loop Closure modules.

The datasets used to test the pipeline belong to two different environments using cameras with different models. The first type of data was acquired for the project and belongs to a retail store environment, which is the main environment in which the system will operate. The camera used is the Intel® RealSense™ T265 fisheye stereo camera, which has an image capture rate of 30 FPS. The second type of data is data from the KITTI dataset, which has several outdoor sequences obtained from a moving vehicle that has two pinhole cameras Point Grey Flea 2 (FL2-14S3M-C) arranged in parallel, forming a stereo pair. The cameras capture new images at a rate of 10 FPS.

Two datasets from the store environment and two datasets from KITTI are used. From the store environment, the two datasets will be referred to as STORE1 and STORE2. From the KITTI datasets, sequences 00 (only the first 300 images) and 07 are used, which will be referred to as KITTI00 and KITTI07. Table I shows the distance traveled in each of the datasets.

For error metrics, two were used, one for rotation errors and one for translation errors. The metrics are as follows:

$$e_R(\mathbf{R}) = \text{acos}\left(\frac{\text{trace}(\mathbf{R}^{-1}\mathbf{R}_{\text{GT}}) - 1}{2}\right) \tag{23}$$

$$e_t(\mathbf{t}) = \|\mathbf{t} - \mathbf{t}_{\text{GT}}\|^2, \tag{24}$$

where $\mathbf{R}_{\text{GT}}$ is the ground-truth rotation and $\mathbf{t}_{\text{GT}}$ is the ground-truth translation.

Using (23) and (24) both relative and absolute errors were estimated. The relative errors are computed between consecutive frames and express the error obtained in the estimation of the movement between consecutive frames. The absolute error is computed with the last frame of the sequence and expresses the total error of the trajectory.

A. Tracking

In this section, the results obtained for the Tracking module alone will be discussed. Being tracking only, the trajectories are just related to the quality of odometry estimation. The results to be shown belong to the datasets KITTI00 and STORE1. For each dataset the three types of implemented features (SIFT, AKAZE, and ORB) were tested, as well as each of the following methods of odometry estimation methods:

- Method EM-R: Essential Matrix for rotation and 1-Point RANSAC for translation (described in Section III);
- Method VP-R: Vanishing Points for rotation and 1-Point RANSAC for translation (described in Section III);
- Method R-Um: 3-Point RANSAC and Umeyama;
- Method Um: Umeyama.

Figures 5(a) and 5(b) show the results obtained using various odometry estimation methods in the STORE1 and KITTI00 datasets, respectively, using SIFT features. Since the method VP-R needs structured environments, it is only used in the STORE1 dataset. As there is no ground truth available for the STORE1 dataset, the results are only evaluated qualitatively and the result produced by ORB-SLAM3 for this dataset is given as reference.

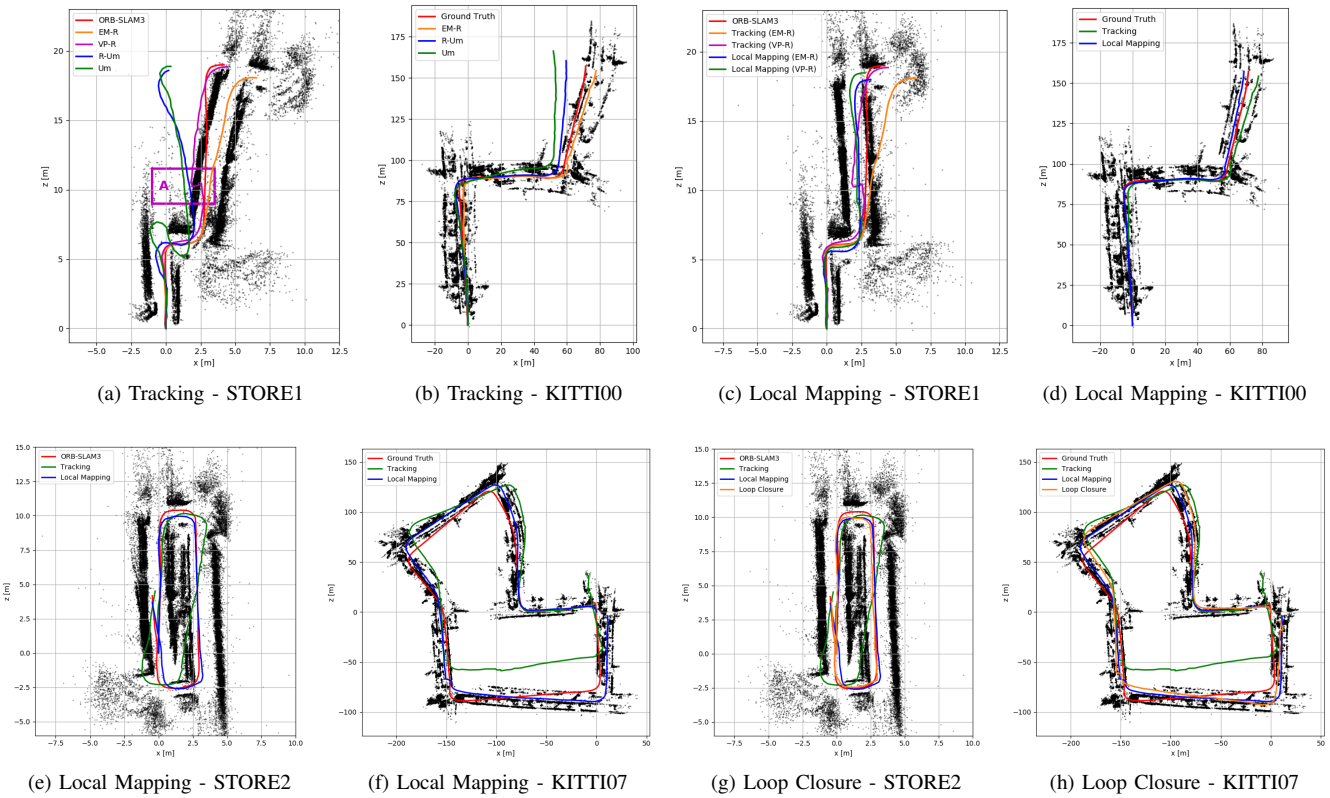
As for features, SIFT features were the best performers, followed by AKAZE features. ORB features were the worst since they caused large errors in a few frames. The major difference is that SIFT features can produce a larger number of keypoints, which gives more frame matches and therefore better estimation.

Regarding the odometry estimation methods, EM-R was the best in both datasets. VP-R on the STORE1 dataset seems to be able to produce good results, but it is not robust and in certain areas of the store it gave poor estimates. In both datasets, the introduction of an outlier removal method improved the results a lot, as can be seen by comparing the results of the Um and R-Um methods.

B. Local Mapping

This section will take the best results from the previous section and add the Local Mapping module. The goal is to analyze if there are improvements in the trajectory by locally optimizing the positions of the keyframes and map points. Therefore, only SIFT features and the methods EM-R and VP-R defined in the previous section will be used.

Figures 5(c), 5(d), 5(e), and 5(f) show the results obtained for the four datasets used. All four results show that the trajectory obtained by using the Local Mapping module improved the Tracking module results by reducing the error of the trajectory and smoothing it when there are sudden position variations caused by estimation errors (zone A in Figure 5(a) ceases to exist in Figure Figure 5(c)).



(a) Tracking - STORE1 (b) Tracking - KITTI00 (c) Local Mapping - STORE1 (d) Local Mapping - KITTI00

(e) Local Mapping - STORE2 (f) Local Mapping - KITTI07 (g) Loop Closure - STORE2 (h) Loop Closure - KITTI07

Fig. 5: Experimental results: (a) and (b) present results using only the Tracking module with SIFT features; (c), (d), (e) and (f) present comparative results of the Tracking module with and without the Local Mapping module; (g) and (h) present comparative results of the Tracking module, the Tracking module with Local Mapping, and the Tracking module with Local Mapping and Loop Closure.

C. Loop Closure

The last set of experiments aims to evaluate the two implemented types of methods for error distribution along the trajectory (keyframe optimization and transformation averaging) and also make a final analysis of the evolution of the results obtained along the pipeline with the implementation of each module.

For the STORE2 dataset result in Figure 5(g), the transformation averaging method was able to close the loop correctly. This result is due to the trajectory not having sharp variations, so it is correct to assume for this case that the accumulated error is constant along with the keyframes. For the keyframe optimization method, keyframes close to the loop are corrected correctly, however, keyframes far from the loop are adjusted incorrectly.

Table II shows the error values obtained in the KITTI07 dataset for the Tracking module, Local Mapping module, and Loop Closure module using either keyframe optimization or transformation averaging for loop correction, and Figure 5(h) show the trajectories obtained using each of the modules (for the loop closure it is used keyframe optimization for loop correction). It can be seen that neither method was able to correctly correct the loop error. In the case of the transformation averaging method, this is mainly due to the error accumulated along the trajectory not being approximately constant or to the estimation of the correct position of the current keyframe in the loop. For the keyframe optimization method, there was also no correct loop correction. Although the absolute rotation error improved, the translation still has a high error. This may also be related to the estimation of the correct position of the current keyframe, which may be incorrect because there are too few established points between the keyframes in the loop, so the estimation does not produce a good result, or because the accumulated error is too large to be corrected in this way.

Overall, a clear evolution in the results is observed from using just Tracking to Local Mapping and Loop Closure.

VI. Conclusion

The developed SfM system is able, based on stereo images from pinhole or fisheye cameras, to estimate the trajectory traveled by the camera while creating a map of the traveled region. In the tracking module, several feature detectors and visual odometry estimation methods were tested. Hence, regarding the goal of developing a modular pipeline, it can be concluded that the goal has been achieved. Concerning the exploration of the structure of retail stores environments, the developed method showed potential, however, because it does not work for fisheye images and because in certain regions it cannot correctly estimate vanishing points, it was not able to

Setup	Relative Error				Absolute Error	
	Rot [rad]		Trans [m]		Rot [rad]	Trans [m]
	μ	σ	μ	σ		
Tracking	0.012	0.019	0.17	0.29	0.27	33.95
Local Mapping	**0.0022**	**0.0064**	**0.067**	**0.19**	0.21	**18.23**
Loop Closure (Transformation Averaging)	0.0031	0.016	0.081	0.30	0.19	23.76
Loop Closure (Keyframe Optimization)	0.0027	0.0099	0.075	0.22	**0.069**	21.07

TABLE II: Rotation and translation errors obtained on the KITTI07 dataset using Tracking, Local Mapping, and Loop Closure modules.

produce the desired results. As an alternative, another method was proposed, which, despite not exploiting the store structure, was able to produce good results. However, regarding the goal of exploring the geometrical structure of the store, the goal was incomplete. The last objective of creating an SfM pipeline based on a known SLAM system was mostly fulfilled, with some work still missing on the loop closure module since there was no correct loop closure in all the datasets tested, especially the larger ones.

Future work can be done throughout the pipeline, from improving or changing certain methods to testing parameters related to optimization, feature extraction, feature matching, and more. Regarding the Tracking module, one aspect that could improve the results is the integration of information from other sensors, such as IMU or encoders. This type of sensor can be used to give initial estimates int the visual odometry estimatation or used as the motion value when the visual odometry does not produce a correct estimate (*e.g.* by detecting few keypoints). Another improvement to be made involves all modules and consists of having better tracking of keypoints over several frames. By having a better tracking of the keypoints there will be more map points to be seen by several keyframes which consequently creates more connections between keyframes and improve the bundle adjustment optimization process. Overall, the whole system implemented should be optimized so that it runs in real-time and a localization module should also be implemented to transition from the implemented SfM system to a SLAM system.

REFERENCES

[1] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Trans. Robotics (T-RO)*, vol. 33, no. 5, pp. 1255–1262, 2017. 1, 2, 3

[2] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, 1992, pp. 586–606. 2

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. 2

[4] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European Conf. Computer Vision (ECCV)*, 2006, pp. 404–417. 2

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *IEEE Int'l Conf. Computer Vision (ICCV)*, 2011, pp. 2564–2571. 2

[6] P. F. Alcantarilla and T. Solutions, "Fast explicit diffusion for accelerated features in nonlinear scale spaces," *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 34, no. 7, pp. 1281–1298, 2011. 2

[7] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981. 2

[8] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966. 2

[9] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, pp. 376–380, 1991. 2

[10] C. Choy, J. Park, and V. Koltun, "Fully convolutional geometric features," in *IEEE Int'l Conf. Computer Vision (ICCV)*, 2019, pp. 8958–8966. 2

[11] Z. Li and N. Wang, "Dmlo: Deep matching lidar odometry," in *IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS)*, 2020, pp. 6010–6017. 2

[12] G. D. Pais, S. Ramalingam, V. M. Govindu, J. C. Nascimento, R. Chellappa, and P. Miraldo, "3dregnet: A deep neural network for 3d point registration," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7193–7203. 2

[13] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski, "Bundle adjustment in the large," in *European Conf. Computer Vision (ECCV)*, 2010, pp. 29–42. 2

[14] R. Hartley, J. Trumpf, Y. Dai, and H. Li, "Rotation averaging," *Int'l J. Computer Vision (IJCV)*, vol. 103, no. 3, pp. 267–305, 2013. 2

[15] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004. 3

[16] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Trans. Pattern Analysis and Machine Intelligence (T-PAMI)*, vol. 26, no. 6, pp. 756–770, 2004. 3

[17] X. Lu, J. Yaoy, H. Li, Y. Liu, and X. Zhang, "2-line exhaustive searching for real-time vanishing point estimation in manhattan world," in *IEEE Winter Conf. on Applications of Computer Vision (WACV)*, 2017, pp. 345–353. 3, 4