2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

ProSLAM: Graph SLAM from a Programmer's Perspective

Dominik Schlegel Mirco Colosi Giorgio Grisetti

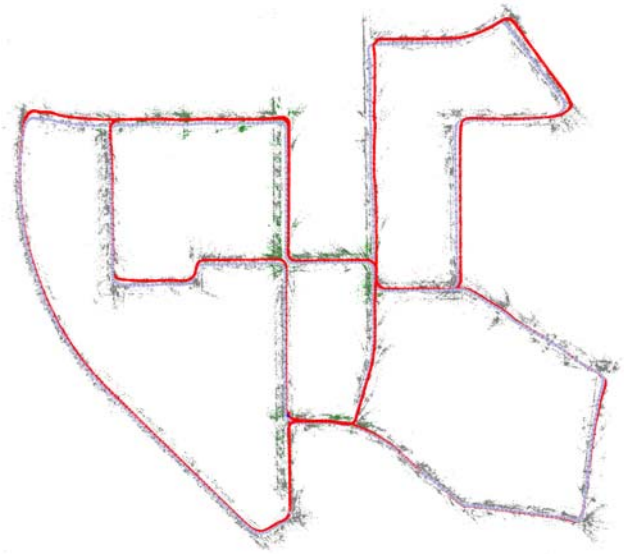
Abstract— **In this paper we present ProSLAM, a lightweight open-source stereo visual SLAM system designed with simplicity in mind. This work stems from the experience gathered by the authors while teaching SLAM and aims at providing a highly modular system that can be easily implemented and understood. Rather than focusing on the well known mathematical aspects of stereo visual SLAM, we highlight the data structures and the algorithmic aspects required to realize such a system. We implemented ProSLAM using the C++ programming language in combination with a minimal set of standard libraries. The results of a thorough validation performed on several standard benchmark datasets show that ProSLAM achieves precision comparable to state-of-the-art approaches, while requiring substantially less computation.**

I. Introduction

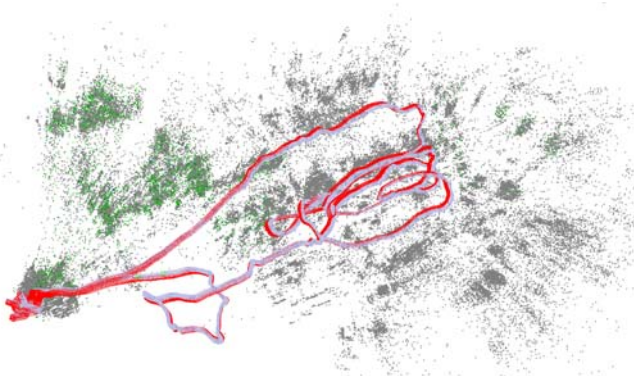
Simultaneous Localization and Mapping (SLAM) systems manage to deliver incredible results and after years of extensive investigation, the topic still captures the imagination of many young students and prospective researchers. Among others, S-LSD-SLAM [1], S-PTAM [2] and ORB-SLAM2 [3] are three prominent stereo visual SLAM approaches that are regarded as the state-of-the-art in the robotic community. The increasing sophistication of these systems generally comes at the price of more complex implementations which are difficult to fully understand and extend for people new to the field.

In this paper we present ProSLAM (Programmers SLAM), a complete open-source[1] stereo visual SLAM system that combines well known techniques and encapsulates them into a single pipeline with separated components and clear interfaces. We further provide multiple code snippets that realize the core functionality of our system. ProSLAM is implemented in pure C++ and relies on very few publicly available external libraries such as Eigen and OpenCV to perform basic operations. In contrast to our previous work [4] this paper focuses more on the concepts beyond the proposed architecture rather than on coding aspects.

Using a stereo camera relieves us from handling typical problems arising in the monocular case such as feature initialization and scale drift. We made this choice to limit the complexity of our system that is targeted for education. However, these two aspects can be addressed with minor modifications to the proposed pipeline. For pursuing simplicity, we substituted full bundle adjustment in the refinement of our map by structure-only optimization. Our experiments confirm that despite these simplifications, ProSLAM still manages to achieve state-of-the-art accuracy.



(a) UGV: KITTI Seq. 00, 3.7 km trajectory in urban environment.



(b) UAV: EuRoC MH_01_easy, machine laboratory indoor setting.

Fig. 1: Final maps generated by ProSLAM for a UGV and a UAV dataset in different environments using identical parameters. The reconstructed path from pure stereo image input is illustrated in blue, the respective ground truth in red. Point clouds associated to loop closures are highlighted in green.

Similar to ORB-SLAM2, our approach is feature based: it tracks a selection of features in the scene and thanks to the known geometry of the stereo cameras it determines the 3D position of the corresponding points. The image points tracked along multiple subsequent frames are grouped to form landmarks, that are salient points in the 3D space characterized by similar appearance. The landmarks observed along a small portion of the trajectory are grouped in small point clouds (local maps) and the local maps themselves

All authors are with the Department of Computer, Control and Management Engineering, Sapienza University of Rome, Rome, Italy {last name}@diag.uniroma1.it

[1]Source code: www.gitlab.com/srrg-software/srrg_proslam

are arranged in a pose graph. This pose graph [5] provides a deformable spatial backbone for the local maps and it grows as the robot explores the environment. Whenever the robot reenters a known location the graph is augmented with spatial constraints representing relocalization events and a new consistent configuration of the map is estimated through pose-graph optimization.

Albeit the different modules of our system could be easily parallelized, we present a single-threaded implementation, thus avoiding the complexity deriving from the need of synchronizing multiple threads and preserving the integrity of the memory.

We conducted comparative experiments on two standard stereo visual SLAM benchmarks. Each benchmark consists of multiple datasets acquired from heterogeneous platforms, namely cars and drones. Fig. 1 shows the outcome of our approach for two sequences of the KITTI and EuRoC benchmarks. With a straightforward, single-threaded pipeline our approach achieves precision comparable to the one of state-of-the-art algorithms, while requiring substantially less computation. Finally, we contribute to the community by providing an overseeable and complete stereo visual SLAM system that can compete with the state-of-the-art.

II. Related Work

One of the first online large-scale stereo visual SLAM system that appeared in literature is FrameSLAM. Konolige and Agrawal [6] introduced a complete feature based SLAM approach with bundle adjustment running in real-time. For their approach they use CenSure features and integrate IMU information into the odometry computation.

Another renown system in the domain of stereo visual SLAM is FABMAP [7], presented by Cummins *et al.* FABMAP is a purely appearance based Filter SLAM system using SURF features. The use of SURF that provide salient floating point descriptor vectors significantly contributes to the mapping precision, at the cost of an increased computation required to obtain these descriptors. Once the descriptors are computed, FABMAP can efficiently retrieve similar images for loop closure thanks to a visual vocabulary. The implementation is fully open-source and certain components have been integrated into the OpenCV library.

Pire *et al.* [2] presented a compact, appearance based stereo visual method S-PTAM. S-PTAM runs on 2 threads for achieving real-time computation. The first thread is in charge of tracking the position of the camera, and relies on BRIEF features, while the second thread performs anytime full bundle adjustment and is based on g2o [8]. In contrast to the other presented algorithms S-PTAM is not performing explicit relocalization and relies on full bundle adjustment for preserving the map consistency. This results in a computational complexity that grows with the size of the environment being mapped. The code of S-PTAM is publicly available.

Mur-Artal and Tardos [3] recently introduced the open-source stereo visual SLAM system ORB-SLAM2. The system originated from the prominent, monocular ORB-SLAM published by the same authors. ORB-SLAM2 achieves extraordinary performance thanks to a highly reliable tracking front end and frequent relocalization using ORB features. ORB-SLAM uses compact data structures to store the system state that partially inspired the architecture of ProSLAM. ORB-SLAM2 closes loops in real-time by utilizing a bag of words approach first proposed by Galvez-Lopez and Tardos [9]. Like S-PTAM, ORB-SLAM2 employs g2o for local bundle adjustment. The ORB-SLAM2 pipeline is designed to run on 3 parallel threads that handle tracking, relocalization and optimization.

S-LSD-SLAM proposed by Engel *et al.* [1] is a direct, hence featureless SLAM approach operating in large-scale at high processing speeds faster than real-time on a single thread. Engel exploits static and temporal stereo image changes at pixel level while also considering lightning changes. The system is able to perform relocalization and allows the integration of FABMAP components for loop closure detection. While LSD-SLAM is open-source, the code for the stereo version is not publicly available.

In contrast to these approaches, which aim at advancing the state-of-the-art at the cost of increasing the system complexity, ProSLAM is designed to be easy to understand and implement while maintaining state-of-the-art performance. This claim is backed up by the obtained results on standard benchmarks presented in Sec. IV.

III. Our Approach

The goal of ProSLAM is to process sequences of stereo image pairs to generate a 3D *Map*. This map should represent the environment perceived by the robot and support crucial functionality required in a SLAM system. The basic geometric entity that constitutes a map is a *Landmark*. A landmark is a salient 3D point in the world characterized by its position and its appearance. The appearance of a landmark is represented as the set of descriptors computed from each image that captures the landmark.

Landmarks acquired in a nearby region form a *Local Map*. A local map can be imagined as a point cloud where each point (landmark) can have multiple descriptors (appearances). Local maps are arranged spatially in a *Pose Graph*. Each node of a pose graph thus encodes a 3D isometry (rotation and translation), representing the origin of the corresponding local map in the world. Edges between local maps represent spatial constraints correlating local maps close in space. These constraints are generated either by *Tracking* the camera motion between temporally subsequent local maps or by aligning local maps acquired at distant times (*Loop Closure*) as a consequence of *Relocalization* events.

Relocalization is achieved by comparing the sets of descriptors of local maps and aligning the associated landmarks. Arranging the local maps in a pose graph allows us to utilize existing factor graph optimization engines for adjustment. Furthermore they enable us to limit the size of the adjustment problem, compared to a full bundle adjustment approach, substantially reducing the computational cost. Our

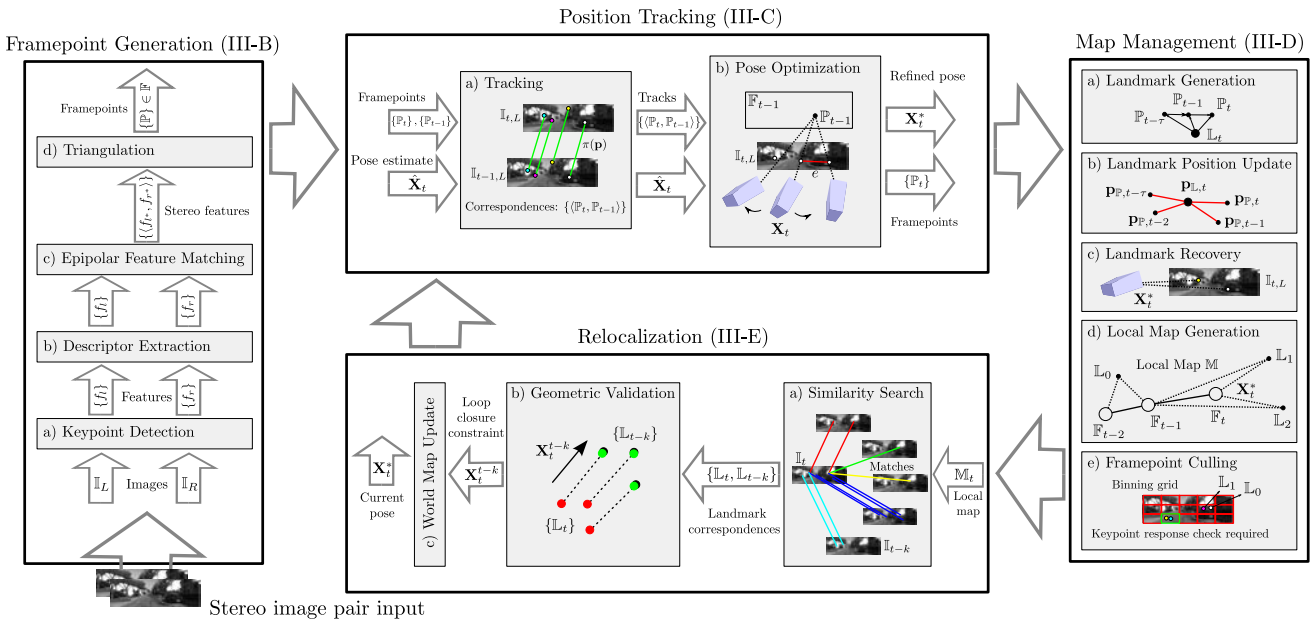


Fig. 2: Full ProSLAM system overview with the 4 core processing modules. The only inputs to the system are stereo images. A complete processing cycle is triggered for every stereo image pair fed to the framepoint generation module (bottom left). Each component is explained in detail from Sec. III-B throughout Sec. III-E. Note that the all modules except the framepoint generation are sensor-agnostic.

experiments show that this can be done with only minimal losses in precision.

Fig. 2 illustrates our complete SLAM pipeline consisting of the following 4 processing modules, that are executed sequentially for each stereo image pair in the input stream:

- Framepoint Generation (Sec. III-B) - takes a stereo pair of images as input and produces 3D points plus corresponding features for the left and right image.
- Position Tracking (Sec. III-C) - processes two subsequent image pairs and estimates the relative motion of the camera between the two instants.
- Map Management (Sec. III-D) - consumes the image pairs with known camera pose determined by position tracking and refines the landmark positions. Additionaly this module is in charge of assembling sufficiently large trajectory chunks into more compact local maps.
- Relocalization (Sec. III-E) - seeks if the current local map appears similar to some other local map generated in the past. Whenever a good match is found, it estimates the relative pose between the two maps and updates the entire world representation.

In the remainder of this section we outline the data structures of our system. Subsequently we describe each processing module in detail.

A. Data Structures

Fig. 3 shows the class diagram and the interactions between the data structures used to store the state of our system, while Fig. 4 depicts the evolution of the class instances at runtime. The only input to our system is a stream of stereo image pairs. For simplicity, we assume the images to be undistorted and rectified. This assumption typically holds for most standard stereo visual SLAM benchmarking datasets such as KITTI [10] and Málaga [11]. Should this not be the case, undistorted and rectified images can be easily obtained by using publicly available tools (e.g. OpenCV library).

The input *Images* are represented by a pair of 2D arrays containing intensity values. The value of a pixel at the *image coordinates* $(u, v)^\top$ lies in the interval $[0, 1]$. A *Feature* f is a data structure containing the image coordinates of an keypoint, the response of the keypoint and the descriptor computed at $(u, v)^\top$ on the image plane.

The detection of a salient 3D point *in a stereo image pair* is stored in a *Framepoint* $\mathbb{P}$. A framepoint contains the left and the right features f_l and f_r, corresponding to the estimated projections of the point in the two images. The subscripts $_{l,r}$ indicate the feature index in the left and right image. In addition to that, we also memorize the triangulated position of the point with respect to the left camera in $\mathbb{P}$.

A landmark $\mathbb{L}$ represents a salient 3D point *in the world*, and is part of the map. To support efficient incremental updates for the landmark pose estimate, in addition to its *world coordinates* $(x, y, z)^\top$, we store also the information matrix and information vector of the landmark's position estimate. A landmark keeps references to all consecutive image observations (i.e. framepoints $\{\mathbb{P}\}$) that observed it.

To speed up queries in the map, we augmented the above by adding to each framepoint, an optional reference to the landmark it observed. Multiple framepoints referring to the same landmark share the same landmark reference. Furthermore we organized all framepoints arising from the same landmark $\mathbb{L}$ in a temporally ordered doubly linked list (track). Traversing this list results in directly analyzing the subsequent detections of $\mathbb{L}$ in the scene.

All framepoints generated from a stereo image pair are stored in a *Frame* $\mathbb{F}$. $\mathbb{F}$ is part of a local map $\mathbb{M}$ and stores

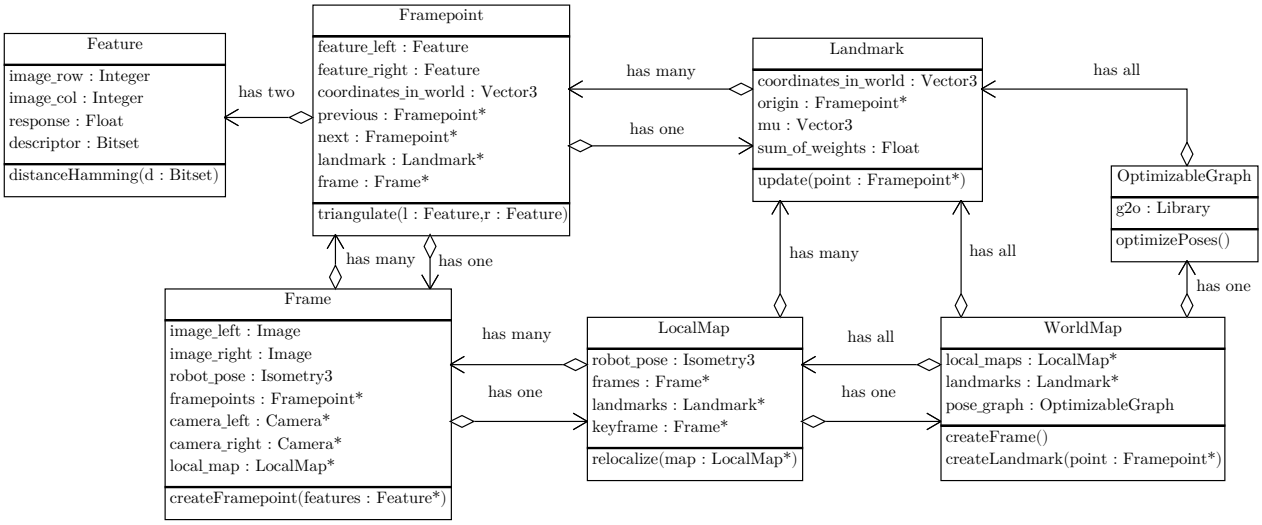


Fig. 3: Complete ArgoUML class diagram of the ProSLAM data structures. The diagram shows all direct aggregation relations between our class objects. Notable functions (e.g. factory methods) are mentioned in the corresponding class method list.

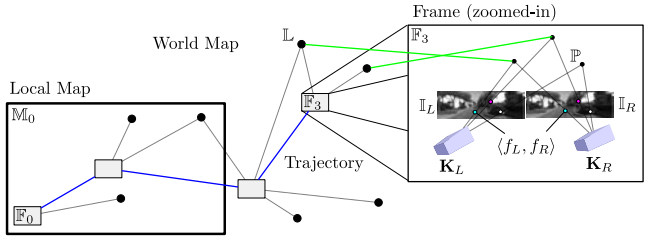


Fig. 4: Runtime snapshot of the ProSLAM data structures. The trajectory passing through the frame nodes $\{\mathbb{F}\}$ is highlighted in blue. The green lines link landmarks with their corresponding framepoints in the scene.

the relative camera position with respect to the local map.

A local map contains consecutive frames and references to all landmarks seen by the enclosed frames. Additionally, a local map stores its position in the world frame. The origin of a local map is selected to match the origin of one of the contained frames. That specific frame is called *Keyframe*.

The *World Map* is the global map container. It encapsulates the data manipulation functionality for all contained objects. These functions are designed to preserve the consistency of the bookkeeping between the object instances. Additionally the world map stores all spatial relations between local maps. These spatial relations arise either from visual odometry (position tracking) or from loop closing (relocalization) and constitute a pose graph. This structure is used to support global map optimization for the local maps after detecting loop closures.

B. Framepoint Generation

The goal of the *Framepoint Generation* module is to compute an evenly distributed cloud of salient 3D points for a scenery captured in a stereo image pair. Fig. 2 left illustrates the entire module with its inputs, outputs and subunits.

a) Keypoint Detection: To this extent we first perform a keypoint detection on both images, using the default FAST detector [12] implemented in OpenCV. The number of detected keypoints might vary significantly between subsequent frames. Thus in order to maintain an approximately constant number of detections, we dynamically adjust the FAST threshold based on the number of points detected in the preceding frame. If the number of detections is below the desired number, the threshold is lowered, otherwise it is increased within certain limits.

b) Descriptor Extraction: Once we have two sets of detected keypoints (left and right image), we determine the appearance for all keypoints by extracting their descriptors. In our current system we use standard, 256-bit sized BRIEF descriptors [13] implemented by OpenCV. They represent a reasonable choice for real-time applications due to their limited computation cost and low memory footprint.

c) Epipolar Feature Matching: At the end of the above procedure we have two sets of features: $\{f_l\}, \{f_r\}$. To determine the position of the salient points in 3D we need to identify which feature in the left image f_l corresponds to a feature in the right image f_r. Having rectified images, we can assume that corresponding features in the images lie on the same horizontal line (epipolar line). To find the corresponding stereo matches, we consider each feature in the left image f_l and seek along the epipolar line in the right image for the best matching feature f_r, defined by the smallest descriptor distance.

Upon lexicographical ordering of the feature positions in the images (first by row, and then by column index), the search procedure can be executed in a time linear in the number of features, as sketched in Alg. 1. We refer the readers interested in an implementation of the search to the online snippet[2]. The H function mentioned in Alg. 1 returns the *Hamming* distance between two descriptors of a feature pair (common for BRIEF). Upon completion, the search returns a set of stereo feature matches $\{\langle f_{l^*}, f_{r^*} \rangle\}$.

[2]Epipolar matching: `www.gitlab.com/srrg-software/srrg_proslam/snippets/1670504`

Algorithm 1 $\{\langle f_{l^*}, f_{r^*}\rangle\}$ = getEpipolarMatches($\{f_l\}, \{f_r\}$)

Require: $\{f_l\}, \{f_r\}$: sets of features from left and right image
Require: N_L, N_R: number of features in the left resp. right image
Require: d_{max}: maximum descriptor matching distance threshold
Ensure: $\{\langle f_{l^*}, f_{r^*}\rangle\}$: set of epipolar (stereo) feature matches
  //sort input vectors in ascending keypoint u, v coordinates
  sort($\{f_l\}, (u_{a,l} < u_{b,l})$ **or** $(u_{a,l} == u_{b,l}$ **and** $v_{a,l} < v_{b,l}))$
  sort($\{f_r\}, (u_{a,r} < u_{b,r})$ **or** $(u_{a,r} == u_{b,r}$ **and** $v_{a,r} < v_{b,r}))$
  for $l = 0 : N_L - 1, r = 0$ **do**
    while $u_l > u_r$ **and** $r < N_R$ **do**
      $++\, r$ //proceed until RHS is on equal u coordinate
    end while
    $r_{epi} = r$ //check the epipolar line for matches on the RHS
    while $u_l == u_{r_{epi}}$ **and** $v_{r_{epi}} \leq v_l$ **and** $r_{epi} < N_R$ **do**
      $d = H(f_l, f_{r_{epi}})$ //descriptor distance (Hamming)
      if $d < d_{best}$ **then**
        $d_{best} = d, r_{best} = r_{epi}$ //bookkeep best match
      end if
      $++\, r_{epi}$ //evaluate next candidate on epipolar line
    end while
    if $d_{best} < d_{max}$ **then**
      $\{\langle f_{l^*}, f_{r^*}\rangle\} \leftarrow \langle f_l, f_{r_{best}}\rangle$ //add new match
      $r = r_{best}$ //update RHS search index
    end if
  end for
  return $\{\langle f_{l^*}, f_{r^*}\rangle\}$

d) Triangulation: For each stereo match $\langle f_L, f_R\rangle$ we have the corresponding pair of epipolar image points:

$$\left\langle (u_L, v_L)_i^\top, (u_R, v_R)_i^\top \right\rangle \tag{1}$$

From Eq. (1) we can easily compute the 3D position of a point $\mathbf{p}_i = (x, y, z)^\top$ in camera frame through a standard closed-form stereo triangulation [14]. Once all stereo feature matches have been processed for triangulation, the module constructs a frame $\mathbb{F}$ by assembling all gathered information into a set of framepoints $\{\mathbb{P}\}$.

C. Position Tracking

This module estimates the motion of the left camera between two consecutive frames. It first seeks for pairs of framepoints in the previous frame and in the current frame, that are likely to originate from the same point in the world. Subsequently, given a set of framepoint pairs, it determines the current pose of the frame $\mathbf{X}_t$ at time t based on the previous pose $\mathbf{X}_{t-1}$ at time $t-1$. In this paper we will assume all transforms to be represented as homogeneous matrices of the form:

$$\mathbf{X} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}, \tag{2}$$

with $\mathbf{R}$ being a 3×3 matrix and $\mathbf{t}$ a translation vector.

a) Tracking: With reference to Fig. 2, this unit is in charge of finding the correspondences between consecutive framepoints (tracks). The core idea is to seek for points in the current frame whose appearance is similar to the corresponding points in the previous frame *and* whose location in the image is close to the expected location.

Our system proceeds by first computing the expected location of the reprojected points of a previous frame $\mathbb{F}_{t-1}$ in the image of the current frame $\mathbb{F}_t$. We assume the platform velocity to be nearly constant between subsequent frames. Let $\mathbf{X}_{t-2}^{t-1}$ be the motion between the two preceding frames $\mathbb{F}_{t-2}$ to $\mathbb{F}_{t-1}$, we compute a guess of the current left camera pose $\hat{\mathbf{X}}_t$ as:

$$\hat{\mathbf{X}}_t = \mathbf{X}_{t-2}^{t-1}\mathbf{X}_{t-1}. \tag{3}$$

Here $\mathbf{X}_{t-1}$ represents the left camera pose estimate for the previous frame. When a reliable estimate of the motion is not available, either because the system is initializing or a failure occurs we set $\mathbf{X}_{t-2}^{t-1}$ to the identity.

Knowing $\hat{\mathbf{X}}_t$, we can reproject each previous framepoint $\mathbb{P}_{i,t-1} \in \mathbb{F}_{t-1}$ onto the image plane of the current frame $\mathbb{F}_t$. This is feasible since we calculated the 3D position $\mathbf{p}_{i,t-1}$ of each framepoint within the triangulation unit (Sec. III-B). The reprojection provides a guess for the expected location of each $\mathbb{P}_{i,t-1}$ in $\mathbb{F}_t$:

$$(\hat{u}_L, \hat{v}_L)_{i,t|t-1}^\top = \pi(\mathbf{K}_L \cdot \hat{\mathbf{X}}_t \cdot \mathbf{p}_{i,t-1}). \tag{4}$$

Here $\pi(\mathbf{p}) = (x/z, y/z)^\top$ represents the homogeneous division and $\mathbf{K}_L$ the left camera calibration matrix.

To obtain the corresponding framepoints in two subsequent frames $\{\langle \mathbb{P}_{i,t-1}, \mathbb{P}_{j,t}\rangle\}$, we perform a local search in the current frame within a small region around each prediction $(\hat{u}_L, \hat{v}_L)_{i,t|t-1}^\top$. Inside that region, we match the closest measured image point $(u_L, v_L)_{j,t}^\top$ of $\mathbb{P}_{j,t}$, whose descriptor distance to $\mathbb{P}_{i,t-1}$ is still within a certain threshold. A snippet of the above procedure can be found online[3].

b) Pose Optimization: Given a set of correspondences $\{\langle \mathbb{P}_{i,t-1}, \mathbb{P}_{j,t}\rangle\}$ we refine the current guess of the left camera pose $\hat{\mathbf{X}}_t$, by looking for a transform $\mathbf{X}_t^*$ that minimizes the reprojection error between the corresponding points in the left and right image:

$$\mathbf{e}_{ij}(\mathbf{X}_t) = \begin{pmatrix} \pi(\mathbf{K}_L \cdot \mathbf{X}_t \cdot \mathbf{p}_{i,t-1}) - (u_L, v_L)_{j,t}^\top \\ \pi(\mathbf{K}_R \cdot \mathbf{X}_t \cdot \mathbf{p}_{i,t-1}) - (u_R, v_R)_{j,t}^\top \end{pmatrix}. \tag{5}$$

Here $\mathbf{p}_{i,t-1}$ denotes the position of $\mathbb{P}_{i,t-1}$ in world coordinates and $\left(u_{L/R}, v_{L/R}\right)_{j,t}^\top$ represents the left and right image coordinates of the corresponding $\mathbb{P}_{j,t}$. We use a robustified nonlinear least squares approach to minimize the cumulative squared sum of all reprojection errors:

$$\mathbf{X}_t^* = \underset{\mathbf{X}_t}{\operatorname{argmin}} \sum_{\{\langle \mathbb{P}_{i,t-1}, \mathbb{P}_{j,t}\rangle\}} \left[\mathbf{e}_{ij}^\top(\mathbf{X}_t) \cdot \mathbf{\Omega}_{ij} \cdot \mathbf{e}_{ij}(\mathbf{X}_t)\right] \tag{6}$$

for two sets $\{\mathbb{P}_{i,t-1}\}, \{\mathbb{P}_{j,t}\}$ of subsequent frames $\mathbb{F}_{t-1}, \mathbb{F}_t$. Since we perform a transformation $\mathbf{X}_t \cdot \mathbf{p}_{i,t-1}$ in conjunction with a projection operation $\pi(\mathbf{p})$ in Eq. (5), we obtain a Jacobian in chained form:

$$\mathbf{J}_{ij} = \begin{bmatrix} \mathbf{J}_{\pi,L}(\mathbf{X}_t) \\ \mathbf{J}_{\pi,R}(\mathbf{X}_t) \end{bmatrix} \mathbf{J}_{\mathbf{X}}(\mathbf{X}_t). \tag{7}$$

$\mathbf{J}_{\pi,L/R}(\mathbf{X}_t)$ are the Jacobians resulting from the left respectively right image projection and $\mathbf{J}_{\mathbf{X}}(\mathbf{X}_t)$ is the Jacobian of the 3D point transformation. Alg. 2 summarizes our least squares pose estimator.

[3]Framepoint tracking: www.gitlab.com/srrg-software/srrg_proslam/snippets/1670635

To compute the reprojections, the algorithm evaluates whether framepoints are connected to landmarks and if so, reprojects the landmark position $\mathbf{p}_{\mathbb{L},i}$ instead of the framepoint position $\mathbf{p}_{\mathbb{P},i}$. At each iteration we compute a 6D perturbation $\mathbf{\Delta x}$ of the current solution $\mathbf{X}_t$. The perturbation consists of a 3D translation vector and three components of a unit quaternion. It is applied to the current solution through the $\boxplus$ operator, as explained in [5]. Once the optimization has finished, the current frame with its refined pose estimate $\mathbf{X}_t^*$ is passed to the next module. A snippet of Alg. 2 is available online[4]. As a rule of thumb, using 1'000 framepoints $\{\mathbb{P}_{j,t}\}$ per frame is a reasonable tradeoff between processing speed and accuracy for Alg. 2.

Algorithm 2 $\mathbf{X}_t^*$ = getOptimizedPose($\{\mathbb{P}_{i,t-1}\}, \{\mathbb{P}_{j,t}\}, \hat{\mathbf{X}}_t$)

Require: $\{\mathbb{P}_{i,t-1}\}$: set of framepoints from previous image pair
Require: $\{\mathbb{P}_{j,t}\}$: set of framepoints from current image pair
Require: $\hat{\mathbf{X}}_t$: initial pose estimate
Require: e_{max} : maximum kernel error (squared, in pixels)
Require: d_{near} : maximum depth for near points (3D information)
Require: d_{max} : maximum measurable depth
Require: λ : damping factor
Ensure: $\mathbf{X}_t^*$: final camera pose estimate
  $\mathbf{X}_t = \hat{\mathbf{X}}_t$ //initialize pose for first iteration, Eq. (3)
  for desired number of iterations **do**
    $\mathbf{H} = \mathbf{0}_{6\times6}, \mathbf{b} = \mathbf{0}_{6\times1}$ //linearization buffers
    for all $\mathbb{P}_{j,t} \in \{\mathbb{P}_{j,t}\}$ **do**
      //if we have a track for the current framepoint
      if $\mathbb{P}_{i,t-1} \in \{\langle \mathbb{P}_{i,t-1}, \mathbb{P}_{j,t} \rangle\}$ **then**
        //if the framepoint is connected to a landmark
        if $\mathbb{P}_{i,t-1} \rightarrow \mathbb{L}$ **then**
          $\mathbf{p}_{i,t-1} \leftarrow \mathbf{p}_{\mathbb{L},j}, \mathbf{\Omega}_i = \mathbf{\Omega}_{\mathbb{L}}$ //use landmark, Eq. (5)
        else
          $\mathbf{p}_{i,t-1} \leftarrow \mathbf{p}_{\mathbb{P},i}, \mathbf{\Omega}_i = \mathbf{I}_{3\times3}$ //use framepoint, Eq. (5)
        end if
        if $\mathbf{e}_{ij}^\top(\mathbf{X}_t)\,\mathbf{e}_{ij}(\mathbf{X}_t) > e_{max}$ **then**
          $\mathbf{\Omega}_{ij} \cdot= \mathbf{e}_{ij}^\top(\mathbf{X}_t)\,\mathbf{e}_{ij}(\mathbf{X}_t)\,/e_{max}$ //saturation kernel
        end if
        if $z_{j,t} < d_{near}$ **then**
          $\mathbf{\Omega}_{ij} \cdot= \frac{d_{near}-z_{j,t}}{d_{near}}$ //relative weighting near
        else
          $\mathbf{\Omega}_{ij} \cdot= \frac{d_{max}-z_{j,t}}{d_{max}}$ //relative weighting far
        end if
        $\mathbf{H} \mathrel{+}= \mathbf{J}_{ij}^\top(\mathbf{X}_t) \cdot \mathbf{\Omega}_{ij} \cdot \mathbf{J}_{ij}(\mathbf{X}_t)$ //see Eq. (7)
        $\mathbf{b} \mathrel{+}= \mathbf{J}_{ij}^\top(\mathbf{X}_t) \cdot \mathbf{\Omega}_{ij} \cdot \mathbf{e}_{ij}(\mathbf{X}_t)$
      end if
    end for
    $\mathbf{\Delta x} = -(\mathbf{H} + \lambda \mathbf{I}_{6\times6})^{-1}\,\mathbf{b}$ //compute perturbation
    $\mathbf{X}_t = \mathbf{X}_t \boxplus \mathbf{\Delta x}$ //update estimate
  end for
  return $\mathbf{X}_t^* = \mathbf{X}_t$ //return final camera pose estimate

D. Map Management

The *Map Management* module governs the generation and maintenance of landmarks $\{\mathbb{L}\}$ and local maps $\{\mathbb{M}\}$. It incorporates the high-level information of an individual frame $\mathbb{F}$ that was gained from the two preceding modules.

[4]Stereo pose optimization: www.gitlab.com/srrg-software/srrg_proslam/snippets/1670637

a) Landmark Generation: Whenever the same framepoint $\mathbb{P}$ is detected for a minimum number of consecutive frames, a new landmark $\mathbb{L}$ is created. Its pose estimate is initialized by taking into account all observations and the corresponding estimated camera poses. A landmark is present in the current scene as long as it is continuously observed. Hence, a landmark $\mathbb{L}$ with *age* τ combines the position information of a continuous track of framepoints $\langle \mathbb{P}_t, \mathbb{P}_{t-1}, .., \mathbb{P}_{t-\tau} \rangle$.

b) Landmark Position Update: As the system progresses, a landmark's position $\mathbf{p}_{\mathbb{L},t} = (x, y, z)^\top$ is subsequently updated for each newly found framepoint position $\mathbf{p}_{\mathbb{P},t}$ associated to it. In this context $\mathbf{p}_{\mathbb{P},t}$ is referred to as a measurement with weight $w_t = \frac{1}{z}$ (inverse depth). The measurement integration into $\mathbf{p}_{\mathbb{L},t}$ is based on a straightforward *running information filter* update:

$$\mathbf{p}_{\mathbb{L},t} = \frac{\nu_t}{W_t}, \quad \nu_t = \sum_{i=t-\tau}^{t} w_i \mathbf{p}_{\mathbb{P},i}, \quad W_t = \sum_{i=t-\tau}^{t} w_i \tag{8}$$

Here ν_t is the sum of all *weighted* measurements and W_t is the sum of all weights at time t.

c) Landmark Recovery: Since our tracking input is driven by the keypoint detector used in the framepoint generation (Sec. III-B), it might happen that some points are not reported by the detector although they are visible in the image. Therefore valuable tracks might be lost and with them the associated landmarks. However our system can recover lost landmarks. This is done by projecting the lost landmarks onto the current image plane and seeking for a descriptor in the current frame at that location. If a sufficiently similar descriptor is found we recover the lost track with its landmark.

d) Local Map Generation: We generate a new local map each time the robot has traveled a certain distance since the last local map was created, or the position tracking (Sec. III-C) fails. As $\mathbb{M}_t$ is created, the current frame becomes a keyframe, determining the pose of $\mathbb{M}_t$. All other frames collected since the last generation are stored in $\mathbb{M}_t$.

e) Framepoint Culling: In order to maintain a homogeneous framepoint density - and consequently landmark density - over the image, we discard new framepoints that would create a cluttering effect. To this extent we divide the image into bins. For each bin that is occupied by framepoints without links to previous ones (tracks), only the framepoint with the highest response in the bin is kept. Tracked framepoints are always kept regardless of their response. Additionally, we discard framepoints that did not pass the robust kernel of the pose optimization (Alg. 2).

E. Relocalization

The modules described so far (Sec. III-B to III-D) constitute a stereo visual odometry pipeline and as such its pose estimate is affected by unavoidable drift. This becomes particularly evident when revisiting already seen places (loop closures). One can compensate for the drift by relocalizing in a previous portion of the map. This would introduce a "jump"

in the trajectory, thus rendering it inconsistent with respect to the pose of the previous frame estimated by position tracking. To recover a trajectory that respects all constraints we solve a graph optimization problem [5]. Each node of the graph represents the global position of a local map. Edges represent a relative transformation between local maps and are treated as measurements within the optimization process. An edge can arise either from position tracking (Sec. III-C) or relocalization. Optimizing the graph results in finding the arrangement of local maps that is maximally consistent with the measurements. With reference to Fig. 2, we outline the details of our relocalization procedure as follows:

a) Similarity Search: The goal of the similarity search unit is to identify past observations that are likely to be taken at the current location. A naive approach to determine if two observations appear similar is to count the number of similar descriptors between them. A straightforward implementation would result in an N^2 computational cost, where N is the average number of descriptors in a frame. This search has to be carried out between the current frame and *all* frames in the past, and its computational time will quickly grow, preventing real-time operation. In ProSLAM we propose an efficient approach for this search by reducing the complexity of frame-to-frame matching to a time logarithmic in the number of descriptors N. To this extent we use one of our previous works, the open-source *Hamming binary search tree* library HBST [15]. The HBST library returns a set of descriptor matches by performing a tree search.

Instead of performing the similarity search for each new frame, we execute this operation only for each local map. This has two advantages: first, the number of local maps grows much slower than the number of frames, hence reducing the number of HBST queries, and second, the local maps capture larger portions of the environment that render them more distinctive than single frames. Assuming we have a total number of K local map candidates, we obtain descriptor associations for each matching pair $\langle \mathbb{M}_t, \mathbb{M}_{t-k} \rangle$.

b) Geometric Validation: This unit aims at finding a consistent transform that maximizes the overlap between two locations which are reported to match by the similarity search. If no such a transform is found the correspondence is discarded. For each pair of corresponding local maps our similarity search reports a set of associations between the landmarks of the two local maps $\{\langle \mathbb{L}_{t,i}, \mathbb{L}_{t-k,j} \rangle\}$. One can use the *Horn formula* or its iterative counterpart [16] to find the relative transform $\mathbf{X}_t^{t-k}$ between $\mathbb{M}_t$ and $\mathbb{M}_{t-k}$. In ProSLAM we use an iterative solver that can find a solution within few iterations. We determine the quality of the solution by counting how many correspondences are satisfied by $\mathbf{X}_t^{t-k}$. A correspondence is considered good if the euclidean distance between the associated landmarks after applying the transform remains within a certain threshold d_{max}:

$$\left\| \mathbf{X}_t^{t-k} \mathbb{L}_{t,i} - \mathbb{L}_{t-k,j} \right\|^2 < d_{max}. \tag{9}$$

If the number of good correspondences is high enough, we assume the two local maps capture the same portion of the environment and we add a loop closure edge with the transform $\mathbf{X}_t^{t-k}$ to the graph.

c) World Map Update: Whenever one or more loop closures are added to the graph, we refine the position of all local maps to incorporate the newly added constraint by using the g2o [8] library. The output of the optimizer is a set of refined positions for all local maps $\{\mathbf{X}_0^*, \mathbf{X}_1^*, .., \mathbf{X}_t^*\}$ that are then applied to our world representation. Similarly, the landmark positions are adjusted based on the new pose estimates of the local maps.

IV. Experimental Evaluation

We compare the performance of ProSLAM with the stereo visual SLAM systems S-LSD-SLAM [1], S-PTAM [2] and ORB-SLAM2 [3]. For this purpose we use the two standard benchmarks KITTI [10] and EuRoC [17], that provide the ground truth of the robot trajectory. KITTI is representative for autonomous driving applications and captures urban and highway environments. EuRoC is acquired with a quadrotor in indoor environments. Our system runs with identical parameters throughout all sequences of a particular benchmark. In the remainder of this section we report a system precision and runtime comparison of all approaches for each evaluated sequence of the two benchmarks.

a) Precision: In Fig. 5 we report the relative translational (%) and rotational errors (deg/100m) obtained by running the evaluation system of the KITTI benchmark. We processed the datasets with ProSLAM and S-PTAM. For S-PTAM we utilized the configuration files provided by the authors. For S-LSD-SLAM and ORB-SLAM2 we report the values published by the authors in [1], [3].

From Fig. 5 we can see that all tested approaches achieve a relative translation error of less than 1% on average expect for Sequence 01. This sequence features a challenging highway scenario with very few near features to track. S-PTAM encountered difficulties in 01, resulting in overflowing error values which have been cut off in the figure. On the KITTI benchmark ProSLAM outperforms S-LSD-SLAM and S-PTAM in most cases while going head-to-head with ORB-SLAM2. The same holds for the rotation errors that remain well under 0.5 degrees over 100 meters of trajectory.

In Tab. I we report the absolute root mean squared error (RMSE) for the EuRoC sequences.

Dataset	S-LSD-SLAM	ORB-SLAM2	S-PTAM	ProSLAM
V1_01	0.066	**0.035**	2.008 (1.999)	0.092
V1_02	0.074	**0.020**	1.640 (1.910)	0.136
V1_03	0.089	**0.048**	1.494 (1.655)	0.860
MH_01	-	**0.035**	-	0.045
MH_02	-	**0.018**	-	0.044
MH_03	-	**0.028**	-	0.237

TABLE I: EuRoC benchmark. Absolute translation RMSE. For S-PTAM, the values in brackets are obtained using the default KITTI parameters. In the case of S-LSD-SLAM and ORB-SLAM2 we report the values presented in [1], [3].

The holonomic motions in EuRoC, often accompanied by motion blur and rapid illumination changes, require reliable visual tracking. S-PTAM looses track rather frequently as a consequence of these factors, resulting in a large error and

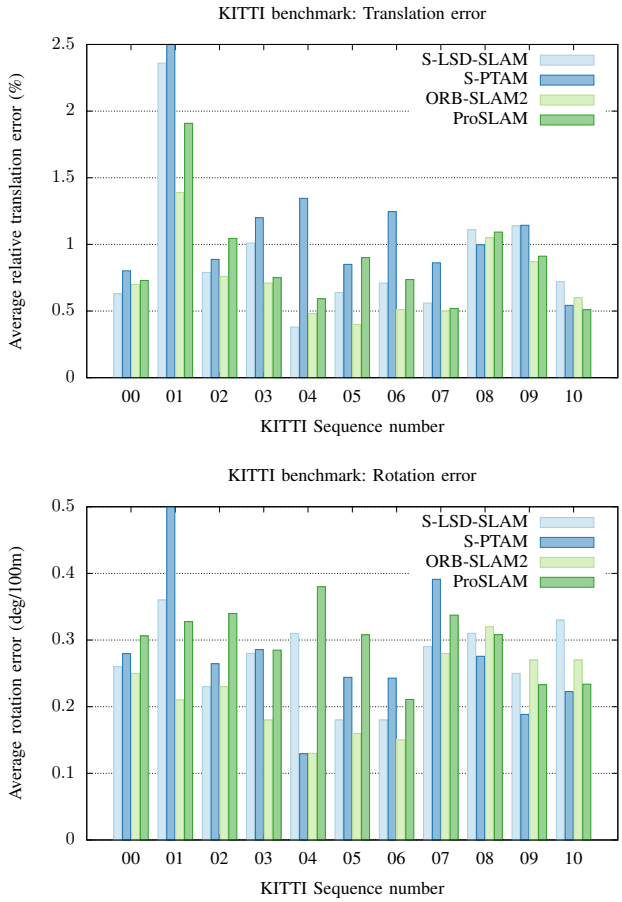


Fig. 5: KITTI benchmark. Rotational and translational error. Sequences 00, 02, 06, 07 and 09 contain loop closures.

a visually inconsistent map. Being a featureless approach, S-LSD-SLAM can deal better with these issues and ORB-SLAM2 is the most precise in the comparison. Despite the challenging maneuvers, ProSLAM always produced visually consistent maps in all reported sequences.

b) Runtime: To measure system runtimes we executed ProSLAM, S-PTAM and ORB-SLAM2 on the same computer equipped with an Intel i7-4770T CPU (4 cores @3.7GHz, 8MB cache), running Ubuntu 16.04. For S-LSD-SLAM we report the numbers of [1], where a more powerful Intel i7-4900MQ CPU (4 cores @3.8GHz, 8MB cache) was used. This results in a slight bias in favour of S-LSD-SLAM in the runtime comparison.

Approach	Source lines	# Threads	Mean (s)	Freq. (Hz)
S-LSD-SLAM	-	1	0.067	15
S-PTAM	9'369	2	0.037	27
ORB-SLAM2	10'821	3	0.062	16
ProSLAM	4'946	**1**	**0.018**	**56**

TABLE II: Size of the source code and average per frame processing speed over 10 runs on each KITTI sequence.

In Tab. II we report the time required to process a frame, averaged over the 11 KITTI sequences. We show that our single-threaded implementation is significantly faster than all compared approaches. Regarding the individual time spent in each of our modules, for 70% accounts the framepoint generation, 20% position tracking, 7% map management and 3% relocalization. Note that S-PTAM runs on 2 and ORB-SLAM2 on 3 threads. All systems are implemented in C/C++ and we report, where available, the size of the source code measured with the *cloc* source code analyzer.

V. Conclusion

In this paper, we presented ProSLAM, a stereo visual SLAM system developed to be extended and used both as a learning tool, and as a framework to improve particular SLAM aspects. ProSLAM achieves high computational speed by using a simple yet effective combination of algorithms and data structures. In terms of accuracy, our approach is comparable to more sophisticated systems. Our statements are supported by comparative experiments on standard benchmarking datasets. All results of this paper can be reproduced by running the open-source implementation, spanning over about 5'000 lines of documented C++ code.

References

[1] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct SLAM with stereo cameras," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[2] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. J. Berlles, "S-PTAM: Stereo Parallel Tracking and Mapping," in *Journ. of Rob. & Aut. Systems*, vol. 93, pp. 27–42, 2017.

[3] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Trans. on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] D. Schlegel, M. Colosi, and G. Grisetti, "ProSLAM: Graph SLAM from a Programmer's Perspective," 2017.

[5] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Trans. on Intelligent Transportation Systems*, 2010.

[6] K. Konolige and M. Agrawal, "FrameSLAM: From bundle adjustment to real-time visual mapping," *IEEE Trans. on Robotics*, 2008.

[7] A. J. Glover, W. P. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, "OpenFABMAP: An open source toolbox for appearance-based loop closure detection.," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[8] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[9] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Trans. on Robotics*, 2012.

[10] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[11] J.-L. Blanco, F.-A. Moreno, and J. Gonzalez-Jimenez, "The Málaga Urban Dataset: High-rate Stereo and Lidars in a realistic urban scenario," *Int. Journal of Robotics Research*, vol. 33, no. 2, pp. 207–214, 2014.

[12] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. of the Eur. Conf. on Computer Vision (ECCV)*, 2006.

[13] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proc. of the Eur. Conf. on Computer Vision (ECCV)*, 2010.

[14] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[15] D. Schlegel and G. Grisetti, "Visual localization and loop closing using decision trees and binary features," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[16] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, 1992.

[17] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *Int. Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.