SAPIENZA
UNIVERSITÀ DI ROMA

A Comparison of Features Descriptors on Tracking in Monocular Visual SLAM

Facoltà di ingegneria dell'informazione, informatica e statistica
Master's degree in Artificial Intelligence and Robotics

Luca Lobefaro
ID number 1890195

Advisor
Prof. Giorgio Grisetti

Academic Year 2021/2022

A Comparison of Features Descriptors on Tracking in Monocular Visual SLAM
Tesi di Laurea Magistrale. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: luca.lobefaro95@gmail.com

To mom and dad
best pieces of my life.

Abstract

This thesis has two purposes: the first is to propose a novel monocular visual-SLAM system that is efficient and able to produce good results in trajectory prediction, the second is to test how different image features affect the tracking.
In particular, the SLAM system is developed from scratch, using different techniques from literature, with great focus on initialization and tracking. It does not have the aim to produce results that are better than existing systems, but instead to produce a good baseline on which to determine what is the drift accumulated on various sequences using different features. This while keeping the algorithm light and easy to compute.
The features explored are three: we have ORB as baseline and representative of classical approaches and then Superpoint and LF-NET are used to understand how good neural features can be on this kind of system. Actually, it is shown in the results that neural features outperform ORB in almost any case. In particular, the proposed system, in combination with Superpoint, is able to maintain a good tracking in every sequence tested, even in those that are very difficult, because full of noisy images and motion blur.

Acknowledgments

I want to firstly thank my advisor Giorgio Grisetti for his support, availability and guidance during my thesis project. The same thanks go to Emanuele Giacomini, in particular for his constant advice and his patience.
Special thanks go to my parents, rarely those you find are the same ones you would choose, they are. They always believed in me, and that made all the difference.
Thanks to Francesca, her light accompanied me perpetually during these years, too many moments have turned into good memories thanks to her, even the worst candidates. And I thank her for the unconscious inspiration she gave me for my works.
Lastly, I want to thank all my friends, all the people and professors that I met on this path, for the richness of experiences, for the variety of ideas, they allowed me to grow as a student and as a man.

Contents

Chapter 1

Introduction

The ability of moving in an unknown environment, building a representation of it, while being able to understand the actual position and orientation is common to almost all intelligent species. For this reason, having a robot that is capable to perform such a task is a major research focus for mobile robotics, especially if we consider as goal to obtain an intelligent agent. Reaching this purpose without using an external tool or reference systems (like GPS) but only relying on on-board sensors (like a camera) is known as the Simultaneous Localization And Mapping (SLAM) problem. Its importance is deductible from the large quantity of application that formulate this problem and try to solve it, both on research and industrial settings: rovers for space exploration, vacuum cleaner, surveillance with unmanned air vehicles, reef monitoring, exploration under dangerous situations and so on. In particular, the ability of doing this with cheap sensors like monocular cameras is to great interest for different reasons linked to production costs. From this the importance of monocular visual-SLAM.
In literature different approaches are proposed, in this thesis the goal is to develop, from scratch, a novel monocular visual-SLAM system, that is both efficient and able to produce good tracking of video sequences. In particular, different ideas and proposal from research literature are adopted to reach this purpose, with a focus on initialization, for its delicacy, and tracking. This last stage is then analyzed with different features extracted from input images, in order to understand how different approaches can change the results. Indeed, the initial step of visual-SLAM is to extract, from video sequences, a set of representative points of interest, one set per each frame, that will then be used in all the algorithm as substitutes of the images. In particular a set of 2D points and corresponding features descriptor are needed and classical approach like ORB [45], SIFT [33] and SURF [2] are used in general. In particular, in this thesis the ORB features are used for two main reasons: they are not patented and are very fast to compute.
Artificial neural networks are revolutionizing the field of machine learning for several years, showing that it is possible to obtain algorithms that are even better than human in specific cases. In particular, this is obvious with images, where convolutional neural networks are reaching impressive results in different tasks. For this reason, different researcher began to explore the use of neural networks in feature extraction for visual-SLAM and related fields. Two approaches are to great interests: Superpoint [12] and

LF-NET [43]. Because of the lack of specific tests on how these approaches influence the tracking of a monocular visual-SLAM system, with a baseline constructed exactly for this purpose, the second aim of this thesis is to test them, applied on the original system proposed, on different sequences. They are compared with each other and with ORB features. We expect that neural features, able to "understand" better the kind of features that they are extracting, outperforms at least the baseline with ORB and perform better on more complex sequences.

Chapter 2

Related Works

The simultaneous localization and mapping problem (commonly abbreviated as SLAM), firstly proposed by Smith in 1986 [46], is one of the most fundamental problems in robotics. It can be viewed as a combination of other two fields of robotics: localization and mapping. The first one consists in predicting the actual pose of a robot in a known environment, given exteroceptive and proprioceptive measurement from its sensors, together with the history of poses. The mapping problem, instead, concerns the estimation of a unknown map, perceived from on-board sensors. The SLAM problem arises when neither the robot's poses nor the map is known, so them must be estimated "simultaneously".
Simultaneous localization and mapping is a challenging problem for these main reasons [52]:

- The space of all possible maps is huge, since their are defined over a continuous space and the corresponding hypothesis space has infinitely many dimensions. For this reason a probabilistic approach is needed.
- It is a "chicken-and-egg problem", because when the robot moves it predicts its pose basing on a set of noisy measurements, accumulating uncertainty and then it must use these predictions to estimate a set of landmark poses, based on measurements that are noisy, again.

For these reasons a huge quantity of research has been produced in the field, also because the possibility of having a mobile robot that can operate in complex environments relying on only its on-board sensors, is tempting for its large amount of applications: transportation, search and rescue robots, automated vacuum cleaning robots, rovers and landers for exploring Mars and field robots in agriculture are only few examples.
Relevant works in order to obtain an introduction to the SLAM problem, with a good description of its main components and maps representations used in the field, are those produced by Durrant-White and Bailey [14, 1]. To obtain, instead, a good overview on the open problems of SLAM, togheter with the direction the research has taken, the work of Cadena et al. [4] is a valid proposal. Instead, a recent survey that approaches in particular the visual-SLAM, analyzing a big number of techniques and with a huge number of valid references is the work produced by A. M. Barros et al. [35].

A first classification of SLAM approaches is based on the nature of map produced, that strongly depends from the kind of sensors used. In particular landmark maps [37, 47] are better suited for maps produced used sensors that detects locally distinguishable features, such as cameras, whereas dense representations [22, 54, 19] are used in conjunction with range sensors.
Talking about the solutions adopted in literature, we can mainly classify them into filtering and smoothing. The first class of approaches is also known as on-line SLAM and exploit classic techniques of probabilistic robotics to find a solution: Kalman and information filter [47, 8], particle filters [37, 22, 19] and information filters [15]. Smoothing approaches, instead, try to solve the SLAM problem estimating the full trajectory of the robot from the entire history of robot measurements [34, 42, 11], for this reason it also known as off-line SLAM.
A popular component of this last class of approaches is the one based on the graph-based formulation of SLAM. It was first proposed by Lu and Milios in 1997 [34] but it took several years before to make it popular, because of the huge complexity of solving the error minimization problem using standard techniques. Only recently, with the advancements in the field of sparse linear algebra and with the acquisition of insights into the structure of the SLAM problem, the error minimization problem has become feasible. These recent approaches brought the graph-based SLAM in the Olympus of SLAM research, becoming in fact the state-of-the-art technique in terms of speed and accuracy. Many popular SLAM system uses this approach [39, 40, 6, 49, 28].

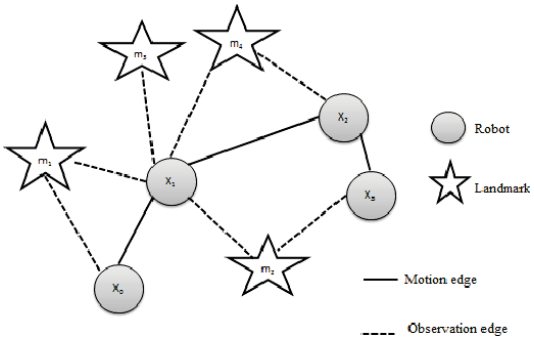


Figure 2.1. Graph-based SLAM illustration [13].

Solving a graph-based SLAM results in building a graph (see Figure 2.1) in which each node is a pose or a landmark position and each edge is an observation of a landmark from a pose (measurement arcs) or of another pose (motion arcs). Indeed, an edge is also a constraint for the graph formulation that allows to find a configuration in which the position of the nodes is maximally consistent with the constraints. This can be done with a large error minimization problem, because the linearization of the set of constraints results in an information matrix and an information matrix and the information matrix inherits the sparseness of the graph [52]. For these reason sparse linear algebra approaches allows to solve this problem in a fast way [27].
Visual-based SLAM algorithms play a significant role in SLAM field because they guarantees a lot of advantages over other sensor-based SLAM techniques as they

are based on low-cost and small cameras. In literature exist different approaches and method to approach visual SLAM, in particular Barros et al. [35], recently, propose three main categories in which to classify them: visual-only, visual-inertial and RGB-D SLAM. In figure 2.2 these three approaches are graphically represented together with the different types of maps that they can produce.

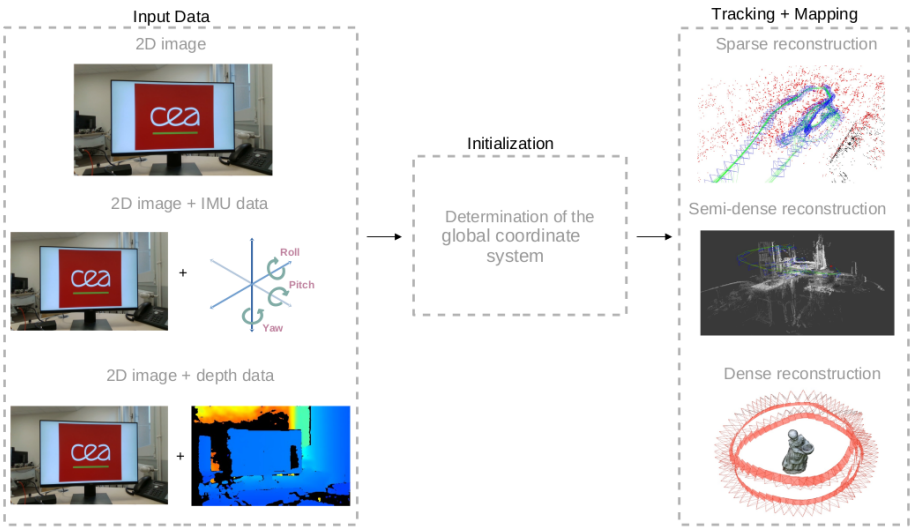


Figure 2.2. General components of a visual-based SLAM. The depth and inertial data may be added to the 2D visual input to generate a sparse map, semi-dense map and a dense reconstruction [35]

The visual-only approach [55, 18] is based on 2D image processing only and the system uses the images acquisition to extract points of interest that will be used to predict the poses of the robot (using multi-view geometry [23]) and of such points that are considered relevant (landmarks). The visual-only SLAM can use monocular or stereo camera. The first one has the advantage of allowing to use cheaper sensors that are also easy to calibrate, but introduces more complexity in all the stages of the process, because drift and scale estimations become more difficult to estimate. For this reasons stereo cameras are used in some cases, but also there we can have problems mainly due to the need for an image rectification process in the stereo matching stage.
The visual-inertial SLAM [21, 9, 25] adds to camera sensors information from an inertial measurement unit (IMU), which consists of a combination of gyroscope, accelerometer and magnetometer devices. This enrich the information from the environment making the process more accurate but, on the other hand, makes the algorithm much more complex.
Lastly, we have SLAM systems based on RGB-D data [10, 56], which add to monocular RGB camera also a depth sensor allowing to use information also about the depth of points of interest in the images. This makes the algorithm much different from the two previous approaches, decreasing the complexity of the initialization stage [50]. This approach is more suitable for indoor environments, and requires large memory and power consumption [7].

Recently, neural networks promised a lot in every field of artificial intelligence, so their application also for the SLAM problem immediately attracted attention. CNN-SLAM [51] is one of the first algorithms that introduce convolutional neural networks inside a real-time SLAM pipeline. In particular, a CNN is used to understand which is the depth of a given keyframe and it is launched in parallel with the tracking algorithm. For this reason this algorithm has the advantage to not suffer from absolute scale limitation. The only drawback is that in order to be performed real-time this algorithm requires a GPU for the CNN.
Others remarkable algorithms that incorporate neural networks into the SLAM pipeline are those proposed by Wang et al.: UnDeepVO [31] and DeepSLAM [30]. The first one is a monocular visual-odometry system that can perform pose and depth estimation using an unsupervised deep neural network, reaching results that are more accurate and robust than other monocular methods without loop closure. The second one, instead, uses three networks: one for tracking, one for mapping and one for loop detection. This approach demonstrates better than other classical approaches.
Another way to adapt neural networks inside the visual-only SLAM is to use them to extract relevant points features from the input images, as an alternative of classical methods such as ORB [45], FAST [44], SIFT [33] and SURF [2]. One system that uses this approach is DF-SLAM [26], which develops a pipeline very similar to ORB-SLAM, but using deep local features by the TFeat network, instead of classical hand-made features. Another neural approach that can be used for points features extraction for visual SLAM is Superpoint [12].

Chapter 3

Basics

3.1 Monocular Visual SLAM

Let us consider a sequence of frames from a video as input for a monocular visual SLAM system. Because our focus is on the off-line implementation of the algorithm, the system will receive the entire set of images at the beginning.
The first step consists into an elaboration of these images in order to extract a set of relevant features from each of them. Independently from the representation chosen, these features stand for the measurements of our system, that we can generalize with this notation $z_{1:t}$ in order to represent the observations done from the frame 1 to the frame t (the time window of our video sequence).
Now, if togheter with the images sequence we consider also the controls of our robotic system we can formalize the SLAM problem from a probabilistic perspective as the posterior [52]:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{3.1}$$

where $x_{1:t}$ is the history of robot's poses, m is the map and $u_{1:t}$ is the control input history. Solving this posterior, knowing that the dimensions of the map space can be infinite, and knowing also the high dimensionality of the other variables involved, is impossible without exploiting an assumption on the SLAM problem: the Markov assumption. Relying on this, we can describe the probabilistic formulation of the SLAM problem as a dynamic Bayesian network (DBN), depicted in figure 3.1.

A Bayesian network is a graphical model that describes a stochastic process as a directed graph. Here we have a node for each random variable in the process and an edge for each dependency. In particular we can distinguish between observed variables (the measurements $z_{1:t}$ and the inputs $u_{1:t}$) and the hidden variables (the state $x_{1:t}$ and the map m) that are the variables to estimate. The edge connections follow the idea of a transition model in which we have a state transition model and an observation model. The state transition model represents the probability that the robot is in pose x_t given that in time $t-1$ it was in pose x_{t-1} and it acquired the inputs u_t and it is formalized as: $p(x_t|xt-1, u_t)$. The observation model, instead,

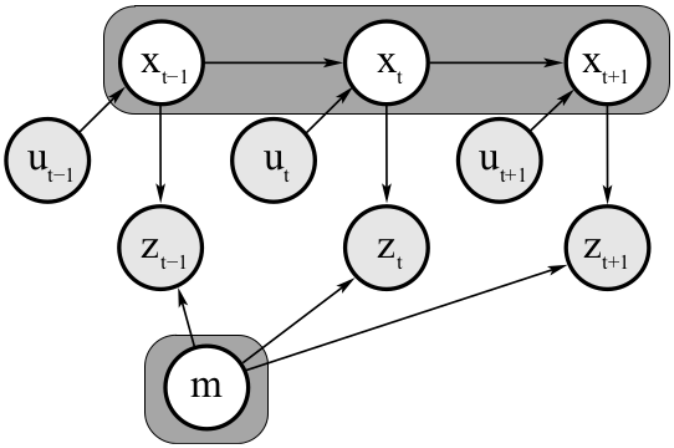


Figure 3.1. DBN model of the full-SLAM problem [52]

represents the probability of performing the observation z_t, given that the robot is in pose x_t in the map m at time t. It is formalized as $p(z_t|x_t, m_t)$.
The DBN formulation of SLAM highlights the temporal structure of the problem and it is well suited for filtering approaches to solve SLAM. A better alternative representation, that can be used with more recent methods, is the graph-based one. In this case the spatial constraints are exploited to build a representation in which each node is either a pose of the robot in a given position in the map or a set of measurements performed from such a pose, as depicted in figure 2.1. Spatial constraints are then encoded into the edges between nodes. Once the graph is constructed one can formulate an optimization problem in which we try to find robot poses that better satisfy those constraints. Indeed, also the measurements positions, that represent the poses of landmarks for the map construction, are optimized togheter with the poses. So the graph-based SLAM approach decouples the problem into two steps: (1) the front-end step (graph construction), that is heavily sensor dependent, in which the graph is being built and (2) the back-end step (graph optimization) in which the best poses' history is found, togheter with the best map. This last step is sensor agnostic and depends on an abstract representation.
The graph-based approach to SLAM problem is the most used in recent literature for monocular visual SLAM [6, 35, 38] and in particular it is integrated in a pipeline that is generally composed by the following steps:

- image features detection: in this step the interest points from an image are detected, togheter with a descriptor that can be viewed as a feature vector that is representative for a given point in the image. These points and descriptors are actually the input data of the SLAM algorithm.

- initialization: then, given the points of interest of at least two images of the input video sequence, an initial map estimation is performed. In particular multi-view geometry algorithms are used to extract information such as the pose of an image with respect to another one. In this way we can obtain two camera poses that are useful to triangulate an initial set of map points. This

is the most delicate part of the algorithm, because both the poses and the map points computed in this step will be used as the baseline for the entire algorithm. For this reason errors in this stage mean errors for the rest of system lifetime, so many techniques must be applied in order to avoid large uncertainty.

- tracking: once we have an initial map and pose estimations we can continue to integrate images in our system in order to compute the evolution of the robot displacement. In this step the new images must be aligned with the previous poses and the map points observed, so a good recognition of points observed previously is important (this is heavily dependent by the image features chosen) in order to predict a pose that better align with the measurements of the current frame. For this stage pose only optimization algorithms can be used, such as Gauss-Newton methods applied to Projective Iterative Closest Point (PICP).
  Because this approach to the tracking will accumulate a certain drift with respect to the real path followed by the robot, some other optimization algorithm must be used in order to exploit the graph-based model and reduce the error. This optimization is called Bundle Adjustment and uses again algorithms such as Gauss-Newton or Levenberg-Marquardt (LM) in order to better suite the graph constraints. Also, in order to maintain fast performance also with huge maps and trajectories, some sort of Local Bundle Adjustment can be performed optimizing only a subset of poses and map points.
  In particular, in order to this to work, a good graph building is necessary during the first stage of tracking. This can be done by inserting in the graph only some frames that are considered relevant to be inserted as nodes in the graph: the keyframes. The decision of which one is a candidate for this treatment is a key point of tracking.

- loop closure: nevertheless bundle adjustment drastically reduces the error accumulated some quantity of drift can still remain in the predictions, this will cause also errors in the scale. For this reasons loops in the trajectories can be detected in order to understand, when an already visited map area is reached, to how much this drift amount. In this way the loop can be closed by connecting detected nodes and performing a global bundle adjustment to zeroing the accumulated error and adjusting the scale.

In the rest of the chapter all these steps are analyzed, finally the graph-based SLAM approach is formalized and the way in which this representation can help the SLAM pipeline is illustrated by analyzing the optimization algorithms on which it relies on.

3.2 Features Detection

Given an image our purpose is to detect a set of points in it that are representative of the environment that the image represents: the keypoints. Those keypoints should be stable and repeatable from different lighting conditions and viewpoints. In particular we want also, for each point, a feature vector that describes uniquely that point, in such a way we can recognize the same point between multiple images

of the same environment just looking at this feature vector: the descriptors of the point. Different algorithms has been developed for this purpose, in the following methods applied in the current thesis are explained in detail.

3.2.1 ORB

ORB feature detection [45] has been developed for feature matching as an alternative to SIFT [33] and SURF [2]. In particular it is based on BRIEF [5] and it is rotation invariant and resistant to noise. It is demonstrated that ORB outperforms in terms of speed SIFT at two orders of magnitude, performing as well as it in many situations. It also performs better than SURF in most cases. An important characteristic of ORB is that it is free to use, whereas SIFT and SURF are patented.
ORB algorithms starts from the idea of FAST to select a pixel as keypoint according to the following method: taking a pixel p in the image, drawing an imaginary circle around it and selecting the surrounding 16 pixels that fall into the circumference of the circle (see Figure 3.2). Then FAST take these pixels and subdivide them into three classes: lighter, darker or similar to p. If more than 8 pixels are in the class of the darkest, then p is considered a keypoint. This approach is well suited to detect images' edges.

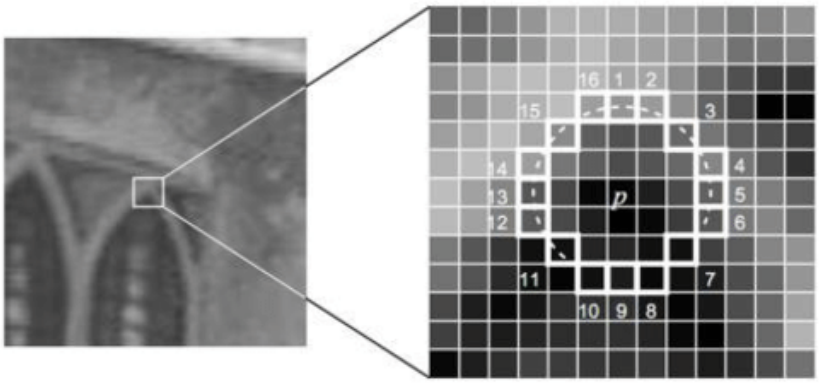


Figure 3.2. The way in which FAST algorithms decide a keypoint basing on the surrounding pixels of a given pixel p [29]

But ORB does not stop here, it wants to be scale invariant, for this reason it introduces the image pyramid. It is basically a multiscale representation of the input image at different resolutions. Now, by detecting keypoints at different resolutions of the same image we can detect them at different scale, obtaining a scale invariant version of FAST.
Rotation invariance is instead obtained by assigning a direction at each keypoint by computing how the intensity change around it using intensity centroid. Knowing the direction of a keypoint allows to rotate it to a canonical rotation and compute the descriptor on it, in order to avoid rotation dependent information.
To do this ORB exploit the binary feature vector used by BRIEF algorithm. In particular BRIEF computes a vector of 0s and 1s called binary feature descriptor from all the keypoints computed with the FAST method. In this way we have a

descriptor associated to each keypoint. The descriptor is computed by taking a random pair of pixels around the keypoint and assigning 1 is the first of these points is brighter than the second, 0 otherwise. In particular, to avoid descriptor from being sensitive to high-frequency noise, the two points are determined according to a Gaussian distribution: this method is known as smoothing of the image. The process is repeated n times, where n is the length of the descriptor vector. To formalize it let's define the binary test as:

$$\tau(p_1, p_2) = \begin{cases} 1 : I(p_1) < I(p_2) \\ 0 : I(p_1) >= I(p_2) \end{cases} \tag{3.2}$$

where $I(.)$ is the intensity value of a given pixel and p_1 and p_2 are the two pixels of the test. Basing on this function we can define the descriptor vector of length n as:

$$f_n(p) = \sum_{1<i<n} 2^{i-1} \tau(p_{1,i}, p_{2,i}) \tag{3.3}$$

where p is the pixel for which we want to compute the descriptor and for which we have pre-computed a set of n test pixels around it, defined as:

$$S = \begin{pmatrix} p_{1,1} & \cdots & p_{1,n} \\ p_{2,1} & \cdots & p_{2,n} \end{pmatrix} \tag{3.4}$$

Again, this algorithm is not rotation invariant, so ORB modify it a bit by steering the process according to the orientation of the keypoints: this is called rBRIEF (Rotation-aware BRIEF). In particular it rotates with discretized angles the points for the BRIEF test, according to the previously computed rotation of the keypoint. Formally, given the keypoint angle θ and the corresponding rotation matrix R_θ it computes a steered version of the test points vector:

$$S_\theta = R_\theta S \tag{3.5}$$

discretizing the angle to increments of $2\pi/30$ that corresponds to 12 degrees. Thee steered version of the S vector is used inside the test formula to compute the descriptors. This approach shows significant improvement with respect to the original method.

3.2.2 Superpoint

As we said in the introduction of this section, keypoints must respect the following conditions: stability and repeatability, in particular way under different lighting conditions and viewpoints. The problem with this definition is that it is ambiguous in terms of semantic of what the keypoints should represent. Keypoints can be corners, ending points of a segment and so on; actually they are defined at very low level. For this reason it is impossible to have a human annotator that can be repeatedly identify the same set of keypoints from an image. This means that it is very difficult to develop a supervised machine learning algorithm that is able to do what algorithms like ORB and SURF do. DeTone et al. [12] have solved this problem by proposing a self-supervised learning algorithm: Superpoint. It actually is an unsupervised machine learning approach that uses a convolutional neural network and a pseudo-ground truth dataset generated using a base detector called Magic Point.
Superpoint architecture, depicted in Figure 3.3, is made-up of three components:

- the Encoder: it is a VGG-style encoder which job is to reduce the dimensionality of the input image. In particular it consists of eight 3×3 convolutional layers of size $64, 64, 64, 64, 128, 128, 128, 128$ with a ReLU non-linear activation function and Batch Normalization and with a 2×2 max pool layer every two of them.
- the Interest Point Decoder: it is a decoder with no parameters (also known as "sub-pixel convolution" or "depth to space") which goal is to determine a probability of being a keypoint for each pixel in the image.
- the Descriptor Decoder: this components has the job to compute the descriptor of each keypoint and actually it computes a D-dimensional vector for each pixel in the image. In particular it apply a bi-cubic interpolation and a L2-normalization to have unit length activations.

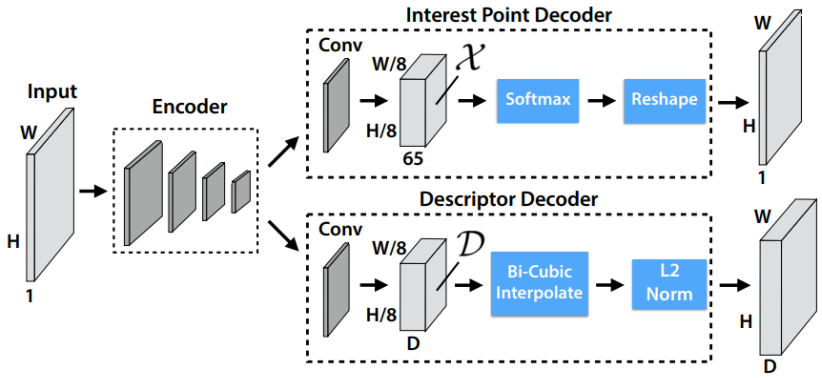


Figure 3.3. The architecture of Superpoint neural network [12]

Magic Point uses the same neural network scheme (ignoring the descriptor head) and it is pre-trained on a synthetic dataset composed by simple geometrical shapes

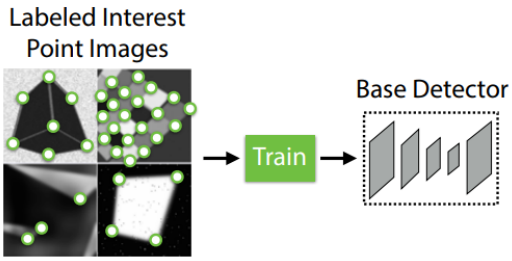


Figure 3.4. The Magic Point Base Detector trained on a synthetic dataset of simple shapes [12]

(cubes, check boards, stars) produced with a simple python program and called "Synthetic Shapes" (see Figure 3.4).

Once the base detector is trained on millions of examples from Synthetic Shapes we obtain a network that is very good when tested on the same kind of data (actually it performs better than FAST and Harris corners detection) but performs very bad on real images of textures and patterns, missing a lot of keypoints. To bridge this gap a multi-scale, multi-transform technique called "Homographic Adaptation" was developed (see Figure 3.5). In particular it warps the input image multiple times with random homography. It is used in conjunction with Magic Point to get the interest points on the warped image. Then the detected keypoints are unwarped obtaining the points for the original image computed from different viewpoints and scales.

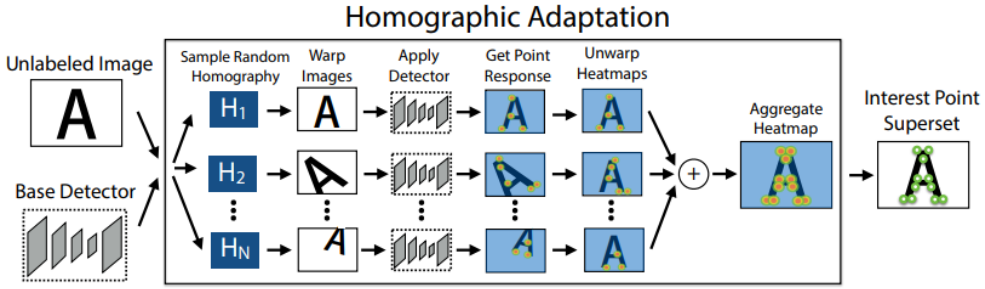


Figure 3.5. A form of self-supervision for boosting the geometric consistency of a keypoint detector trained with a CNN [12]

Once the whole model is trained the first time, the authors of the work use this model to generate the pseudo-ground truth dataset composed by keypoints on the images taken from MSCOCO 2014 [32] (see Figure 3.6). Then they train again the same model that they will call Superpoint multiple times. The Figure 3.7 shows how the multiple training of the model produces a clear improvement of the detection task.

It is important to note that Superpoint is trained jointly with two images, one the randomly warped version of the other (see Figure 3.8) in order to train togheter keypoints detection task and descriptors extraction task. Most of the network's

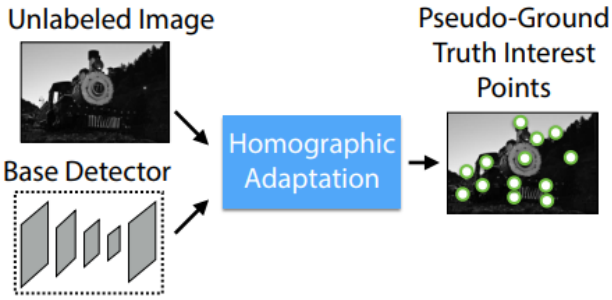


Figure 3.6. Generation of the pseudo-ground truth dataset of keypoints [12]

parameters are shared between the two tasks, differently to what traditional systems do: first computing the keypoints, then the descriptors, lacking the ability to share representations between the two functions.

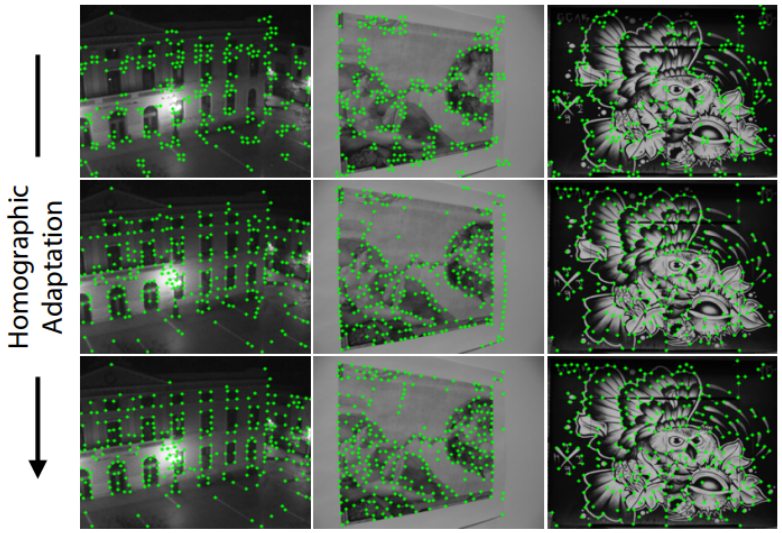


Figure 3.7. In the top row the Magic Point base detector output is showed, unable to find easily repeatable keypoints. In the other rows it is clear how the further training process is able to improve the performance [12]

Figure 3.9 shows how Superpoint tends to produce more dense and correct matches than LIFT, SIFT and ORB. This characteristic is very important for SLAM problem because obtaining a lot of correspondences between images allow to better perform pose optimization task, whereas more distribution of keypoints among all the "surface" of the image allows to obtain a better tracking, this because not only parts of the environment that are dense of objects will be exploited to compute the path evolution but a more significant part of it.

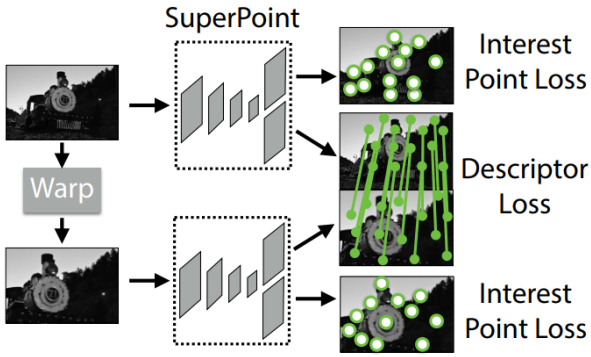


Figure 3.8. Superpoint is trained to jointly extracts interest points and descriptors from an image [12]

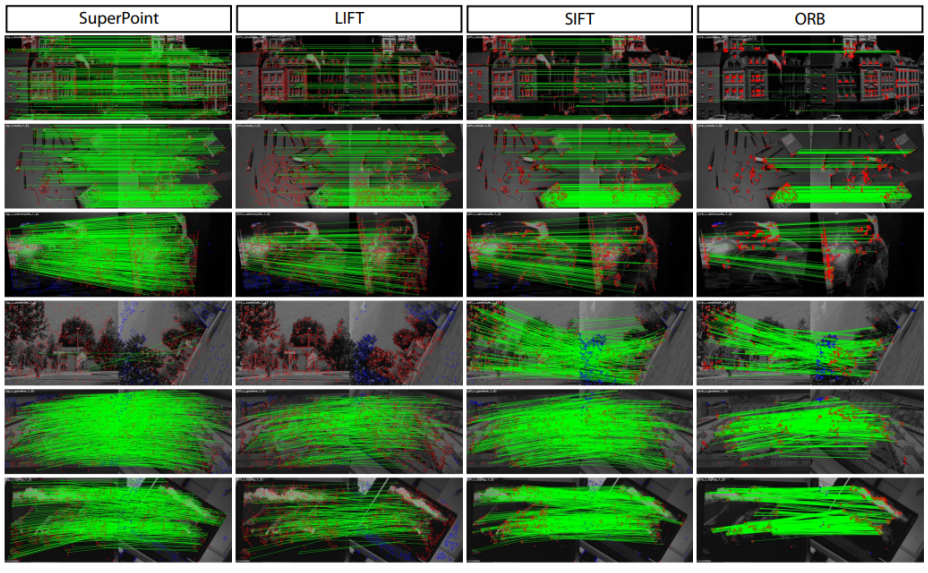


Figure 3.9. Superpoint results compared with LIFT, SIFT and ORB [12]

3.2.3 LF-NET

Another interesting neural approach to extract image keypoints and descriptors is the one proposed by Y. Ono et al. [43]: LF-NET. It is a deep architecture proposed togheter with a training strategy to learn local features from scratch without need of human supervision. Its name comes from Local Feature Network and it is trained without using hand-crafted detector to generate supervised data, but using image pairs for which the relative pose is known. Indeed, the approach used to train the network is the most interesting part of the work.
The network is composed by two main components: a dense, multi-scale, fully convolutional network that returns keypoints, scales and orientations, called detector, and a network that outputs local descriptors given patches cropped around the keypoints produced from the first one, called descriptor. In particular, the first network is designed to achieve fast inference time and to be agnostic to image size.

The structure is depicted in Figure 3.10.

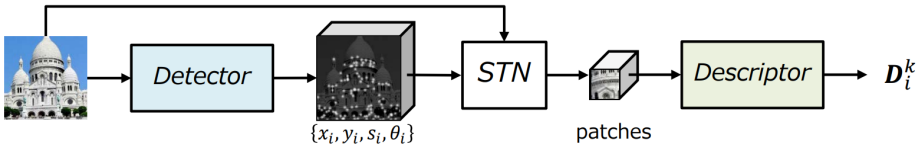


Figure 3.10. The LF-NET architecture [43]

The detector network is basically a simple ResNet [24] with three blocks and it has the aim to produce a rich feature map from an input image, map that will be used to extract keypoints location, scale and orientation. In particular each block of the network contains 55 convolutional filters with batch normalization leaky-ReLU activations followed by other 55 convolutions. Once the map is computed a scale-invariant keypoints detection is proposed: the map is resized N times, at uniform intervals between $1/R$ and R and the resulting versions are convolved with N independent 55 filters. On the resulting N score maps is applied a differentiable form of non-maxima suppression by applying a softmax operator over 1515 windows in a convolutional manner. The N sharper score maps are then resized back to the original image size and merged into a final scale-space score map with a softmax-like operation: in this way scale-invariance is ensured. Then, the K top pixels are chosen as keypoints and a local softargmax is applied for sub-pixel accuracy. Formally, the scale-invariant keypoint detection part of the network is defined as:

$$S = \sum_n \overline{h}^n \cdot softmax_n(\overline{h}^n) \tag{3.6}$$

where $\overline{h}^n$ is the sharper score map resized to the original size.
For descriptor extraction, instead, for each chosen keypoints a image patch around it is selected and passed to 33 convolutional filters with a stride of 2 and 64, 128 and 256 channels respectively, with batch normalization and ReLU activation. The network ends with two fully-connected layer to reduce the dimensionality to 256 and l_2 normalized.
But the most interesting part is how these networks are trained. They propose to use a two-branch architecture with the same network used in both branches whose inputs are two images of the same scene, but different. The input also includes camera intrinsic and extrinsic. The score maps produced for both images can be warped in order to determine the ground truth correspondences between images. It is distinctive the fact that the second branch holds the components which break differentiability, so it cannot be back-propagated. For this reason the parameters learned for the first network are copied in the second one in the next iteration. In Figure 3.11 is schematized the training process.

The training objective is formulated as a combination of two types of loss functions: image-level (that affects keypoint detection) and patch-level (that affects both

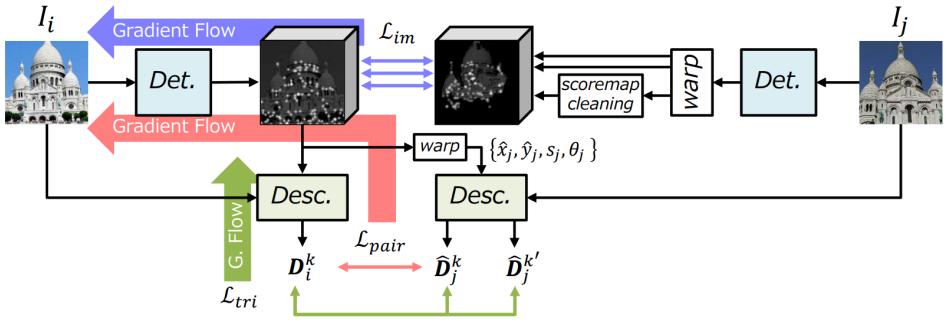


Figure 3.11. The LF-NET training procedure [43]

keypoint detection and descriptor network).
The image-level loss function is, formally:

$$\mathcal{L}_{im} = (S_i, S_j) = |S_i - g(w(S_j))|^2 \tag{3.7}$$

where S_i and S_j are, respectively, the scale-space score map of the first network and of the second network; w is the warping applied to transform the image of the second network in the "frame" of the second one; g is a Gaussian kernel with standard deviation $\sigma = 0.5$, applied to locations of the K top keypoints selected from the image given at the second network. The g function is useful to obtain a clean score map and, because it is not differentiable, it is applied only on the non-optimized network.
To compute the patch-wise loss, instead, once K keypoints are selected from the first network, they are warped in their spatial coordinates in order to match the second image (the same as before but in the opposite direction) and descriptors are extracted at the corresponding regions. If a keypoint falls on occluded regions after warping, they are dropped from the optimization process. The detector network is trained using the descriptors with the following loss function:

$$\mathcal{K}_{pair}(D_i^k, \hat{D}_j^k) = \sum_k |D_i^k - \hat{D}_j^k|^2 \tag{3.8}$$

where D_i and $\hat{D}_j$ are the descriptors and k is the number of "filtered" regions.
In addition to the descriptors, also the geometrical consistency over the orientation of the detected and warped points is enforce, by using the loss:

$$\mathcal{L}_{geom}(s_i^k, \theta_i^k, \hat{s}_i^k, \hat{\theta}_j^k) = \lambda_{ori} \sum_k |\theta_i^k - \hat{\theta}_j^k|^2 + \lambda_{scale} \sum_k |s_i^k - \hat{s}_j^k|^2 \tag{3.9}$$

where $\hat{s}$ and $\hat{\theta}$ are the warped scale and orientation of a keypoints, respectively, whereas λ_{ori} and λ_{scale} are weights to learn.
An important point is that, to learn the descriptor network, also non-corresponding pairs of patches should be considered. For this reason a triplet loss is proposed in order to learn the embedding space for the patches. Whereas for positive pairs they have the ground-truth geometry to find matches, for negative (non-matching) they employ a progressive mining strategy to obtain the most informative patches possible. They sort the negative for each sample by loss in decreasing order and sample randomly over the top M (with $M = mas(5, 64e^{0.6k/1000})$ where k is the current iteration). Actually, they propose to start with a pool of the 64 hardest samples and reduce it as the networks converge, up to a minimum of 5. Finally, with the matching and non-matching pairs, the triplet loss is proposed as:

$$\mathcal{L}_{tri}(D_i^k, \hat{D}_j^k, \hat{D}_j^{k'}) = \sum_k max(0, |D_i^k - \hat{D}_j^k|^2 - |D_i^k - \hat{D}_j^{k'}|^2 + C) \tag{3.10}$$

with $k' \neq k$ (it can be any non-corresponding sample) and $C = 1$ is the margin. To conclude, the loss function used is, for the detector:

$$\mathcal{L}_{det} = \mathcal{L}_{im} + \lambda_{pair}\mathcal{L}_{pair} + \mathcal{L}_{geom} \tag{3.11}$$

whereas, for the descriptor:

$$\mathcal{L}_{desc} = \mathcal{L}_{tri} \tag{3.12}$$

In the work [43] it is shown how the LF-NET outperforms other methods (especially the classical ones) both on outdoor and indoor datasets, with large margin. The best result is given by the fact that the entire feature extraction pipeline is embedded inside a single neural architecture that can be trained end-to-end with just a collection of images. In Figure 3.12 a qualitative result is shown, in particular it is easy to see how the density of "good" correspondences of the LF-NET with respect to other methods is higher. This can be very useful in SLAM systems, where it is better to have an higher number of regions of the environments recognized between different images.

3.3 Initial Pose Estimation

The initialization step in a SLAM system is the most delicate of the entire pipeline. In particular in monocular visual SLAM we cannot get an initial set of 3D points

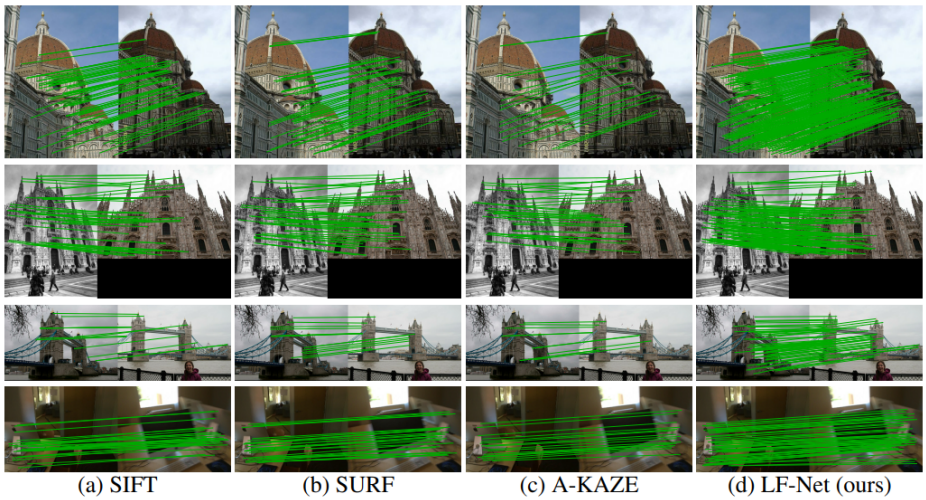


Figure 3.12. Qualitative matching results, with correct matches drawn in green [43]

(the map) from the first measurement only, because it will be represented by a single image from which we cannot recover the depth of the observed environment. A solution for this problem is to consider more than one measurement exploiting methods from multiple view geometry [23], actually two images can be enough because we can compute the displacement between them and triangulate a set of commonly seen keypoints. Two methods can be used to determine the pose of one camera with respect to the other: the eight-point algorithm [23] (or better the five-point algorithm of Nistér [41]) to obtain an essential matrix able to model general scenes or the DLT-algorithm [23] from which obtain an homography that model better planar dominant scenes. From the fundamental matrix multiple solutions can be extracted in terms of poses of one image with respect the other, whereas methods like the one proposed by Faugeras et al. [16] can be used to extract the relative pose from the homography. Problem arises when we have low parallax between the two cameras, in particular to recover the pose without ambiguity from the fundamental matrix we should avoid "rotation only" situations and planar dominant scenes. Indeed, also triangulation is ill-defined under low parallax and "rotation only" displacements. For this reason methods to prevent these situations must be used [53, 40].
Another issue is related to the fact that all the methods proposed to recover an initial motion and so an initial map from two images relies on keypoints matching. Bad matching (outliers) in initialization will produce a bad map that will accompany the system in all its lifetime. This will produce problems also in tracking and graph optimization, so it is important to obtain a big enough set of good correspondences, without requiring human intervention. Iterative methods like RANSAC [17] can be used to obtain the best possible (or almost the best) set of inliers.

3.3.1 Fundamental Matrix

Given a set of points **x** in one image and a corresponding set of points **x'** in another image, we can compute a 3×3 matrix F of rank 2 called Fundamental Matrix that

satisfy the relation $\mathbf{x'}^T F\mathbf{x} = 0$ [23]. This matrix encapsulate the intrinsic geometry between the two views. This geometry is independent of the scene structure and only depends on the cameras' internal parameters and relative pose.
In particular, the fundamental matrix is the algebraic representation of the epipolar geometry, that is defined between two views and represent the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis. In epipolar geometry we define the following entities:

- the epipole: the point of intersection of the line joining the camera centers (actually the baseline) with the image plane (e and e' in Figure 3.13;
- the epipolar plane: the plane that contains the baseline (see Figure 3.14;
- the epipolar line: the intersection of an epipolar plane with the image plane (l and l' in Figure 3.15).

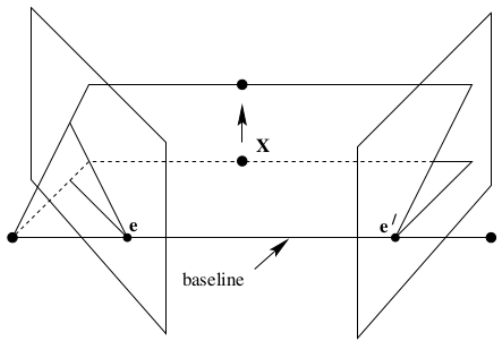


Figure 3.13. The baseline and the epipole in the epipolar geometry [23]

Now, it is easy to see that, for each point $\mathbf{x}$ in one image, there exists a corresponding epipolar line l' in the other image. Actually, the epipolar line is the projection in the second image of the ray from the point $\mathbf{x}$ through the camera center of the first camera. The mapping from a point in one image to the epipolar line in the other image is given by the fundamental matrix: $l' = F\mathbf{x}$. From this relation and from the fact that $\mathbf{x'}^T F\mathbf{x} = 0$ we can say that the rays defined by the points $\mathbf{x}$ and $\mathbf{x'}$ are coplanar: a necessary condition for two points to correspond.

The concepts illustrated so far are necessary to understand that the matching between two image points (the fact that they both refers to the same 3D point in the space) are important in order to have the equation $\mathbf{x'}^T F\mathbf{x} = 0$ to be valid. It is important because can be exploited to compute, given only a set of matched point (actually at least eight), the fundamental matrix that describes the epipolar geometry of two cameras. The numerical method that allows to obtain the F matrix is called eight-point algorithm.
Now, given two points $\mathbf{x} = (x, y, 1)^T$ and $\mathbf{x'} = (x', y', 1)^T$ the equation of the Fundamental Matrix can be unrolled obtaining:

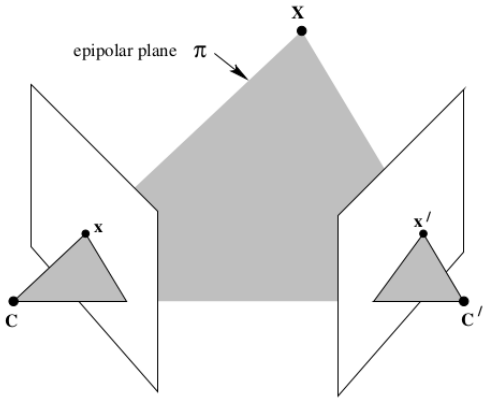


Figure 3.14. The epipolar plane [23]

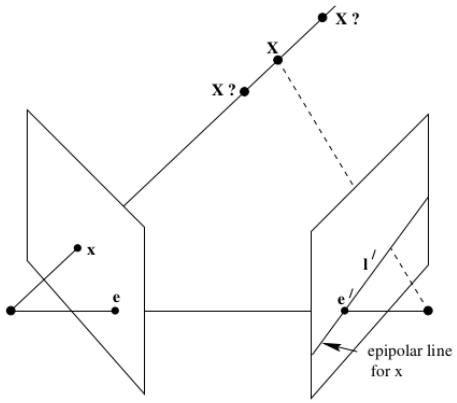


Figure 3.15. The epipolar lines for a point X [23]

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23}xf_{31} + yf_{32} + f_{33} = 0 \qquad (3.13)$$

that can also be expressed as a vector inner product:

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\mathbf{f} = 0 \qquad (3.14)$$

where $\mathbf{f}$ is the vector of 9 elements made up of the entries of F in row-major order.
Given a set of n point matches, we can write down the set of linear equation:

$$\mathbf{Af} = \begin{pmatrix} x'_1x1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nxn & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{pmatrix} \mathbf{f} = 0 \tag{3.15}$$

This homogeneous set of equations produces a solution only if the matrix $\mathbf{A}$ have rank at most 8, if it is exactly 8 then the solution is unique and can be found by linear methods. The solution, that is the vector $\mathbf{f}$ is obviously defined up to a scale, and can be used to reconstruct the Fundamental Matrix $\mathbf{F}$.
If we have not exact points coordinates (because of the noise in the measurements), then the rank of $\mathbf{A}$ may be greater than 8 (actually 9, because we have 9 columns). In such a case a Least-Squares solution can be computed by computing the vector $\mathbf{f}$ that minimize the norm:

$$\mathbf{f}^* = \underset{\mathbf{f}}{argmin} \sum_i \mathbf{f}^T \mathbf{A}^{[i]T} \mathbf{A}^{[i]} \mathbf{f} \tag{3.16}$$

from which we can isolate an $\mathbf{H}$ matrix defined as follows:

$$\mathbf{H} = \sum_i \mathbf{A}^{[i]T} \mathbf{A}^{[i]} \tag{3.17}$$

and rewrite:

$$\mathbf{f}^* = \underset{\mathbf{f}}{argmin}(\mathbf{f}^T \mathbf{H} \mathbf{f}) \tag{3.18}$$

Now, we know that the Fundamental matrix is singular (because it has rank 2) but solving the set of linear equations defined before we will obtain a solution that will not have rank 2 in general. So we need to impose this constraint by searching for a solution of norm 1 ($\mathbf{f}^T\mathbf{f} = 1$) [23]. We can solve it by defining the Lagrange cost:

$$\mathcal{L}(\mathbf{f}, \lambda) = \mathbf{f}^T \mathbf{H} \mathbf{f} - \lambda(\mathbf{f}^T \mathbf{f} - 1) \tag{3.19}$$

that incorporates the constraints adding the Lagrangian multiplier λ. We are searching for a minimum solution, so we need to derive the cost function with respect

to **f**:

$$\frac{\partial \mathcal{L}(\mathbf{f}, \lambda)}{\partial \mathbf{f}} = 2\mathbf{H}\mathbf{f} - 2\lambda\mathbf{f} = 0 \tag{3.20}$$

and with respect to λ:

$$\frac{\partial \mathcal{L}(\mathbf{f}, \lambda)}{\partial \mathbf{f}} = \mathbf{f}^T\mathbf{f} - 1 = 0 \tag{3.21}$$

We can notice that the second equation is our original constraint, whereas the first equation is an eigenvector problem and can be rewritten as:

$$\mathbf{H}\mathbf{f} = \lambda\mathbf{f} \tag{3.22}$$

This simply means that the **f** vector is an eigenvector of the **H** matrix with eigenvalue λ:

$$\mathcal{L}(\mathbf{e}_\lambda) = \mathbf{e}_\lambda^T\mathbf{H}\mathbf{e}_j\lambda = \mathbf{e}_\lambda^T\lambda_\lambda\mathbf{e}_\lambda = \lambda \tag{3.23}$$

Actually, the vector **f** that we are searching for is the eigenvector corresponding to the smallest eigenvalue of the matrix **H**, that can be computed with the singular value decomposition. Furthermore, a normalization of the input points is needed in order to this algorithm to work properly. If we do so, also a de-normalization of the resulting Fundamental matrix is needed [23].
As we said at the beginning, the Fundamental matrix contains cameras' internal parameters and relative pose. So we need to isolate the extrinsic parameters from **F**, parameters that contain information about where one of the camera is with respect to the other one. For this purpose, the intrinsic camera parameters contained in the Camera matrix **K** (supposed as already known) can be exploited:

$$\mathbf{E} = \mathbf{K}^T\mathbf{F}\mathbf{K} \tag{3.24}$$

obtaining the Essential matrix. Actually it contains four solutions about pose, enough accurate but up to a scale. To obtain them, we already need SVD decomposition, because we can factorize the Essential matrix:

$$\mathbf{E} = \mathbf{U}\mathbf{W}\mathbf{S}\mathbf{W}^T\mathbf{V}^T \tag{3.25}$$

where $\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ and:

$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.26}$$

In particular we have the rotational part from:

$$\mathbf{R} = \mathbf{U}\mathbf{W}\mathbf{V}^T \tag{3.27}$$

and the translational part in the form of a skew symmetric matrix from:

$$\lfloor \mathbf{t} \rfloor_x = \mathbf{V}\mathbf{S}\mathbf{W}^T\mathbf{V}^T \tag{3.28}$$

that corresponds to two possible translation vectors **t** and **-t**. The same factorization holds if we use $\mathbf{W}^T$, for this reason four solutions are obtained. In particular only one of them is valid. An empirical method used to determine which solution is the correct one is by searching for the solution that allows to obtain the greater number of triangulated points in front of both the cameras.

3.3.2 Homography Matrix

Given a set of points **x** in one image (defined as a projective plane [23]) and the corresponding set of points **x'** in another image (always defined as a projective plane) we can define a projective transformation (or homography) that takes each point in **x** to **x'**. This homography is formally an invertible mapping $h : \mathbb{P}^2 \to \mathbb{P}^2$ such that three points x_1, x_2 and x_3 (in $\mathbb{P}^2$) lie on the same plane if and only if $h(x_1), h(x_2)$ and $h(x_3)$ do.
In particular, the homography mapping is defined as such if and only if there exists a non-singular 3×3 matrix H such that for any point in $\mathbb{P}^2$ represented by a vector **x** it is true that:

$$h(\mathbf{x}) = H\mathbf{x} \tag{3.29}$$

To conclude, an homography matrix is a matrix that represents a linear transformation on homogeneous 3-vectors:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{3.30}$$

or more briefly, $x' = Hx$. Examples of projective transformations are depicted in Figure 3.16.
Basing on its definition an homography matrix can be very useful to determine which is the transformation between two matched image points. An algorithm that, given at least four point correspondences between two images, allows to determine the H matrix is called DLT-algorithm.
The Direct Linear Transform (DLT) algorithm [23] starts from analyzing the homography equation as a vector cross product:

$$\mathbf{x'} \times H\mathbf{x} = 0 \tag{3.31}$$

This form is useful because allows to derive a simple linear solution for H. In particular, such equation can be rewritten as:

$$\mathbf{x'} \times H\mathbf{x} = \begin{pmatrix} y'_i \mathbf{h}^{3T}\mathbf{x}_i - w'_i \mathbf{h}^{2T}\mathbf{x}_i \\ w'_i \mathbf{h}^{1T}\mathbf{x}_i - x'_i \mathbf{h}^{3T}\mathbf{x}_i \\ x'_i \mathbf{h}^{2T}\mathbf{x}_i - y'_i \mathbf{h}^{1T}\mathbf{x}_i \end{pmatrix} \tag{3.32}$$

where $\mathbf{x}'_i$ and $\mathbf{x}_i$ are the i-th points correspondences and $\mathbf{h}^{jT}$ is the j-th row of the H matrix. Now, since $\mathbf{h}^{jT}\mathbf{x}_i = \mathbf{x}_i\mathbf{h}^j$ for $J = 1, ..., 3$, this gives a set of three equations in the entries of H, which may be written in the form:

$$\begin{pmatrix} \mathbf{0}^T & -w'_i\mathbf{x}_i^T & y'_i\mathbf{x}_i^T \\ w'_i\mathbf{x}_i^T & \mathbf{0}^T & x'_i\mathbf{x}_i^T \\ -y'_i\mathbf{x}_i^T & x'_i\mathbf{x}_i^T & \mathbf{0}^T \end{pmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = A_i\mathbf{h} = 0 \tag{3.33}$$

Actually, because only two of the three equations are linearly independent, we omit the third row of the A_i matrix (indeed it is obtained, up to scale, from the sum of x'_i times the first row and y'_i times the second). For this reason, we have that A_i is a 2×9 matrix.
Now, we are interested in solving such linear equation and we are searching for a non-zero solution of $\mathbf{h}$. The matrix A has rank 8, and thus it has 1-dimensional

null space, which provides an exact, up to scale, solution for **h**. The problem arise when we deal with real data. In such case we have noisy point, so we cannot look for an exact solution, and we need to determine an approximate solution. So we set a condition on the norm $||\mathbf{h}|| = 1$ to avoid zero solutions and try to minimize the norm $||A\mathbf{h}||$. The solution is given by the (unit) eigenvector of the matrix $A^T A$ with least eigenvalues [23]. Again, the solution can be found using SVD decomposition. Also in this case it is a good practice to normalize the points before to compute the H matrix. Then a de-normalization of the result is needed.
To conclude we can notice that projective transformations defined by an homography matrix better explain rotation-only displacements, whereas the Fundamental Matrix fails in such a task. This fact can be exploited in SLAM algorithms because we can understand if we have a rotation-only camera movement by just looking at "how well" a homography matrix explain such displacement with respect to the Fundamental Matrix [40]. Furthermore, methods to extract directly from the H matrix the relative translation and rotation exist in literature [16].

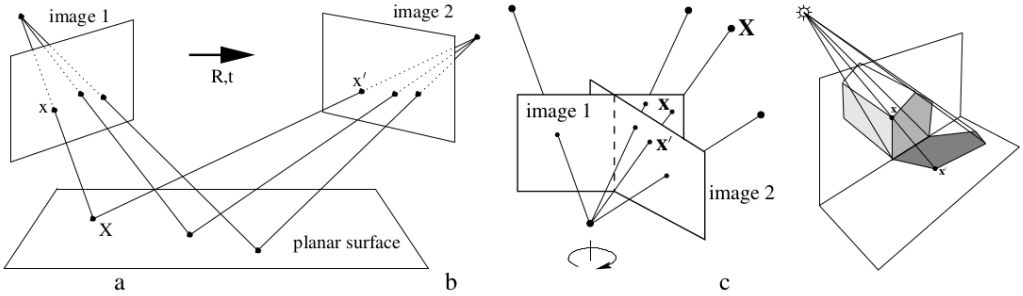


Figure 3.16. Examples of projective transformations: (a) The projective transformation between two images induced by a world plane; (b) The projective transformation between two images with the same camera centre (e.g. a "rotation-only" displacement); (c) The projective transformation between the image of a plane (the end of the building) and the image of its shadow onto another plane (the ground plane) [23]

3.3.3 RANSAC

The RANdom SAmple Consensus (RANSAC) schema proposed by Fischler and Booles [17] is a general method designed to cope with large proportion of outliers in a general set of input data. Its particularity is the fact that it has been proposed within the computer vision community, so it is well suited for all those problems that requires to deal with outliers in image data.
RANSAC is basically an iterative robust estimator that uses resampling of data in order to find a mathematical model from a dataset that contains outliers, by using the minimum number observations required to estimate it. Basically, it proposed to use random data (a minimal set of it, that is actually the minimum number of elements required for the problem at hand, e.g. eight in the eight-point algorithm) to estimate models that will be then evaluated computing the set of inliers that such a model produces (the consensus) and the error. This procedure is repeated a certain number of times, and each time the best model is kept. At the end, the set of inliers found with the best model (actually, the maximum number of inliers) is

exploited to compute a new model, only on them.
As you can see it is only a general schema, because an actual implementation will require both a method to compute the consensus and the error, an algorithm to obtain a model and a way to compute a random minimal set at each iteration. The complete schema is illustrated in Figure 3.17, where also a threshold T is used to stop the cycle when a certain number of inliers if found (after N iteration the method will stop anyway).

Objective

Robust fit of a model to a data set S which contains outliers.

Algorithm

(i) Randomly select a sample of s data points from S and instantiate the model from this subset.
(ii) Determine the set of data points S_i which are within a distance threshold t of the model. The set S_i is the consensus set of the sample and defines the inliers of S.
(iii) If the size of S_i (the number of inliers) is greater than some threshold T, re-estimate the model using all the points in S_i and terminate.
(iv) If the size of S_i is less than T, select a new subset and repeat the above.
(v) After N trials the largest consensus set S_i is selected, and the model is re-estimated using all the points in the subset S_i.

Figure 3.17. The RANSAC schema [23]

The common question that arises by itself is: how we can determine the optimal number of iterations? It must be chosen big enough to ensure a certain probability p (usually 0.99) that at least one of the sets of random samples does not include an outlier. In particular defining the value w as the "inlier ratio", that is the number of good matches divided the number of data in the model and the value n as the minimum number of data required by the model to be estimated, we can compute the number of iterations required as:

$$N = \frac{log(1-p}{log(1-w^n)} \tag{3.34}$$

that is obtained by looking at the probability of not fetching n inliers $(1-w^n)$ from which results that the probability of not having n inliers in N rounds is $(1-w^n)^k$. It actually corresponds to the probability of failure $1-p$. A simple inversion results in the equation illustrated before.
RANSAC is, for the reasons illustrated so far, a good method to use in a SLAM pipeline in order to find the maximum possible number of inliers matches between points. In particular, using appearance of keypoints features (the descriptors) for matching, usually leads to few good correspondences and becomes natural to use it. Finding the maximum number of inliers is then useful, for example, to have a robust base on which estimate Fundamental Matrix or Homography.

3.4 Graph-based SLAM Tracking

The graph-based approach to SLAM proposes to implement a spatial structure of the system evolution, as explained in the introduction of this chapter. For this reason, the tracking mechanism is relevant in order to obtain a good graph on which rely on. In an off-line SLAM problem each measurement, represented by an image and the corresponding set of keypoints and descriptors, is used to compute the pose of the camera from which such image is taken with respect to already known points in the map. This step is known as Protective Iterative Closest Point and it is an image-based version of the registration problem, where a pose is aligned with respect to observations made on points in the space. A good initial guess is required for this approach to work.
Once the graph is built it is useful to exploit its representation to execute an optimization algorithm that better align all poses (both of cameras and map points) in order to match the constraints (represented by edges in the graph). This step is called Global Bundle Adjustment and approaches like the Gauss-Newton method or Levenberg-Marquardt algorithm are used.
Actually, deal with real data means also to have a lot of poses (a video can contains around 25 frames per second) and huge quantity of map points. For this reason a smart approach in order to choose what are the relevant poses in the trajectory is required: the keyframe selection. In any case, even with a reduction in terms of frames considered, a Global Bundle Adjustment can still be an heavy computation to perform. Local maps can be computed around each keyframe in order to still be able to apply optimization but focusing only on a subset of the graph, reducing drastically the execution time. Actually, this approach, known as Local Bundle Adjustment, has been shown to be enough to obtain a result near to the one obtained using Global Bundle Adjustment during tracking [38, 39, 40, 6].

3.4.1 Projective ICP (Iterative Closest Point)

Given a pose of the camera as initial guess and a set of correspondences between points on the image and 3D points in the space, projective ICP allows to compute a new camera pose that better explain such matches. In particular it iteratively move slightly the initial guess towards a pose that minimize the re-projection error of the points in the space, such that their will be aligned with the correspondents image points. For this reason it can be accomplished as an optimization problem. Actually it must be also able to deal with bad matches (outliers) so a Gauss-Newton algorithm with robust kernel for Least-Squares minimization can be a good choice.
In order to formalize it we first need to determine a good representation for the state of the problem: the pose. A pose is typically represented as a transformation matrix:

$$X \in SE(3) : X = (R|t) \tag{3.35}$$

that is an element of the $SE(3)$ space because we have a non-Euclidean part, that is the rotation matrix $R \in SO(3)$. A rotation matrix has 9 elements to represent 3 angles, so if we use it as parameters for our optimization algorithm we will find non-valid solutions, since the orthogonality constraint is not enforced. So we need to exploit the fact that rotations are manifold and they admit a local parameterization which is homeomorphic to the vector space $\mathbb{R}^n$. This means that given a generic rotation R parameterized with three angles $\alpha_x, \alpha_y, \alpha_z$ we can define a mapping between rotation matrices and Euler angles:

$$\alpha = r2v(R) \tag{3.36}$$

and viceversa:

$$R = v2r(\alpha) \tag{3.37}$$

where $\alpha = (\alpha_x, \alpha_y, \alpha_z)^T$ represents the vector containing the three Euler angles. But, if we are near to singularities in those angles a small orientation difference might lead to an abrupt difference in the local parameters α, with bad results in pose prediction. For this reason we need operators that allows to convert a global difference in the manifold space to a local perturbation, and viceversa. Given an arbitrary rotation R_0, we can compute a local mapping to Euler angles that is away from singularities by setting the reference frame so that $r2v(R) = 0$ is the origin, obtaining the $\boxminus$ operator:

$$\alpha = R \boxminus R_0 = r2v(R_0^{-1}R) \tag{3.38}$$

and the $\boxplus$ operator:

$$R = R_0 \boxplus \alpha = R_0 v2r(\alpha) \tag{3.39}$$

The first equation returns the Euler angles of the rotation that moves R_0 to R in the reference frame of R_0. It computes the coordinates in a local map, centered around R_0. If the two rotations have small differences also α will be small, independently on whether R_0 is close to a singularity or not.
We can now use these operators to be able to apply perturbations to our state vector in order to slightly improve it at each iteration of our optimization algorithm. In particular, we can define a linearization of the transformation matrix:

$$\Delta x \in \mathbb{R}^6 : \Delta x = (x, y, z, \alpha_x, \alpha_y, \alpha_z)^T = (t^T, \alpha)^T \tag{3.40}$$

that can be used as perturbation of our state, applied using the $\boxplus$ operator:

$$X \boxplus \Delta x : v2t(\Delta x)X \tag{3.41}$$

where the operator $v2t()$ is a function that transform a linear vector to a transformation matrix:

$$v2t(\Delta x) = [v2r(\alpha)|t] \tag{3.42}$$

Now, we have a good formal representation of the state and a way to apply perturbations to it nevertheless we are dealing with a manifold space. We also need a representation for the measurements:

$$z^{[m]} \in \mathbb{R}^2 : z^{[m]} = (u^{[m]}, v^{[m]})^T \tag{3.43}$$

that are the points $p^{[m]}$ on the image plane, so they relies on an Euclidean space. In order to compute the error we also need a way to represent the prediction $h^{[n]}$ of each 3D point $p^{[n]}$ in the space, by projecting them on the camera:

$$h^{[n]}(X) = proj(Kh_{icp}^{[n]}(X)) \tag{3.44}$$

where K is the camera matrix, $h_{icp}^{[n]}$ is the classical prediction of ICP problem, that brings a 3D point p (in homogenous coordinates) into the camera frame:

$$\hat{p}^{[n]} = h_{icp}^{[n]}(X) = Xp^{[n]} \tag{3.45}$$

and $proj()$ is the function that, given a 2D point in homogeneous coordinates, return the 2D normalized point:

$$p_{img} = proj(p_{cam}) = (x_{cam}/z_{cam}, y_{cam}/z_{cam})^T \tag{3.46}$$

From these definition it is easy to compute the error that we will use in our Gauss-Newton algorithm as the difference between the points on the image (the measurements) and the corresponding 3D points in the space projected on the camera plane (the predictions):

$$e^{[n,m]}(X) = h^{[n]}(X) - z^{[m]} = proj(Kh^{[n]}_{icp}(X)) - z^{[m]} \tag{3.47}$$

that is also known as the re-projection error (for this to work we need a data association between each 3D point in the space to each measured 2D point [52]). Only two things remain in order to be able to set up a Gauss-Newton algorithm, first the error of the perturbation:

$$e^{[n,m]}(X \boxplus \Delta x) = proj(Kh^{[n]}_{icp}(X \boxplus \Delta x)) - z^{[m]} \tag{3.48}$$

and then its jacobian:

$$J^{[n]} = \frac{\partial e^{[n,m]}(X \boxplus \Delta x}{\partial \Delta x)}|_{\Delta x=0} = \frac{\partial proj(p)}{\partial p}|_{p=\hat{p}^{[n]}_{cam}} K \frac{\partial h^{[n]}_{icp}(X \boxplus \Delta x)}{\partial \Delta x}|_{\Delta x=0} \tag{3.49}$$

$$= J_{proj}(\hat{(p)}^{[n]}_{cam}) K J^{[n]}_{icp} \tag{3.50}$$

where:

$$J^{[n]}_{icp} = \begin{pmatrix} I_{3\times 3} & | & \lfloor -\hat{p}^{[n]} \rfloor_x \end{pmatrix} \tag{3.51}$$

and:

$$J_{proj}(p) = \begin{pmatrix} 1/z & 0 & -x/z^2 \\ 0 & 1/z & -y/z^2 \end{pmatrix} \tag{3.52}$$

With all these definitions we can easily built a Gauss-Newton machinery that will be able to perform Projective ICP. The complete algorithm that iterates over K measurements (point matches) is depicted in Figure 3.18 with a slightly different notation.

Require: $\breve{\mathbf{x}}$: initial guess. $\mathcal{C} = \{\langle \mathbf{z}_k(\cdot), \mathbf{\Omega}_k \rangle\}$: measurements
Ensure: $\mathbf{x}^*$: new solution
//compute the current error
$F_{\text{new}} \leftarrow \breve{F}$
// iterate until no substantial improvements are made
repeat
$\breve{F} \leftarrow F_{\text{new}}$
$\mathbf{b} \leftarrow \mathbf{0} \qquad \mathbf{H} \leftarrow \mathbf{0}$
for all $k = 1 \dots K$ **do**
// Compute the prediction $\hat{\mathbf{z}}_k$ and the error $\tilde{\mathbf{e}}_k$
// Note that the measurements $\mathbf{z}_k$ are overparameterized
$\hat{\mathbf{z}}_k \leftarrow \hat{\mathbf{h}}_k(\mathbf{x})$
$\tilde{\mathbf{e}}_k \leftarrow \hat{\mathbf{z}}_k \boxminus \mathbf{z}_k$
and the jacobians $\tilde{\mathbf{J}}_k$ and $\mathbf{J}_{\mathbf{z}_k}$
// Compute the jacobian of the error function, in the space of the increments
$\mathbf{J}_k \leftarrow \left. \frac{\partial \mathbf{e}_k(\mathbf{h}_k(\breve{\mathbf{x}} \boxplus \mathbf{\Delta x}), \mathbf{z}_k)}{\partial \mathbf{\Delta x}_k} \right|_{\mathbf{\Delta x}_k = \mathbf{0}}$
// Compute the jacobian of the ominus operator in the measurements
$\mathbf{J}_{\mathbf{z}_k} \leftarrow \left. \frac{\partial ((\hat{\mathbf{z}}_k + \mathbf{\Delta z}_k) \boxminus \mathbf{z}_k)}{\partial \mathbf{\Delta z}_k} \right|_{\mathbf{\Delta z}_k = \mathbf{0}}$
// Remap the information matrix
$\tilde{\mathbf{\Omega}}_k \leftarrow \mathbf{J}_{\mathbf{z}_k} \mathbf{\Omega}_k^{-1} \mathbf{J}_{\mathbf{z}_k}^T$
// compute the contribution of this measurement to the linear system
$\mathbf{H}_k \leftarrow \mathbf{J}_k^T \tilde{\mathbf{\Omega}}_k \mathbf{J}_k$
$\mathbf{b}_k \leftarrow \mathbf{J}_k^T \tilde{\mathbf{\Omega}}_k \mathbf{e}_k$
// accumulate the contribution to construct the overall system
$\mathbf{H} \mathrel{+}= \mathbf{H}_k$
$\mathbf{b} \mathrel{+}= \mathbf{b}_k$
end for
// solve the linear system
$\mathbf{\Delta x} \leftarrow \text{solve}(\mathbf{H}\,\mathbf{\Delta x} = -\mathbf{b})$
// update the state
$\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} \boxplus \mathbf{\Delta x}$
// compute the new error
$F_{\text{new}} \leftarrow F(\breve{\mathbf{x}})$
until $\breve{F} - F_{\text{new}} > \epsilon$
return $\breve{\mathbf{x}}$

Figure 3.18. The Gauss-Newton algorithm for Projective ICP

The Gauss-Newton algorithm can then be improved in order to be able to deal to outliers by adding a robust kernel. Actually we can do this by computing a "chi error" as $\chi = e^{[n,m]\prime} e^{[n,m]}$ and damping the vector $e^{[n,m]}$ when the result is over a threshold τ with the following equation:

$$e^{[n,m]} = e^{[n,m]}\sqrt{\tau/\chi} \tag{3.53}$$

It is easy to note that this approach is strongly dependent by the initial guess, so also a good choose of it is relevant in order to obtain a good tracking. We can also use this method to obtain a distinction from inliers and outliers, by just considering as inliers all the matches for which the χ error is under a threshold. In this way we have also a filter for data associations at each pose prediction.

3.4.2 Global Bundle Adjustment

Global Bundle Adjustment is the core of graph-based SLAM system, because it allows to really exploit the graph built during the front-end step. Starting from the formal definitions done in the previous section we actually need to do just some few modification to the method applied for Projective ICP, especially in the state formalization, and then apply basically the same optimization algorithm, or a similar one.
In this step the goal is to optimize the set of all poses and all map points predicted, so we need to model a state that includes all the factor graph. Let's start by modeling the N poses of the trajectory in a vector:

$$X : X = \{X_r^{[1]}, ..., X_r^{[N]}\} \tag{3.54}$$

where each of them is an element of the 3-dimensional special Euclidean group:

$$X_r^{[n]} \in SE(3) : X_r^{[n]} = (R^{[n]}|t^{[n]}) \tag{3.55}$$

On this definition we need to define the increments as a large vector containing the minimal perturbation for each state variable:

$$\Delta x \in \mathbb{R}^{6N} : \Delta x = (\Delta x_r^{[1]T}, ..., \Delta x_r^{[N]T})^T \tag{3.56}$$

where each increment is a linearization of the transformation matrix:

$$\Delta x_r^{[n]T} \in \mathbb{R}^6 : \Delta x_r^{[n]T} = (\Delta x^{[n]}, \Delta y^{[n]}, \Delta z^{[n]}, \Delta\alpha_x^{[n]}, \Delta\alpha_y^{[n]}, \Delta\alpha_z^{[n]})^T \tag{3.57}$$

$$= (\Delta t^{[n]}, \Delta\alpha^{[n]})^T \tag{3.58}$$

The $\boxplus$ operator is now adapted to be applied to the individual perturbation for each variable block:

$$X' = X \boxplus \Delta x \tag{3.59}$$

and so:

$$X - r^{[n]\prime} = X - r^{[n]} \boxplus \Delta x_r^{[n]} = v2t(\Delta x_r^{[n]}) X_r^{[n]} \tag{3.60}$$

For which regards the measurement we have two of them: robot-robot observations and robot-point observation. The first one, if we consider that the pose i observes the pose j, can be modeled as:

$$Z^{[i,j]} \in SE(3) : Z^{[i,j]} = (R^{[i,j]}|t^{[i,j]} \tag{3.61}$$

formulation that leads to an error of the form:

$$e^{[i,j]}(X) = h^{[i,j]}(X) \boxminus Z^{[i,j]} = X_r^{[i]-1} X_r^{[j]} \boxminus Z^{[i,j]} \tag{3.62}$$

where $h^{[i,j]}(X)$ is the prediction. This error then needs the introduction of the function $t2v()$ that is highly non-linear, so the computation of the jacobian becomes too complex. This problem can be solved by introducing the chordal distance. Given two transformation matrix, the chordal distance is the difference between each column in the rotation matrix and the translation vector, that models good what the subtraction between two rotation matrices is. This need the introduction of a 12×1 vector obtained flattening the transformation matrix:

$$flatten(X) = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ t \end{pmatrix} \tag{3.63}$$

where $R = (r_1, r_2, r_3)$. Now we can rewrite the prediction as:

$$h^{[i,j]}(X) = flatten(g^{[i,j]}(X)) = flatten(X_r^{[i,j]-1} X_r^{[j]}) \tag{3.64}$$

So the error function is now modeled through a simple subtraction:

$$e^{[i,j]}(X) = flatten(X_r^{[i]-1} X_r^{[j]}) - flatten(Z^{[i,j]}) \tag{3.65}$$

and the resulting Jacobian is simply a $12 \times 6N$ matrix of the form:

$$\frac{\partial e^{[i,j]}(X \boxplus \Delta x)}{\partial \Delta x} = \begin{pmatrix} 0_{12\times 6} & \dots & \frac{\partial e^{[i,j]}(X \boxplus \Delta x)}{\partial \Delta x_r^i} & \dots & 0_{12\times 6} & \dots & \frac{\partial e^{[i,j]}(X \boxplus \Delta x)}{\partial \Delta x_r^j} & \dots & 0_{12\times 6} \end{pmatrix} \tag{3.66}$$

where:

$$J_j^{[i,j]}(X) = \frac{\partial e^{[i,j]}(X \boxplus \Delta x)}{\partial \Delta x_r^j} = \begin{pmatrix} 0_{9\times 3} & [r'_x | r'_y | r'_z] \\ R_i^T & -R_i^T [t_j]_x \end{pmatrix} \tag{3.67}$$

and:

$$J_i^{[i,j]}(X) = \frac{\partial e^{[i,j]}(X \boxplus \Delta x)}{\partial \Delta x_r^i} = -J_j^{[i,j]}(X) \tag{3.68}$$

At this point we have a vector state for the poses and a way to model the error of observations pose-to-pose, that can be derived to understand "where to go" to better suite the constraints of the graph. We just need a way to model also the part of the state for the map points and the relative error and Jacobian. Actually it is a simple step, we can just expand our state by creating a second vector that contains all the map landmarks:

$$X_l : X_l = \{X_l^{[1]}, ..., X_l^{[M]}\} \tag{3.69}$$

where M is the number of points in the map. Then we can simply model the error and jacobian for them as we have done in projective ICP:

$$J_{proj}(p) = \begin{pmatrix} 1/z & 0 & -x/z^2 \\ 0 & 1/z & -y/z^2 \end{pmatrix} \tag{3.70}$$

but with a difference: now we have also that a measurement influences not only the pose of the robot but also the position of the map point. For this reason we need to model the Jacobian of the robot pose i:

$$J_r^{[i]} = J_{proj} K \begin{pmatrix} -R' & | & R' skew(X_l^{[j]}) \end{pmatrix} \tag{3.71}$$

and the Jacobian of the landmark pose j:

$$J_l^{[j]} = J_{proj} K R' \tag{3.72}$$

At this point, in order to set up the complete Least-Squares problem, we need to generalize a little bit the notation. First, let's re-formulate the state as a generic vector (it will contain both robot poses and landmarks positions, linearized):

$$X^T = (X^{[i]T}, ..., X^{[N+M]T}) \tag{3.73}$$

Now, if we take the generic measurement $z^{[k]}$ (remember that it can be pose-to-pose or pose-to-landmark) we can compute on it the error $e^{[k]}(X^{[k]})$ on a subset of the state variables, that are the only that affect such measurement:

$$X^{[k]T} = (X^{[k1]T}, ..., X^{[kq]T}) \subset (X^{[1]T}, ..., X^{[n]T}) \tag{3.74}$$

On this generalization we can setup a generic optimization algorithm in which we will consider the log likelihood:

$$F(x) =_{k \in C} e^{[k]}(X^k)^T \Omega^{[k]} e^{[k]}(x^{[k]}) \tag{3.75}$$

from which we will search the optimum state $X*$ as $argmin_X F(X)$. You can see that this generic log likelihood models all graph constraints: how the pose of a robot influences another robot pose observed and how a robot pose influences a landmark measured, and viceversa. In order to find an optimal state we can be use approaches like the one presented for the projective ICP (the Gauss-Newton method) where iteratively state increments are computed. To do so we need to build a linear system with the H matrix as:

$$H^{[k]} = \begin{pmatrix} \ddots & \vdots & & \vdots & & \vdots & \\ \dots & J^{[k1]}\Omega J^{[k1]} & \dots & J^{[k1]}\Omega J^{[ki]} & \dots & J^{[k1]}\Omega J^{[kq]} & \dots \\ & \vdots & & \vdots & & \vdots & \\ \dots & J^{[ki]}\Omega J^{[k1]} & \dots & J^{[ki]}\Omega J^{[ki]} & \dots & J^{[ki]}\Omega J^{[kq]} & \dots \\ & \vdots & & \vdots & & \vdots & \\ \dots & J^{[kq]}\Omega J^{[k1]} & \dots & J^{[kq]}\Omega J^{[ki]} & \dots & J^{[kq]}\Omega J^{[kq]} & \dots \\ & \vdots & & \vdots & & \vdots & \ddots \end{pmatrix} \tag{3.76}$$

and the b vector as:

$$b^{[k]T} = \begin{pmatrix} \vdots \\ J^{[k1]T}\Omega e^{[k]} \\ \vdots \\ J^{[ki]T}\Omega e^{[k]} \\ \vdots \\ J^{[kq]T}\Omega e^{[k]} \\ \vdots \end{pmatrix} \tag{3.77}$$

In these structures the jacobians to use are those presented previously, just with a different notation. You can also see that this approach exploit the graph structure of this SLAM formulation, because both nodes and edges are modeled as state variables and measurements, respectively.
We can notice that the H matrix contains blocks and each of its block determine as a measurement influences a part of the state component (the state which comprises both the robot poses and the landmark poses) and same for the b vector, with a big difference: the b vector (assembled from each measurement) is dense, whereas the H matrix (assembled from each measurement) is sparse and full diagonal, because each observation correlates a fixed number of variables and most of them are on the diagonal. This fact can be exploited when the linear system solution must be computed, using methods that allows to deal with sparse matrices (a good example of this theory result applied is represented by the g2o library [27]).
To conclude, let's say that the math computed here can be applied to whatever

optimization algorithm can use such formalization: Gauss-Newton and Levenberg–Marquardt are the most used. The results of each optimization step is a Δx vector that contains both the linearization of robot poses and landmark poses. The way in which these increments are added to the state is the same used in the formulas expressed in this section and in the previous one.

3.4.3 Local Bundle Adjustment

Global Bundle Adjustment has a big issue: it is heavy to compute when the number of nodes in the graph is huge, typical in a real world application. For this reason it must be executed in an alternative thread to maintain a real-time execution [40] and it cannot be used to improve the tracking stage but only as a final refinement or in a loop closure. For this reason different approaches to use the same algorithm in a subset of the map as been explored.
In particular, simple approaches to local map computation are those that, at each frame that we want to optimize, we also insert as optimizable nodes the last $n-1$ frames that precede it and all the map points observed from those n frames. Then, in order to keep the problem constrained with the all map structure, also the $N-n$ frames that comes before the n chosen in the trajectory are inserted in the local map, but "frozen" (the increments are not added to those poses). This exploit the fact the points observed by the n optimizable nodes are constrained (observed) also from some of the previous frames, and such constrained cannot be ignored. Actually, also all the points observed from the $N-n$ set of frames, but not observed from the n optimized, must be added, "frozen", in the state vector. In Figure 3.19 a local map that follows this approach is depicted.

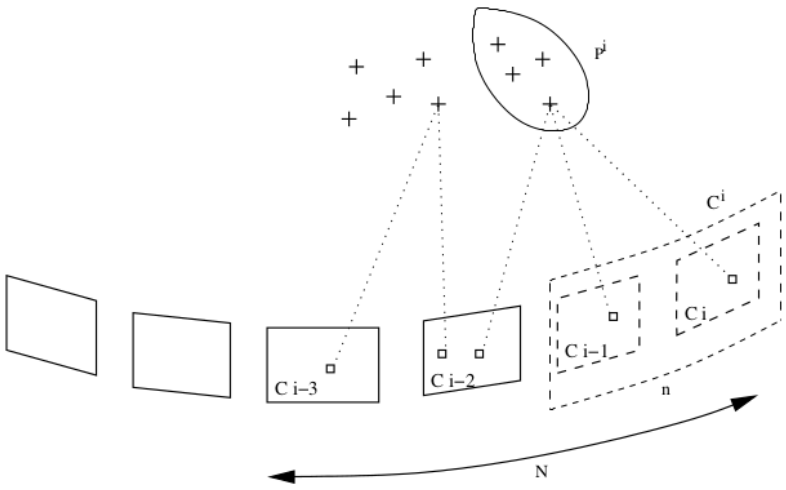


Figure 3.19. Local map obtained when a frame C^i is considered. Only surrounded points and camera poses are optimized. Also reprojections in the N last frames are considered [38]

It has been shown that small values of n (3 or 4 is enough) and N (at least bigger than $n+2$, actually a value comprised between 6 and 11 is enough) lead sufficient

results, comparable to the ones obtained with a global bundle adjustment [38].
Then we can have more complex approaches, like the one proposed by ORB-SLAM [39, 40, 6], where at each keyframe is attached a local map, computed by taking into account all those keyframes that share at least 15 points observation with it. This leads to heavier computations and bigger local maps, but it is also more accurate when same areas of the environment are met again after a while.
Once the local map is obtained, we can launch, at each frame (or keyframe), an optimization algorithm on the subset of nodes at which such frame (or keyframe) belongs. In this way we can exploit the graph structure also during the tracking stage, improving it and also maintaining a good scale of the map and keyframe distances.

3.4.4 Graph Construction

During the exploration, as we have seen in the first part of this section, we need to choose some frame of the input video sequence as relevant, and use it as a node of our graph. The part of the SLAM pipeline that allows to choose if a frame is relevant or not is called "keyframe insertion". Different approaches have been explored, but actually all of them are agree on the fact that, if a consistent number of new points are observed in the new frame from the last keyframe, this should be considered relevant [38, 40]. In this way we can obtain a good tracking, because we have a lot of points that matches with the world and a valid projective ICP result can be obtained. Other approaches range from distances and rotation thresholds to counting the number of frames passed from the last keyframe insertion.
Another relevant thing to take into account when the graph is constructed is map points creation. In particular, again to maintain the computational effort feasible, a map point culling mechanism can be integrated, where landmarks are removed from the system if they are not observed from enough frames [40] or they are too near to other points (also considering re-projection in a set of frames).
To add a new point, triangulation must be implemented. In particular, in order to obtain valid points, mechanism to avoid bad triangulation must be devised: parallax checking, re-projection thresholds and feature matching threshold are the most important checks to be carried out. Up to this inspections the method usually applied for triangulation is the Linear Triangulation Method: a direct analogue of the DLT method. Given a set of matched points $\{x_i, x_i'\}$ and the projection matrices of the cameras P and P', we need to estimate the set of 3D points that the images observes. We know from geometry that $x = PX$ and $x' = P'X$. Each of this equations can be rewritten s $X \times (PX) = 0$ because the cross product of two vectors of same direction is zero. Analyzing such equation we can notice that the last row of the vector resulting in the left-hand part is linearly dependent from the last two (as in DLT algorithm), so it is not enough constrained to obtain a solution for X. So we need to include the same points seen from the other camera into an equations of the linear form as $AX = 0$, with:

$$A = \begin{pmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ x'p'^{3T} - p'^{2T} \end{pmatrix} \tag{3.78}$$

where p^{iT} are the rows of P and p'^{iT} are the rows of P'. This linear system comprises four equations in four homogeneous unknowns. This is a redundant set, and for this reason the solution is determined only up to a scale (by using an SVD approach as the one shown for DLT). This brings to light the fact that a multi-view optimization (bundle adjustment) is needed in order to observe the correct scale.

Chapter 4

Implementation

In order to understand how methods used for keypoint detection impact the evolution of a SLAM system, a pipeline with a focus on tracking is needed. In this chapter the system proposed as baseline on which to test feature extraction methods is presented. It follows the scheme of a general SLAM system presented in the previous chapter: keypoints detection, initialization, tracking, optimization. An illustration of the modules implemented and how they are connected is presented in the Figure 4.1. In particular, keypoints detection and descriptors extraction is performed when all the image sequences are loaded, in this way the execution performances are not influenced by the method used, only the tracking does. All the system relies on the fact that any image is represented by a set of 2D points and associated descriptors, independently of what algorithm is used for its detection. This allows a better focus on how the results are influenced by them.

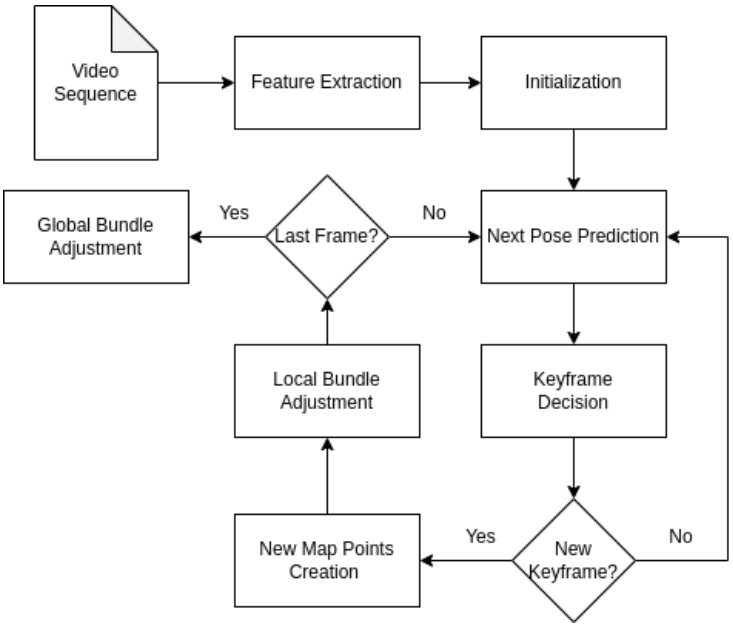


Figure 4.1. The structure of the SLAM system implemented

In this chapter each module is described in details and a lot of references on the theory presented in the previous chapter is done implicitly. Actually, the feature extraction module is illustrated by visualizing how the points are distributed in the different approaches, it follows exactly the methods presented in Section 3.2, so it does not need further explanation in implementation. The code of the whole system is available free on github [1] with the possibility to execute all the test cases presented in this document. In particular it is developed in C++, relying on libraries like OpenCV [3], Eigen [20] and g2o [27]. MATLAB [36] is used to produce visualization from the data produced by the system, while Python and related libraries for deep learning are used for neural features extraction.

4.1 Features Extraction

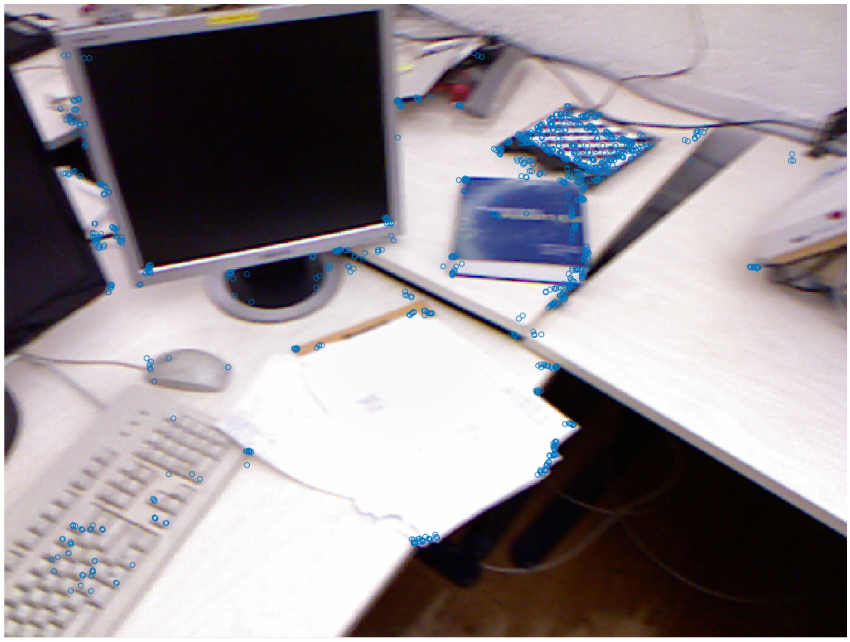
Three different keypoints extractor methods are applied: ORB [45], Superpoint [12] and LF-NET [43]. The ORB features are extracted using the method given in the OpenCV library, whereas Superpoint and LF-NET are taken from the implementation given with the original paper, the first using pre-trained weights from website linked inside our repository, the second using the indoor weights given by the authors. In particular LF-NET is trained on two different datasets: an outdoor and an indoor dataset. We use the second one because all the sequences used for test are obtained in an office environment. Indeed, using the outdoor weights the results are very bad. In Figure 4.2 we can see the keypoints extracted with ORB. It is important to underline that all points are in areas of the image where we have more discriminative components, this will results in denser parts of the environment and empty areas, that surely means bad tracking. The things are much better with Superpoint (see Figure 4.3) and LF-NET (see Figure 4.4). In particular Superpoint is able to detect a lot of points of interest, with a good distribution among all the pixels of the image. LF-NET is also well distributed but contains a lot of point in areas where there is no discriminative texture. This will results in a lot of map points that are no relevant for tracking.

4.2 Initialization

The initialization step is the most delicate part of a SLAM system, for this reason much attention has been paid for this module. To begin, the first and the second frame are taken and all their keypoints are matched using descriptors. Actually, this method leads to a lot of outliers, especially with ORB features, for this reason a method robust to them is needed. Once matches are computed the following steps are executed:

- Computation of the fundamental matrix: here the keypoints couples are used to determine which is the fundamental matrix that explain the transformation between the two camera poses, using the eight-point algorithm. A RANSAC schema is applied for 200 iterations in order to find the bigger set of inliers

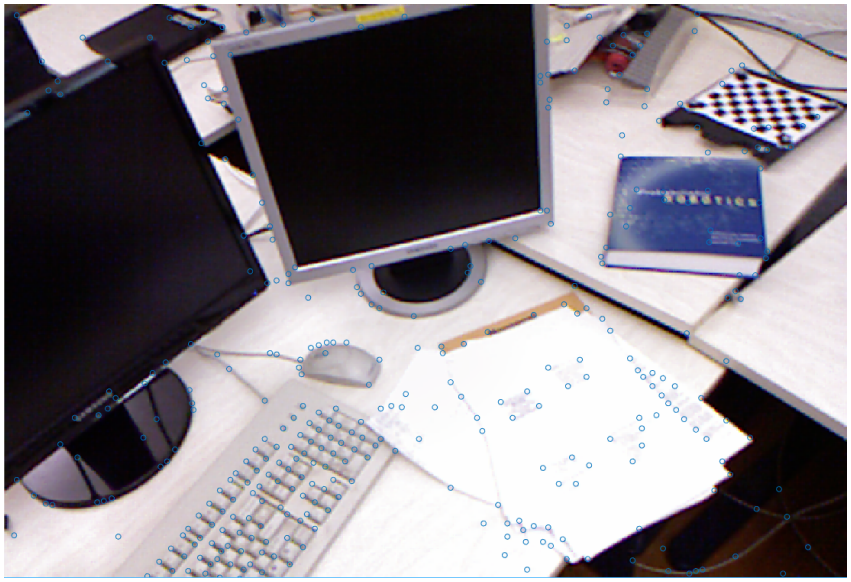
[1] https://github.com/lucalobefaro/SLucAM.git



(a)

(b)

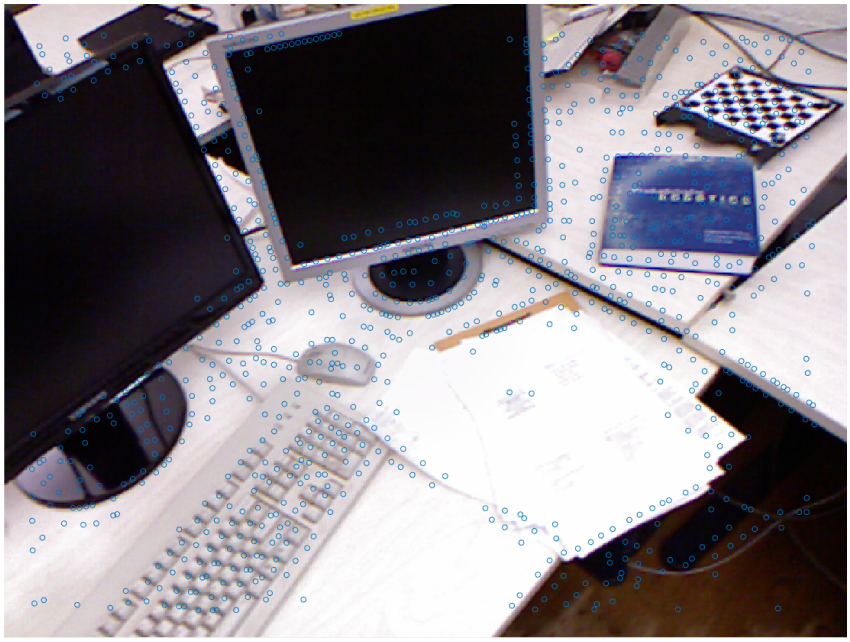
Figure 4.2. Keypoints extracted with ORB on two images, in particular a) is take from the sequence *fr1_desk* and b) from *fr3_structure_texture_far*



(a)

(b)

Figure 4.3. Keypoints extracted with Superpoint on two images, in particular a) is take from the sequence *fr1_desk* and b) from *fr3_structure_texture_far*



(a)

(b)

Figure 4.4. Keypoints extracted with LF-NET on two images, in particular a) is take from the sequence *fr1_desk* and b) from *fr3_structure_texture_far*

with a minimal set of 8 random points for each iteration, then this set is used to compute the best version of the F matrix. In particular each keypoint is normalized by shifting the centroid of the set at the origin and scaling them in such a way the distance from the origin is 1. At each iteration of RANSAC the result is evaluated by computing a score following the approach proposed by ORB-SLAM [39, 40, 6]. In particular two symmetric transfer errors d_{12} and d_{21} are computed on each match [23], and the score is computed basing on such values as:

$$S = \sum_{i \in \#matches} (\rho(d_{12}(x_1^i, x_2^i)) + \rho(d_{21}(x_1^i, x_2^i))) \tag{4.1}$$

where x_1^i and x_2^i are the keypoints of, respectively, image 1 and image 2 at which the match i makes reference, and:

$$\rho(d) = \begin{cases} \Gamma - d & if d < T \\ 0 & if d \geq T \end{cases} \tag{4.2}$$

The constant T is the outlier rejection threshold and it is set to 3.84 [40]. At the end a score F_{score} is computed with the best version of F that is next de-normalized using the inverse of the approach used for normalization [23].

- Computation of the homography: the same approach used for the fundamental matrix is proposed also for the computation of the "best version" of the homography H, using the DLT-algorithm. A RANSAC schema is constructed in order to find the higher number of inliers on which is then computed the final version, this time a minimal set of 4 random points is used at each iteration. A score is produced also in this case, with the same approach, but with a difference symmetric transfer error [23] and with the threshold T that is now set to 5.99. You can notice that the Γ value was set also at 5.99, the reason is that we need to have an homogeneous score both for H and F, so the value of the higher threshold is used [40]. At the end we will have a score H_{score} computed on the best version of H. Also here the keypoints are first normalized and then the final H matrix is de-normalized.

- Rotation only detection: the approach used in ORB-SLAM to compute the scores associated to fundamental and homography matrices is used in order to exploit an important result: such scores can be used to determine if we have a rotation only or a planar situation. As we know, such situations cannot be explained with a fundamental matrix, so they must be avoided in our system, where a fundamental based approach is used for initialization. Actually ORB-SLAM proposes this formula:

$$R_H = \frac{H_{score}}{H_{score} + F_{score}} \tag{4.3}$$

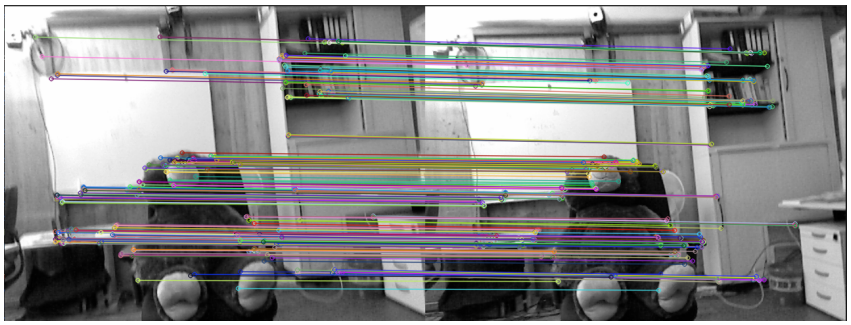
to determine if the transformation between the two images can be explained "better" by the homography. In particular, when $R_H \leq 0.45$ then the fundamental matrix can be used without problem. In the implementation presented in this thesis this approach is exploited as check to understand the validity of the computed F and, if we have a rotation-only case or a (nearly) planar situation, the current couple of images is discarded and the initialization is performed again between the next frame and the first one, until a good result is obtained.

- Map initialization: once a good couple of images is found with a F matrix associated, we can use it to compute the essential matrix from which to extract the pose of the first frame with respect to the second (actually the first valid frame found to associate to the first). Remember that four solutions are derived from an essential matrix and the transformation matrix that allows to triangulate the higher number of 3D points, using only inliers from the matches, is taken as valid. We should remark again the importance of the initialization in order to understand that now further checks must be applied: at each new point triangulated, parallax and re-projection error is computed before to evaluate such triangulation as good. At each possible solution will be associated a score, that is the number of good 3D points computed. Then only if, between the four solution, we have a clear winner (it as at least the 60% of matches triangulated and it has a score higher than the 80% of the second best solution) and it has enough parallax we can output a solution. Otherwise, again, this frame is discarded and the next one is used in couple with the first one. Important is that, when more than 10 frames have been discarded, we assume that it is the first one to be a bad one, so it is discarded and the procedure is repeated using the next one as first.

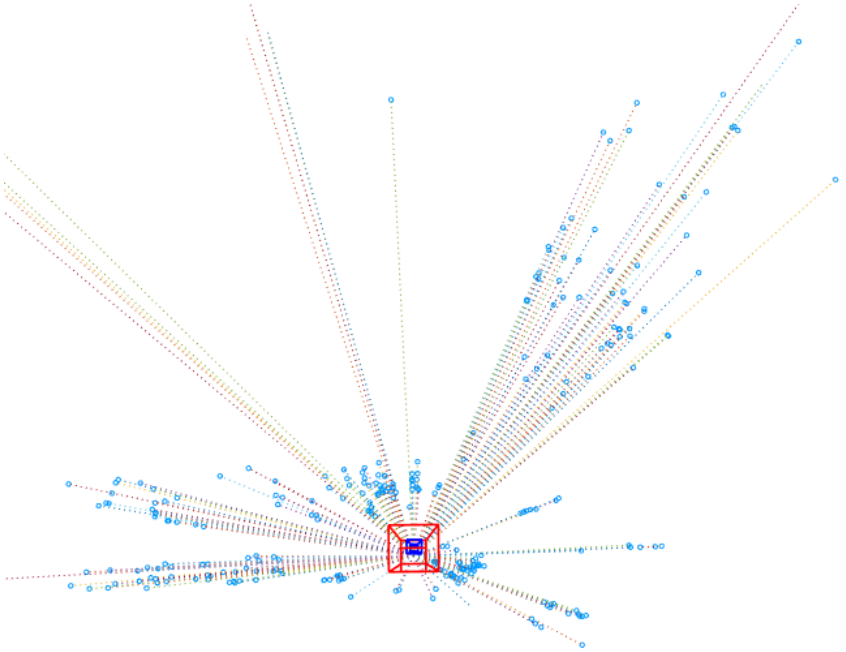
- Keyframes insertion: at this step we will get only if we have a valid set of map points and a valid pose of a camera with respect to the first one (actually the pose of the first valid frame found in combination the first frame of the video sequence). This data can be used to initialize our graph: the first frame's pose is assumed at the origin and inserted as keyframe togheter with the second one and and all the map points are used as "map keypoints" (3D points that represent a node in the graph). Also the connection pose-to-pose and pose-to-landmark are memorized.

- Optimization: to conclude the initialization a global bundle adjustment is performed in order to refine the initial map construction, also the scales are adjusted in order to avoid too far keypoints in the map.

An example of initialization performed on the fr1_teddy sequence of the TUM RGB-D dataset [48] with ORB features matching is depicted in Figure 4.5.



(a)



(b)

Figure 4.5. Initialization performed on the fr1_teddy sequence of TUM RGB-D dataset [48] with ORB features. In figure **(a)** are shown the inliers matches found by the initialization step, as you can see, we do not have any outliers survived to our filters. In figure **(b)** the initial map estimation is represented graphically, the map respect the keypoints seen and the poses of the two images.

4.3 Keyframe and map points insertion

A keyframe is a relevant image from the input video sequence whose representation is used as node for the SLAM graph. For this reason, each time a new one is inserted different things must be done:

- A local map is computed for it, this will be useful during tracking. The local map is computed by taking all the map points seen from the current frame and computing, among all keyframes in the graph, those that share at least 15 observations with it: these will represent the local keyframes. Then, all map points observed from the local keyframes are used as local 3D points.
- Information about which keyframe observe the new one (at least the previous one) is memorized.
- The descriptor of each map point observed from the current keyframe is updated (as explained in a while)

Graph is not made of only camera poses, but also map points, for this reason a good data structure must be maintained for each of them, containing relevant information useful during tracking and optimization. A new map point is added each time two images triangulate a new one, so when it is spawned two observations are memorized for it (from which frame it is observed and from which image point) and the corresponding descriptors. For this reason these information are attached at each map points:

- The estimated position in world frame.
- The set of frames that observe it (and from which image point).
- A representative descriptor, obtained from all the image points that observe it.

The position is estimated the first time it is triangulated and then updated only during optimization processes. The set of frames that observe the point is expanded each time a new frame observation for it is detected. Finally, the descriptor is computed in this way:

- Take all the descriptors of all the image points associated to this map point.
- Compute the mutual distance between them.
- Save as representative descriptor the one with the lower median distance

This process is executed only when a new keyframe that observe such point is inserted in the graph, in order to maintain the map updated but, because this approach is heavy to compute, avoid too much load on computing resources.

4.4 Tracking

The core of the implemented SLAM system is the tracking module. It processes each new frame in order to predict the trajectory that the camera follows in the input video sequence. So first it needs to understand what is the pose of the new frame and then decide if it is a relevant frame to be inserted in the optimizable graph as a new node (keyframe insertion). In such case a local refinement is performed in order to better constrain the trajectory to the environment. Actually, computing a correct pose from each frame is important in order to not loose the track, so we propose to subdivide the pose estimation into two steps: first an initial guess is predicted with a projective ICP algorithm computed on the previous frame's pose and keypoints associated through previous images associations (see Section 4.4.1), then this initial guess is used to set up a complete pose optimization problem using a consistent part of map points (see Section 4.4.2).

4.4.1 Initial pose estimation

Given a new frame, represented by a set of keypoints and associated descriptors, the first step to perform in the tracking module is to understand what are the map points seen from the current image. This is needed in order to boot a projective ICP algorithm correctly. So each keypoint in the frame is matched with keypoints in the last 3 previous frames, using descriptors distances and taking only those points whose distance is lesser than a certain threshold. If not enough matches are found (at least 100 are required to be constrained enough), then the associations are searched in the last 6 frames. In particular, only those matches that refers to keypoints for which we already have a 3D prediction are considered. In this way, at a subset of points on the current image, we have associated a set of 3D points predictions.
Once we have a consistent number of keypoints matched with 3D map points we have the possibility to compute, using a projective ICP algorithm, the pose of the new camera. In particular, we assume that the motion is not too fast, so the previous frame's pose is used as initial guess. The strength of this approach is that the projective ICP algorithm also discards outliers, so only matches that are "good enough" are maintained.

4.4.2 Pose optimization

From the previous step we receive a robust initial guess for the camera pose that we are integrating. Now, it can be exploited to search for more 2D keypoints to 3D map points associations. We know that at each keyframe in the system is associated a local map, so we take each map point in the local map of the last keyframe and project them into the image plane associated to the predicted pose. New data association is searched in this way, for each map point in the local map:

- first a check if it is in front of the camera is executed
- if yes it is projected on the image plane
- if the projected point falls outside the image plane it will be discarded

- the descriptor distance between the projected point and each image point is computed and inserted in a vector

then we search, for each valid local map point, the keypoint that has the minor descriptor distance and we take them as associated only if their Euclidean distance is under a threshold. To avoid duplicates, it is performed also a check on if the current image point is already associated. This exploit the fact that, assuming the pose obtained from the initial pose estimation step as good enough, from it we observe the projected map points on the image plane near to keypoints that should represent them, as long as they have "near" descriptors.
Once all possible associations between image points and map points are found a projective ICP is performed on these matches, with the pose obtained in the previous step as initial guess. The result, togheter with the inliers matches, is used in the keyframe decision step.

4.4.3 Keyframe decision

Given a frame, represented by an image and a set of keypoints and keyframes associated, and a set of matches with map points, we need to decide if this is a relevant frame, and if yes it must be used as keyframe. Important is to notice that when a new keyframe is spawned, then new points are triangulated and a local bundle adjustment is executed. For this reason it is important to decide to spawn a new keyframe each time we risk to loose tracking and this can happen when we have too few inliers from the previous step and we are rotating to fast in new area of map where, maybe, we do not have no point for pose optimization. In particular, during tracking, these simple checks are performed:

- the number of matched points is lower than 100
- at least N frames as been integrated from the last keyframe or we the rotation of the current pose with respect the previous one is too large

The first check ensure that we track always enough map points, in order to obtain a big enough local map that will be exploited from the next frames and that allows to execute a constrained enough local bundle adjustment. Furthermore, it allows to triangulate new points and make greater the number of inliers for the next frame. The second check ensure enough distance (excluding idle video sections) from consecutive keyframes, in order to have the possibility to observe better the scale during the optimization step, this maintaining a good control over the rotation, in order to avoid the problem illustrated before. The value of N is chosen to be 20, but can be set to 5 when we have too fast sequences, in order to triangulate enough points in new sections of map.

4.4.4 Local bundle adjustment

Every time a new keyframe is inserted (refer to Section 4.3) an optimization is performed on a local part of the map. Launching a global bundle adjustment is too heavy, especially when the map becomes huge, for this reason a local approach is

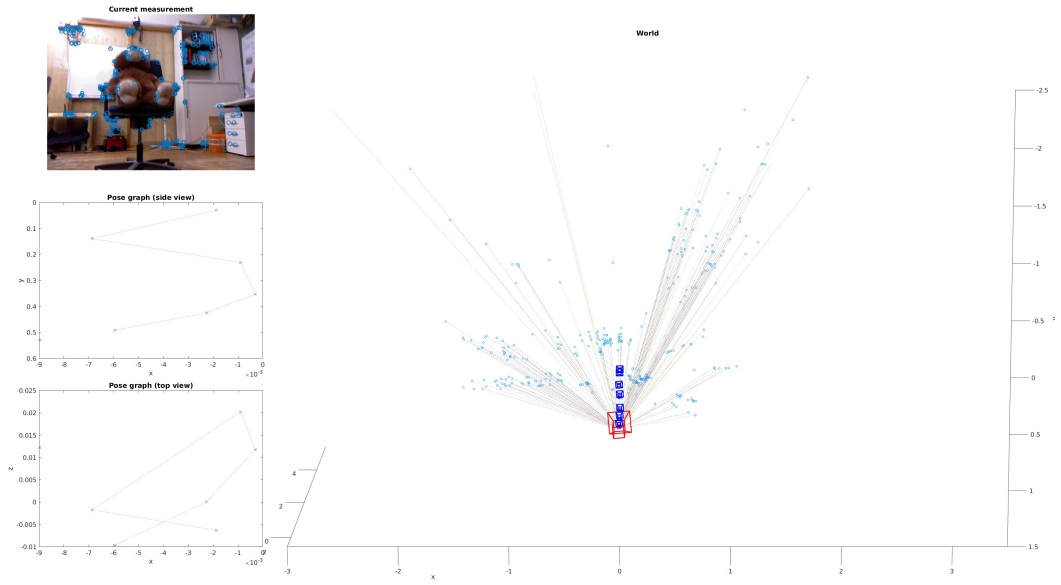
used. In particular, given the new frame, a local bundle adjustment is performed on these nodes:

- the last n keyframes (with $n = 3$) included the new one: these nodes represent the poses of the camera that are more relevant in the recent trajectory and that must be optimized;
- all the map points observed by the keyframes chosen in the previous point: these nodes represent the landmarks observed by the camera poses that must be optimized, so the observations pose-to-landmark represent the constraints of our local map, also these points are optimized;
- the $N - n$ keyframes (with $N = 8$) that precede the keyframes chosen in the first point: these are not optimized, but are useful in order to add constraints to points observed and maintain the results "attached" to the trajectory and consistent with the global environment

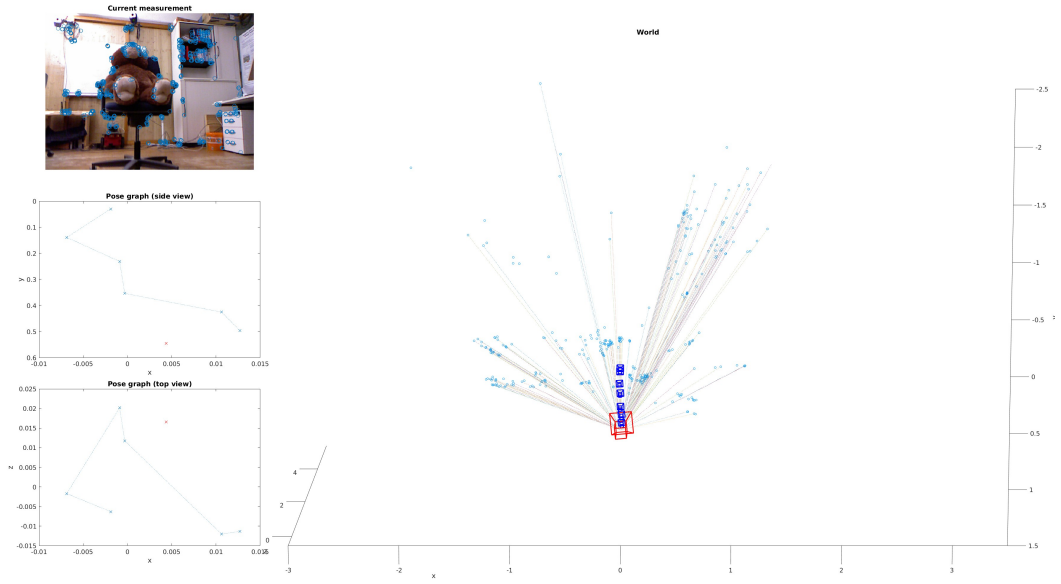
The optimization step is implemented using the g2o library [27] with the Levenberg-Marquardt algorithm and 10 iterations are enough to converge to a stable result. This phase is really fast when compared with a global optimization, ensuring real-time performances. In Figure 4.6 are shown the results of a local bundle adjustment on the local map points and on the trajectory.

4.5 Final Refinement

Once all the frames of the input video are processed, the front-end phase is terminated and the whole graph is constructed. At this point all the constraints of the environment are ensured so we can adjust the scale of the map points and adjust the trajectory according. For this purpose a global bundle adjustment is needed. Also in this case the optimization is implemented using the g2o library [27] with the Levenberg-Marquardt algorithm, and 10 iterations are performed. In Figure 4.7 the improvements obtained with a global optimization are shown. It can be notice the improvement is greater on the global scale and keyframes positions, but it is almost not visible on the trajectory shape, this thanks to the local bundle adjustment executed at each keyframe insertion. It is important to notice that we have a closure on the loop also without an explicit loop detection and closure.

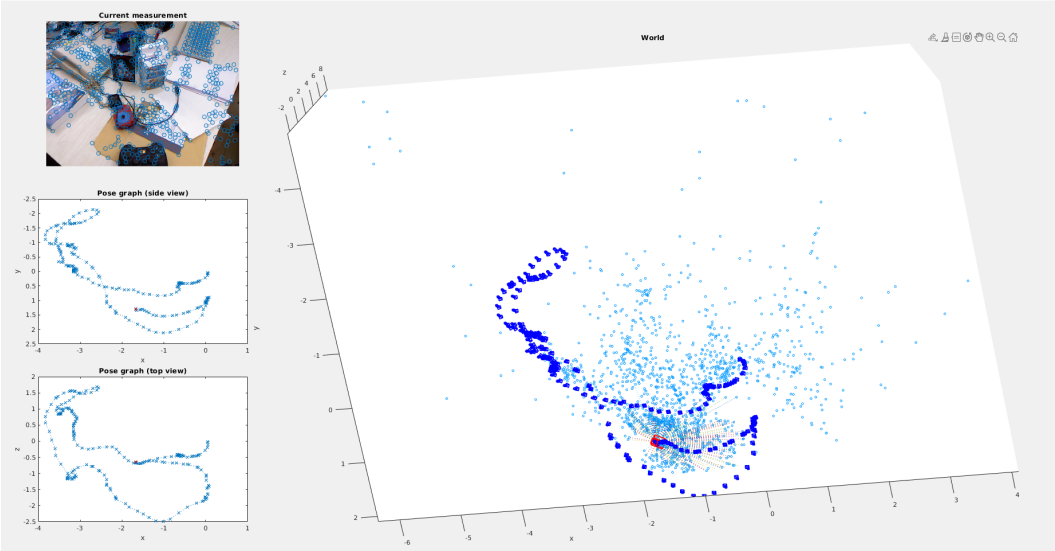


(a)

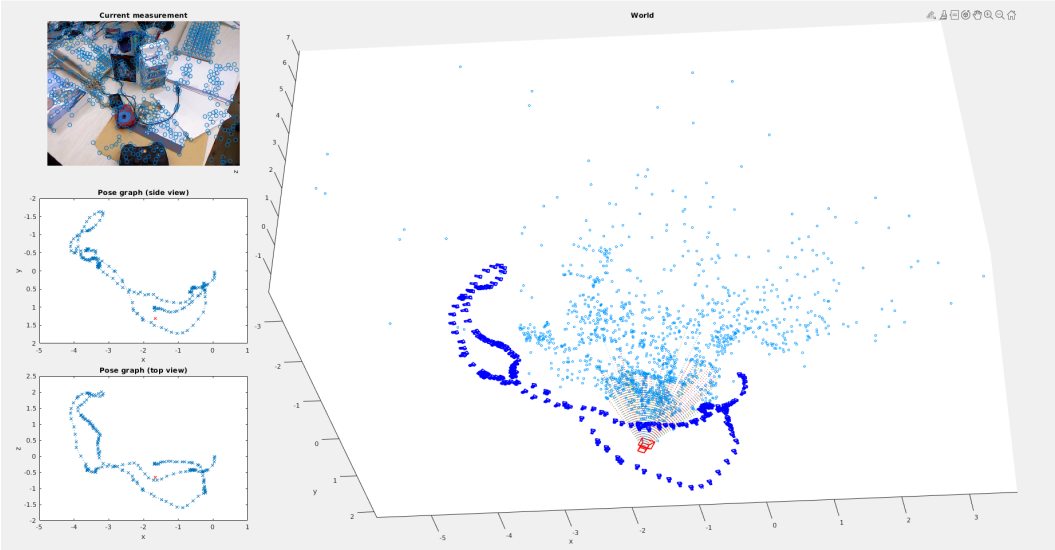


(b)

Figure 4.6. Local bundle adjustment performed during tracking on the fr1_teddy sequence of TUM RGB-D dataset [48] with ORB features. In figure **(a)** the map obtained from the frame before the keyframe insertion is shown. In figure **(b)** the keyframe has been inserted and a local bundle adjustment has been performed. As we can see the map is adjusted according to all the previous keyframes constraints, obtaining a result more consistent with the real map.



(a)



(b)

Figure 4.7. Internal representation obtained from the sequence $fr1\_desk$ using Superpoint: a) before the global BA and b) after the global BA. In b) the red camera represent the last frame integrated and, because it is not spawned as keyframe it was not optimized and it is maintained in its previous position: this to better show the difference between the poses and scale before and after the global BA. The blue camera represents all the spawned keyframes.

Chapter 5

Results

The sequences proposed in the TUM RGB-D dataset [48] are excellent to evaluate the tracking process of a SLAM system especially because they are provided with accurate ground truth obtained from an external motion capture system. In particular, each sequence contains color and depth images of each frame, captured with a Kinect sensor. The data was recorder at 30Hz and at resolution of 640×480. The ground-truth was obtained from a high-accuracy motion-capture system with eight speed tracking cameras at 100Hz. The calibration parameters of each camera used is given, useful to correctly compute all the mathematical formulation of a SLAM system. Also the accelerometer data from the Kinect is given and they also propose an evaluation criterion for measuring the quality of the estimated camera trajectory of a visual SLAM system.
For our test, only RGB images are used and the proposed evaluation criteria is accepted as a good way to determine how good is each kind of features used. In particular two metrics are used: the absolute trajectory error (ATE) and the relative pose error (RPE). The ATE value measures the difference between points of the true and the estimated trajectory, aligning them basing on the timestamp. For this reason each keyframe produced by our system is saved with the ground truth timestamp associated, in such a way we can obtain a good aligning of the predicted poses with the real ones, in order to focus only on the distances in terms of space. In particular, we compute the error basing only on the keyframes of our SLAM system, because they are the only relevant pose produced and the only optimized inside the graph.
The second value, the RPE, computes the error in the relative motion between pairs of timestamps. In particular a fixed window size is used, considering only pose pairs that gave a distance of delta 1.0. This technique is good to evaluate the drift. The RPE is computed using the same timestamp-poses couples used for ATE, so the same considerations are valid also in this case. Actually, three values are extracted from the RPE, in order to have also an idea of how large is the error of the rotational part of the poses: the mean of the rotational error in degrees (that we will call MRE) and the minimum and maximum value of the rotational error, in order to have an idea of the variance (that we will call respectively minRE and maxRE).
In Table 5.1 are shown the results obtained on different sequences. In particular are proposed only those sequences of the dataset that allow to test the different

points features under a good number of different situations, in order to stress how the three approaches affect the tracking. For each sequence are shown the three values introduced before togheter with the number of points produced for the map. This in order to understand also how complex is the internal representation obtained with respect to the results.

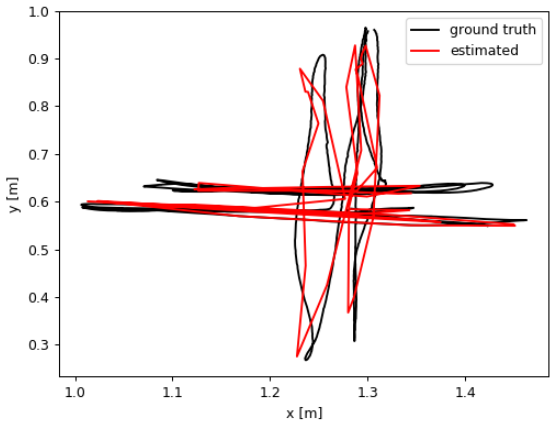
The first sequence in the table ($fr1\_xyz$) is used as a baseline, to understand if the system works well: it has a duration of $30.09s$ and the trajectory explored has a length of $7.112m$, with very slow movements, actually a mean of $0.244m/s$ and $8.92deg/s$. The camera is pointed at an office desk with a lot of objects that allow to produce a lot of map points to help the tracking. The sequence contains only translatory motions along the principal axis, while the orientation is kept mostly fixed. In this case all the three features used are able to produce good results, as expected, but these first results are also able to give an idea of the differences between the three techniques. In particular Superpoint is capable to follow the trajectory producing less map points and obtaining a lower error in the rotational part (which is, however, no relevant in this sequence). LF-NET, instead, is the winner, with a lower error in the trajectory and a lower mean in the rotational error. We can just conclude two thing: the system is able to track video sequences and the neural approaches can produce better results also when the error with ORB is low enough. In Figure 5.1 are depicted the sequences obtained, togheter with the ground truth.
Same story for the second sequence ($fr2\_xyz$) that is a longer version of the sequence $fr1\_xyz$, with a duration of $121.48s$ and a length of $7.029m$. As you can see in this case we have a higher duration and so an higher number of frames to elaborate. Also in this case the results are quite similar, and good for all of them, with Superpoint that performs better than the others on translational part, whereas LF-NET and ORB performs better on the orientation. But, if wee look at the predicted trajectories in Figure 5.2 we can notice the first weakness of LF-NET: it has some error in the predictions on the y axis. This is certainly due to the dependency on the kind of features learned during training, that makes LF-NET less performing when used on environments too different from training data.

The third sequence ($fr2\_desk$) is more interesting, because allows to understand the behavior of the system on accumulated error and so obtain an idea on how the different features influences the drift. Actually, the duration of the video is $99.36s$ with a trajectory $18.880m$ long. Also in this case we do not have too fast motion with an average of $0.193m/s$ and $6.338deg/s$: this allows to focus only on the drift accumulated. Also in this sequence we have an office scene, with two desks covered by different objects. The camera moves along the two tables, until the loop is closed. As you can see from the Table 5.1 both neural approach produce an acceptable error, whereas ORB is not able too keep the tracking along all the sequence. In particular, as we can see also from Figure 5.3, Superpoint is the only one that maintain an optimal trajectory, also with a lower number of points generated.

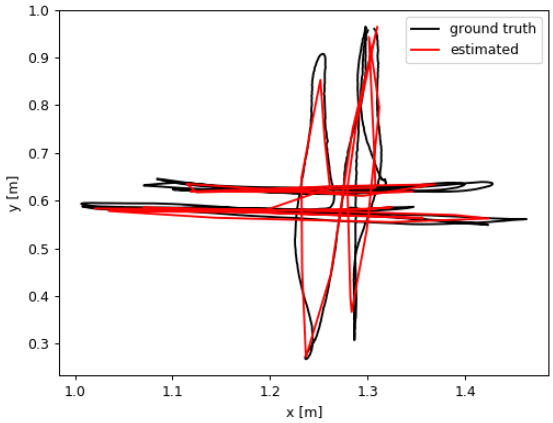
The sequence $fr1\_desk$ is good to understand how the system can handle noise and fast movements. Indeed, here we have a very fast motion, especially in the

		ORB	Superpoint	LF-NET
fr1_xyz	ATE [m]	0.016	0.0132	**0.0127**
	MRE [deg]	1.2337	1.3280	**1.1720**
	minRE [deg]	0.3334	**0.1519**	0.2102
	maxRE [deg]	2.4521	**2.8897**	3.0275
	#Points	1060	**533**	614
fr2_xyz	ATE [m]	0.025	**0.0188**	0.0893
	MRE [deg]	0.4387	**0.4302**	0.8548
	minRE [deg]	0.0314	**0.0132**	0.0772
	maxRE [deg]	**2.0693**	8.4183	4.9315
	#Points	1523	**1305**	2484
fr2_desk	ATE [m]	1.0256	**0.0585**	0.3377
	MRE [deg]	3.1771	**0.4572**	3.4832
	minRE [deg]	0.045	**0.0462**	0.0872
	maxRE [deg]	20.2617	**1.2277**	34.4174
	#Points	5009	**1745**	3147
fr1_desk	ATE [m]	X	**0.0670**	0.4272
	MRE [deg]	X	**2.0424**	12.57
	minRE [deg]	X	**0.3185**	0.4618
	maxRE [deg]	X	**4.8464**	54.3548
	#Points	X	**1490**	3029
fr3_structure_texture_far	ATE [m]	0.0311	**0.0241**	0.0246
	MRE [deg]	0.5115	0.4328	**0.3674**
	minRE [deg]	0.1798	0.1081	**0.0601**
	maxRE [deg]	1.4574	1.1927	**0.6017**
	#Points	1078	**889**	2170
fr3_structure_texture_near	ATE [m]	0.1374	**0.0541**	0.5525
	MRE [deg]	1.48	**0.5768**	6.5324
	minRE [deg]	0.1015	**0.0869**	0.5667
	maxRE [deg]	4.7513	**1.9704**	30.8324
	#Points	1752	**1024**	1907
fr3_nostructure_texture_near	ATE [m]	0.0513	**0.0148**	1.7562
	MRE [deg]	0.8953	**0.7431**	9.644
	minRE [deg]	**0.0545**	0.0616	0.5745
	maxRE [deg]	**2.3145**	3.2280	27.711
	#Points	2289	**1251**	2608

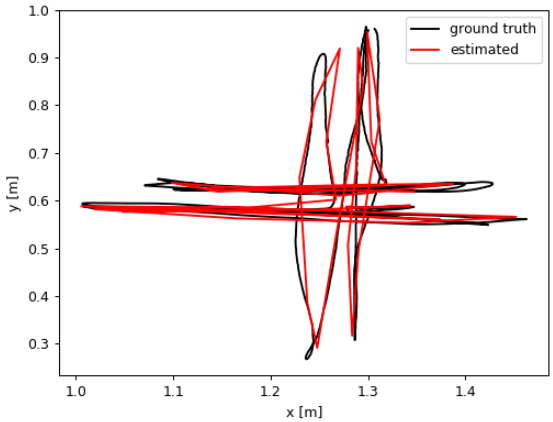
Table 5.1. Results obtained on different sequences. In particular, for the translational part the ATE (in meters) is indicated, whereas, for the rotational part, we have the mean error (MRE), the min (minRE) and the max (maxRE) errors, in degrees. It is shown also the number of points generated. In bold are indicated the best results (lower is better). *X* indicates that the tracking is loose at some point along the sequence.



(a)

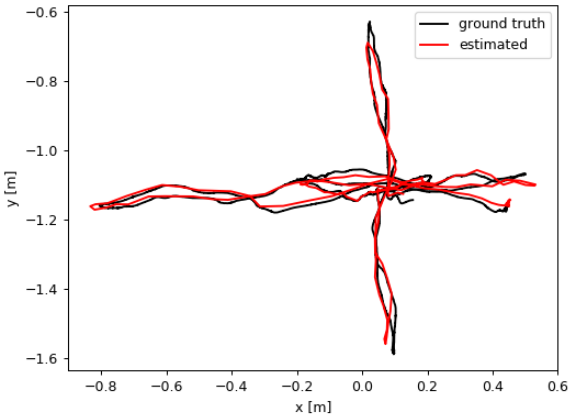


(b)

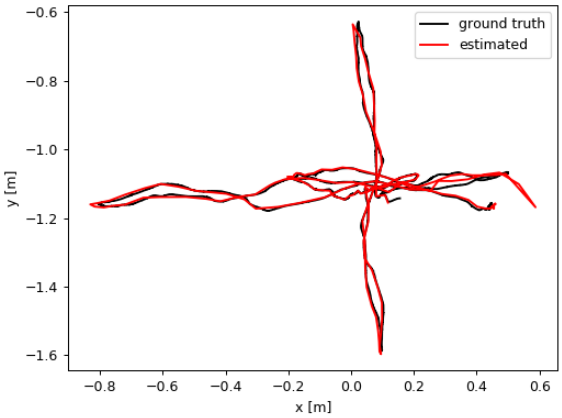


(c)

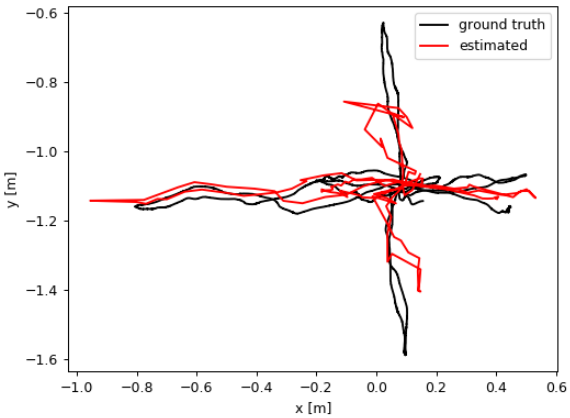
Figure 5.1. Trajectories predicted on the sequence $fr1\_xyz$ with: a) ORB, b) Superpoint, c)LF-NET.



(a)

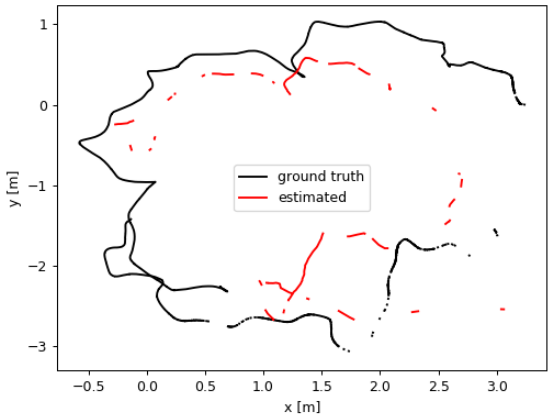


(b)

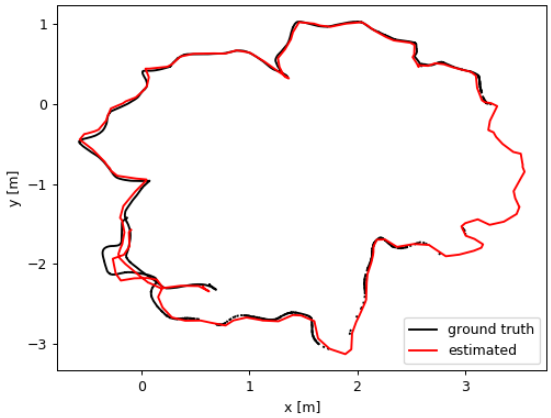


(c)

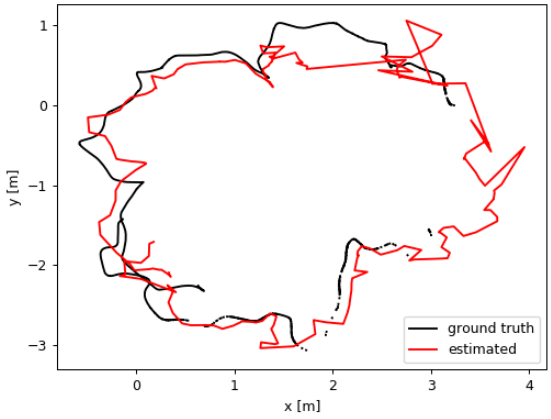
Figure 5.2. Trajectories predicted on the sequence $fr2\_xyz$ with: a) ORB, b) Superpoint, c)LF-NET.



(a)

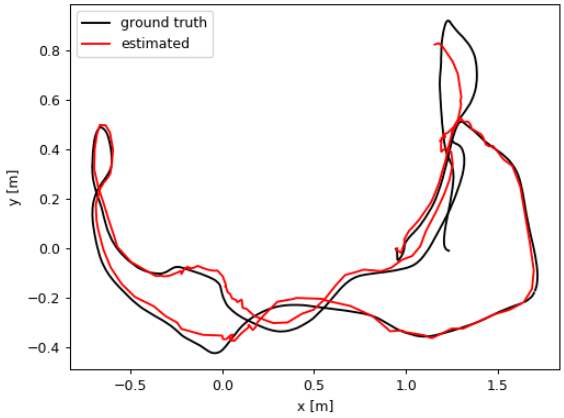


(b)

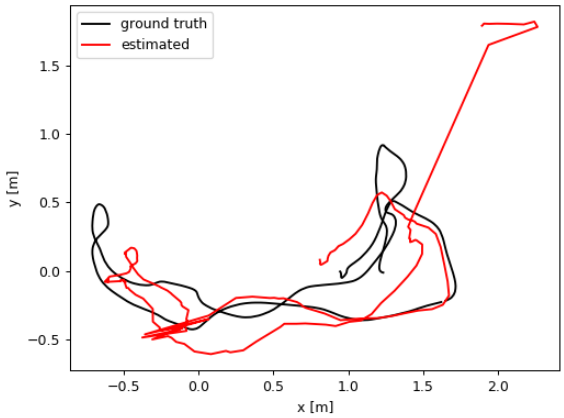


(c)

Figure 5.3. Trajectories predicted on the sequence *fr2_desk* with: a) ORB, b) Superpoint and c)LF-NET.



(a)



(b)

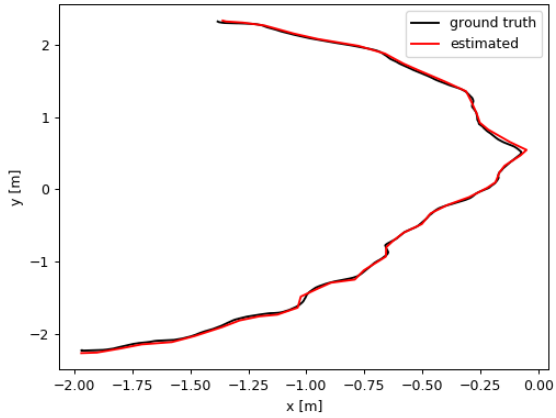
Figure 5.4. Trajectories predicted on the sequence $fr1\_desk$ with: a) Superpoint, b)LF-NET.

angular velocity, with an average of $23.327 deg/s$. We have a trajectory $9.263m$ long with a duration of $23.4s$. Furthermore, we have a lot of motion blur, with a lot of frames that are difficult to handle. The results are clear enough to understand that the classical approach with ORB is not able to produce a result, because the tracking is loose at some point, due to very noisy frames. In particular, only using Superpoint features the system is able to close the trajectory with low error, both in angular and translational part. As we can see from the trajectories predicted from the two neural approaches in Figure 5.4 also LF-NET is able to produce a path that is near to ground truth, but goes away in some part of the trajectory, especially at the beginning, probably due to the fact that is not able to maintain a good scale of the map points.

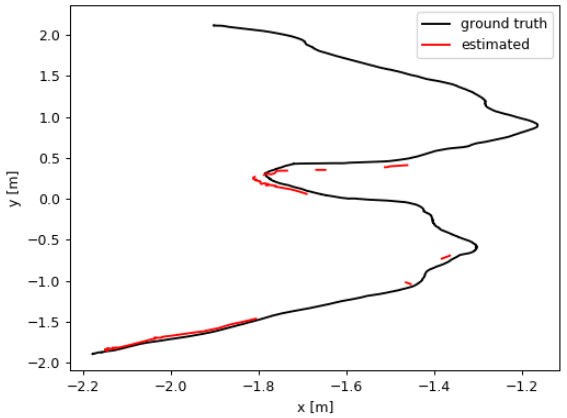
The last three sequences are useful to test how the features can handle different texture contexts. In particular, the sequences $fr3\_structure\_texture\_far$ and

$fr3\_structure\_texture\_near$ are obtained in a room in which different wooden panels are placed in such a way we can obtain a structure that is no planar and, on each panel, we have several posters which have strong texture. The first one has a trajectory $5.884m$ long and has a duration of $31.55s$ whereas the second one lasts $36.91s$ with a trajectory of $5.05m$. The trajectories have both a zig-zag shape, but the second one, as the name suggests, is recorded with the camera near to the panels, this makes the tracking more difficult. Indeed, whereas we obtain good results with all the approaches on the first sequence, we can see that only using Superpoint we are able to produce near-to-truth results on the second sequence. In particular LF-NET behave worst than ORB and the same happen on the last sequence ($fr3\_nostructure\_texture\_far$). This is due to the fact that it is heavily dependent on the training data, as explained in the first section of the previous chapter. Actually, we use a network trained on outdoor images, data that contains very different visual features with respect to the textures proposed in these sequences. Furthermore, the last sequence do not contains structure, it has a planar environment with a lot of conference posters, with a duration of $56.48s$ and a length of $13.456m$ (this sequence, like the previous two, has also not too fast movements, in order to allow to concentrate on the texture's features). Surprisingly, both ORB and Superpoint are able to produce very good results in this case, with Superpoint that confirms its superiority with respect to the other approaches: it is very good in generalizing features learned during the training stage. In Figure 5.5 the trajectory produced using Superpoint are depicted and it is easy to see how good is the system in tracking exploiting these neural features.

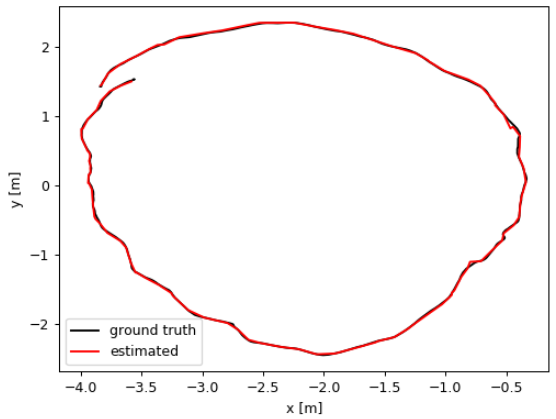
To conclude, we can affirm that, on this dataset, in monocular visual SLAM tracking, neural features outperform ORB in all situations, especially in sequences with fast movements and with a lot of motion blur. This is certainly due to more distribution of points among the pixels of the images that allow to cover all the seen part of the environment with map points to help the tracking. In particular, Superpoint outperforms in almost all the situation (especially the more difficult) LF-NET and this is due to the fact that the second neural approach is more dependant from training data with respect to Superpoint, that is instead able to generalize better, due to how it is trained, using self-produced synthetic features. Furthermore, analyzing the Table 5.1 we can see that Superpoint produces also less map points than the other approaches, that is an advantage because it allows to maintain lighter the internal representation of the algorithm. We can also conclude that the monocular visual SLAM system proposed is capable to produce good results on tracking, exploiting all the best approaches present in literature nowadays. Furthermore, it can handle, in combination with neural approaches like Superpoint, also sequences with motion blur and fast movements.



(a)



(b)



(c)

Figure 5.5. Trajectories predicted using Superpoint features on the sequences: a) *fr3_structure_texture_far*, b) *fr3_structure_texture_near*, c) a) *fr3_nostructure_texture_near*. In particular, in Figure b) we do not have a complete predicted trajectory because of low frequency of the keyframes produced.

Chapter 6

Conclusions

In this thesis a novel monocular visual SLAM system is presented, constructed with a focus on the tracking stage. Great importance was given in the initialization stage, in order to obtain a first representation that is robust and not ambiguous, this to avoid that bad keyframes and map points at the beginning produce problems in the following stages of the SLAM algorithm. Planar cases and situations with low inliers are discarded until a good couple of frames is found, making all the process robust. Different techniques like RANSAC and threshold based approaches are adopted, taking inspiration to different methods presented in literature.
The tracking stage is then introduced, accompanied by graph-based definition of SLAM. Great focus is given by optimization stage, to exploit the graph representation. In particular a local map approach is used in order to maintain the execution real-time, although it was not the goal of the project. Good results are achieved, obtaining low errors on different sequences on the TUM RGB-D dataset [48], even in the most difficult.
ORB features are used as baseline to go towards the second proposal of this thesis: comparison of different features on the tracking stage. In particular two neural approaches are proposed: Superpoint [12] and LF-NET [43]. They promise to outperform classic features approaches, like ORB, especially because they produce more dense representations with points scattered along the images, covering more parts of them. They also promise more matching of similar features. Indeed, during the testing stage, both approaches outperform ORB, in almost all the sequences proposed. In particular, Superpoint is able to produce results with low errors, even in situations with very fast displacements and motion blur.
The thesis finally shows how the method proposed, in combination with the best features, Superpoint, are able to produce very good results on all the sequences proposed for test. Impressing results are obtained during sequences that are fast, with motion blur or with planar environment.

6.1 Future Work

Interesting can be to explore different solutions on the same system, in particular:

- Extend the system with a loop-detection and loop-closure module, testing how neural features impact also this stage of a visual SLAM system

- Test the system also on map reconstruction, visualizing the result with respect to a ground truth; this would require some sort of point culling to discard points not useful for representation
- Extend the system in order to handle long-term maps, by "forgetting" previous part of the map in order to maintain the execution time constant also with huge environments.
- Handling of dynamic objects, in order to support tracking also in video sequences with animated agents in the environment.
- Improve the system in performances, analyzing them in a specific way, under different situation; implementing also a GPU based approach to obtain faster results.

Bibliography

[1] T. Bailey and H. Durrant-Whyte. “Simultaneous localization and mapping (SLAM): part II”. In: *IEEE Robotics Automation Magazine* 13.3 (2006), pp. 108–117.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “SURF: Speeded up robust features”. In: vol. 3951. July 2006, pp. 404–417. ISBN: 978-3-540-33832-1.

[3] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[4] Cesar Cadena et al. “Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32 (June 2016).

[5] Michael Calonder et al. “BRIEF: Binary Robust Independent Elementary Features”. In: vol. 6314. Sept. 2010, pp. 778–792. ISBN: 978-3-642-15560-4.

[6] Carlos Campos et al. “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM”. In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1874–1890.

[7] Bruce Canovas et al. “Speed and Memory Efficient Dense RGB-D SLAM in Dynamic Scenes”. In: *IROS 2020 - IEEE/RSJ International Conference on Intelligent Robots and Systems*. Las Vegas, United States: IEEE, Oct. 2020, pp. 4996–5001.

[8] J.A. Castellanos et al. “The SPmap: a probabilistic framework for simultaneous localization and map building”. In: *IEEE Transactions on Robotics and Automation* 15.5 (1999), pp. 948–952.

[9] Chang Chen et al. “A Review of Visual-Inertial Simultaneous Localization and Mapping from Filtering-Based and Optimization-Based Perspectives”. In: *Robotics* 7.3 (2018). ISSN: 2218-6581.

[10] Kang Chen, Yu-Kun Lai, and Shi-Min Hu. “3D indoor scene modeling from RGB-D data: A survey”. In: *Computational Visual Media* 1 (Dec. 2015).

[11] Frank Dellaert and Michael Kaess. “Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing”. In: *I. J. Robotic Res.* 25 (Dec. 2006), pp. 1181–1203.

[12] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. “SuperPoint: Self-Supervised Interest Point Detection and Description”. In: June 2018, pp. 337–33712.

[13] Abdelhamid Dine et al. “Efficient implementation of the graph-based SLAM on an OMAP processor”. In: *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)* (2014), pp. 1935–1940.

[14] H. Durrant-Whyte and T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110.

[15] R.M. Eustice, H. Singh, and J.J. Leonard. “Exactly Sparse Delayed-State Filters”. In: *Proc. of the IEEE Int. Conf. on Robotics Automation (ICRA)* (2005), pp. 2428–2435.

[16] Olivier Faugeras and F. Lustman. “Motion and Structure from Motion in a Piecewise Planar Environment”. In: *International Journal of Pattern Recognition and Artificial Intelligence - IJPRAI* 02 (Sept. 1988).

[17] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782.

[18] Jorge Fuentes-Pacheco, Jose Ascencio, and J. Rendon-Mancha. “Visual Simultaneous Localization and Mapping: A Survey”. In: *Artificial Intelligence Review* 43 (Nov. 2015).

[19] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters”. In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46.

[20] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[21] Jianjun Gui et al. “A Review of Visual Inertial Odometry From Filtering and Optimisation Perspectives”. English. In: *Advanced Robotics* 29.20 (Sept. 2015), pp. 1289–1301. ISSN: 0169-1864.

[22] D. Hahnel et al. “An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 1. 2003, 206–211 vol.1.

[23] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. New York, NY, USA: Cambridge University Press, 2003. ISBN: 0521540518.

[24] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[25] Guoquan Huang. *Visual-Inertial Navigation: A Concise Review*. June 2019.

[26] Rong Kang et al. *DF-SLAM: A Deep-Learning Enhanced Visual SLAM System based on Deep Local Features*. Jan. 2019.

[27] Rainer Kümmerle et al. “G2o: A general framework for graph optimization”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3607–3613.

[28] Mathieu Labbé and François Michaud. “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation”. In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446.

[29] Aomei Li et al. “An Improved FAST+SURF Fast Matching Algorithm”. In: *Procedia Computer Science* 107 (Dec. 2017), pp. 306–312.

[30] Ruihao Li, Sen Wang, and Dongbing Gu. “DeepSLAM: A Robust Monocular SLAM System With Unsupervised Deep Learning”. In: *IEEE Transactions on Industrial Electronics* 68 (2021), pp. 3577–3587.

[31] Ruihao Li et al. “UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning”. In: (Sept. 2017).

[32] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *ECCV*. 2014.

[33] G LoweDavid. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* (2004).

[34] Feng Lu and Evangelos E. Milios. “Globally Consistent Range Scan Alignment for Environment Mapping”. In: *Autonomous Robots* 4 (1997), pp. 333–349.

[35] Andréa Macario Barros et al. “A Comprehensive Survey of Visual SLAM Algorithms”. In: *Robotics* 11.1 (2022). ISSN: 2218-6581.

[36] MATLAB. *9.12.0.1884302 (R2022a)*. Natick, Massachusetts: The MathWorks Inc., 2022.

[37] Michael Montemerlo et al. “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem”. In: Nov. 2002.

[38] E. Mouragnon et al. “Monocular Vision Based SLAM for Mobile Robots”. In: *Pattern Recognition, International Conference on* 3 (Aug. 2006), pp. 1027–1031.

[39] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System.” In: *CoRR* abs/1502.00956 (2015).

[40] Raúl Mur-Artal and Juan D. Tardós. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[41] David Nistér. “An efficient solution to the five-point relative pose problem”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (2004), pp. 756–770.

[42] Edwin Olson, John Leonard, and Seth Teller. “Fast iterative alignment of pose graphs with poor initial estimates”. In: June 2006, pp. 2262–2269.

[43] Yuki Ono et al. “LF-Net: Learning Local Features from Images”. In: *NeurIPS*. 2018.

[44] Edward Rosten and Tom Drummond. “Machine Learning for High-Speed Corner Detection”. In: vol. 3951. July 2006. ISBN: 978-3-540-33832-1.

[45] Ethan Rublee et al. “ORB: an efficient alternative to SIFT or SURF”. In: Nov. 2011, pp. 2564–2571.

[46] Randall Smith and Peter Cheeseman. “On the Representation and Estimation of Spatial Uncertainty”. In: *The International Journal of Robotics Research* 5 (Feb. 1987).

[47] Randall Smith, Matthew Self, and Peter Cheeseman. “Estimating Uncertain Spatial Relationships in Robotics”. In: vol. 1. Jan. 1986, pp. 435–461. ISBN: 9780444703965.

[48] Jrgen Sturm et al. “A benchmark for the evaluation of RGB-D SLAM systems”. In: Oct. 2012, pp. 573–580. ISBN: 978-1-4673-1737-5.

[49] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. “OpenVSLAM: A Versatile Visual SLAM Framework”. In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM ’19. Nice, France: ACM, 2019. ISBN: 978-1-4503-6889-6/19/10.

[50] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. “Visual SLAM algorithms: a survey from 2010 to 2016”. In: *IPSJ Transactions on Computer Vision and Applications* 9 (Dec. 2017).

[51] Keisuke Tateno et al. “CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction”. In: July 2017.

[52] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005. ISBN: 0262201623 9780262201629.

[53] Philip H. S. Torr, Andrew W. Fitzgibbon, and Andrew Zisserman. “The Problem of Degeneracy in Structure and Motion Recovery from Uncalibrated Image Sequences”. In: *International Journal of Computer Vision* 32 (2004), pp. 27–44.

[54] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. “Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing”. In: Oct. 2006, pp. 2276–2282.

[55] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics”. In: *Intelligent Industrial Systems* 1 (Nov. 2015).

[56] Shishun Zhang, Longyu Zheng, and Wenbing Tao. “Survey and Evaluation of RGB-D SLAM”. In: *IEEE Access* PP (Jan. 2021), pp. 1–1.