Fachbereich Informatik

Technische Universität Kaiserslautern

Masterarbeit

Monocular SLAM for Context-Aware Workflow Assistance

Philipp Hasper

Fachbereich Informatik

Technische Universität Kaiserslautern

Masterarbeit

Monocular SLAM for Context-Aware Workflow Assistance

Autor:	Philipp Hasper
Erstgutachter:	Prof. Dr. Didier Stricker
Zweitgutachter:	Nils Petersen
Datum:	30. September 2014

Ich versichere hiermit, dass ich die vorliegende Masterarbeit mit dem Thema
"Monocular SLAM for Context-Aware Workflow Assistance" selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.
Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Kaiserslautern, den 30. September 2014 Philipp Hasper

Danksagungen

Ich danke an dieser Stelle all jenen, die mich während der Anfertigung dieser Masterarbeit motiviert und tatkräftig unterstützt haben.

Ganz besonders gilt dies für Nils Petersen, der als Betreuer meiner Arbeit durch wertvolle Hinweise und seine Expertise einen großen Anteil an diesem Werk hat und für Prof. Dr. Didier Stricker, der mir mit seiner Arbeitsgruppe Augmented Vision vielfältige Möglichkeiten zur Verfolgung meiner beruflichen Interessen bietet.

Auch meiner Familie und Noura Schmuck möchte ich dafür danken, dass sie mich bei meinem Studium unterstützt haben und mir stets mit Rat und Tat zur Seite stehen.

Zusammenfassung

In dieser Arbeit wird die Integration von kontextuellem Handlungswissen in ein SLAM-Trackingsystem vorgestellt. Ein typisches Einsatzgebiet dieses Ansatzes ist die Bereitstellung eines Augmented Reality-Wartungssystems.

Augmented Reality (zu deutsch: Erweiterte Realität) ist eine intuitive Art und Weise, Handlungswissen Schritt für Schritt zu vermitteln — z.B. für Reparatur- oder Wartungsvorgänge. Dies erfordert jedoch komplexe Modelle des Erscheinungsbildes der Umgebung, der durchzuführenden Aktionen und der räumlichen Beziehungen innerhalb der Szene. Um letztere Modelle zu erstellen, wird in der vorliegenden Arbeit SLAM (Simultanes Lokalisieren und Kartographieren) auf Basis von Bildern einer monokularen Kamera eingesetzt.

Zunächst wird ein eigenständiges SLAM-System vorgestellt, welches auf einer 3D-Punktewolke operiert und diese einerseits kontinuierlich mittels eines dreischrittigen Einfügungsprozesses mit neuen Ansichten der Umgebung erweitert und sie andererseits zum Verfolgen (Tracken) der Kameraposition verwendet. Darauf aufbauend wird kontextabhängiges Wissen, welches automatisch aus Referenzvideoaufnahmen extrahiert wurde, in das SLAM-System integriert. Dies ermöglicht es, beim Tracking mögliche Veränderungen der Umgebung nicht nur zu kompensieren, sondern sie aktiv in das Modell miteinzubeziehen. Um diesen Vorgang zu unterstützen und mehrere Trackingmodelle zu einem zusammenzufassen, wird ein neuartiges Verfahren vorgestellt, das im Gegensatz zu ICP (Iterative Closest Point) unabhängig von der räumlichen Struktur der Umgebung ist.

Wir schließen mit der Darstellung typischer Trackingresultate und evaluieren die einzelnen Module unseres Ansatzes, um deren Einfluss auf das Gesamtsystem zu erfassen.

Abstract

In this thesis, we propose the integration of contextual workflow knowledge into a SLAM tracking system for the purpose of procedural assistance using Augmented Reality.

Augmented Reality is an intuitive way for presenting workflow knowledge (e.g. maintenance or repairing) step by step but requires sophisticated models of the scene appearance, the actions to perform and the spatial structure. For the latter one we propose the integration with SLAM (Simultaneous Localization And Mapping) operating on images of a monocular camera.

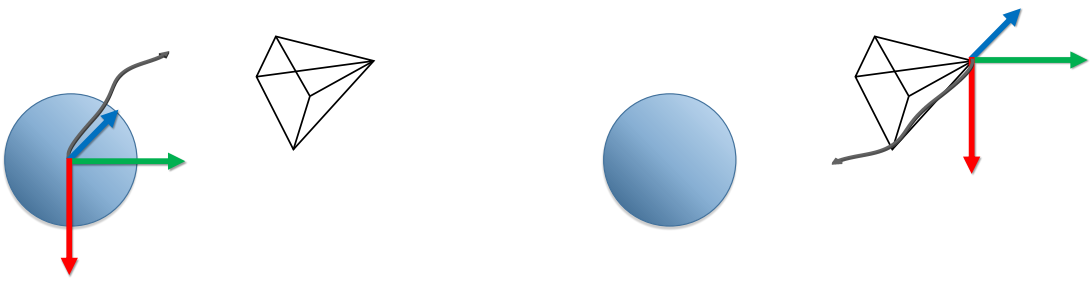
We first develop a stand-alone SLAM system with a point cloud as map representation which is continuously extended and refined by triangulations obtained from new view points using a three-step keyframe insertion procedure. In a second step, we integrate contextual knowledge which is automatically obtained from reference recordings in a so-called offline mapping step. This allows the tracking not only to cope with but actively adapt to changes in the environment by explicitly including them in the tracking model. To aid the data acquisition and to merge multiple tracking models into a single one, we propose a novel method for combining offline acquired maps which is independent of the spatial structure as opposed to ICP (Iterative Closest Point).

We show typical tracking results and evaluate the single components of our system by measuring their performance when exposed to noise.

Contents

1 Introduction

The sense of vision constitutes a major part of human perception and in the past decades, the importance of vision for artificial perception began catching up vastly: Especially for autonomous robots, sensing and processing the environment is vital, e.g. for detecting objects to interact with, determining the spatial relationships and executing movements. Using Computer Vision to aid these tasks has become more and more popular since on one hand it allows for a more detailed information gathering compared to e.g. simple odometers and on the other hand it can serve as additional measurement source for a fusion algorithm [12, 75]. An important problem Computer Vision algorithms have to deal with in such scenarios is the detection of object positions and how they change over time, which is called *tracking*. There are different kinds of approaches summarized under this generic term: object tracking denotes the process of tracking one or several objects relative to the sensor (i.e. in our case the camera) and camera tracking means reconstructing the movement of the camera itself relative to the environment (cf. Figure 1.1).



(a) **(b)**

Figure 1.1: (a) Object tracking as compared to (b) Camera tracking.

While this may seem to be just a swap of reference frames, camera tracking also places different constraints on the scene than object tracking, i.e. the former benefits from a static scene and the latter from a non-moving camera. Camera tracking – or more general: a moving agent tracking itself autonomously – is an important tool for robot navigation where the constant monitoring of the path traveled so far and the next destination is vital for both preventing harm from the robot and accomplishing the given task. While this thesis does not deal with robot navigation, the principal technological foundation of this thesis originates from this area: *Simultaneous Localization and Mapping (SLAM)* [5, 20]. This describes the procedure of a moving agent building a map of its previously unknown environment while simultaneously localizing itself in this newly created map [30]. In the beginning, measurements of odometers and proximity sensors where combined and filtered

(speak: fused) to derive pathway and position but the variety of sensor types quickly grew as inertial measurement units (IMU), GPS, range scanners, and so on were integrated. (e.g. [51, 64]).

This thesis focuses on camera tracking and will examine how to deal with non-static scenes. In contrast to the introductory example of robot navigation, the use case for our approach is a little different: Augmented Reality for workflow assistance.

Augmented Reality (AR) denotes the process of integrating virtual objects into a real environment [4], i.e. placing renderings of virtual objects in a live camera feed so that this superimposition creates the illusion of an object which is not present in reality (cf. Figure 1.2). Augmenting the reality hence means augmenting a 2D camera image which is literally only a projection of the reality.

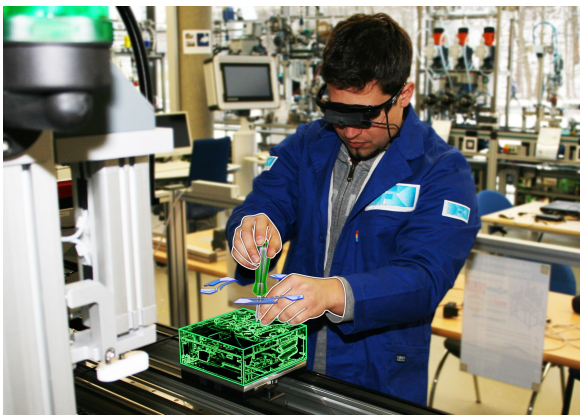
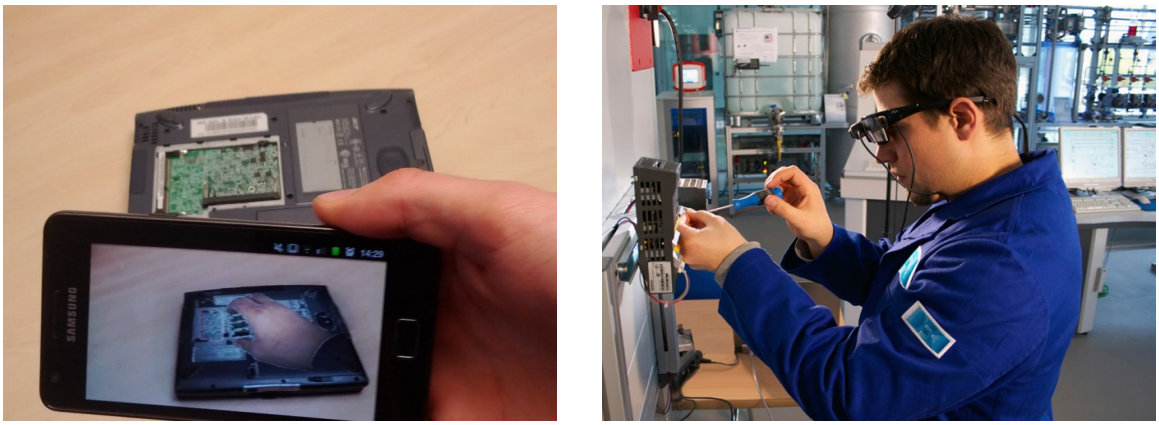


Figure 1.2: Illustration of Augmented Reality

AR is an ideal way for presenting and teaching procedural knowledge to a worker and several studies have shown its effectiveness [74, 49, 33]. By superimposing information about the current workflow like highlighting a screw which needs to be removed, indicating a direction to move in or playing back an animation which shows how to open a lid, workers can be guided through a process previously unknown to them. This approach is called *Augmented Reality manual* and will constitute the use case of this thesis.

When camera and viewing device are closely coupled (as in current smartphones for example), this provides the sensation of a "window" to a world consisting of partly real, partly virtual objects while Augmented Reality googles (a combination of a head-mounted display and a head-mounted camera) provide a somewhat more immersive experience (cf. Figure 1.3). Both ways of presenting Augmented Reality applications strongly rely on the previously discussed tracking to guarantee a realistic looking augmentation: When the object or the camera moves, the virtual objects have to change their positions to adhere the illusion of "sticking to the scene". Since most of todays AR devices are equipped with a simple monocular camera (most notable the smartphones which are far more widespread than any other type of AR device), the used tracking algorithm is often vision-based. Using markers [40, 39], i.e. artificial patterns placed at known locations in the scene, has been



(a) **(b)**

Figure 1.3: Augmented Reality viewing devices: (a) A smartphone as an AR window [31] and (b) Using Augmented Reality glasses for manual tasks.

proven to be a reliable and fast way of pose estimation (and thus tracking). Their major shortcoming is the need to prepare the environment in advance by carefully placing markers all around the scene. A less intrusive technique called *markerless tracking* has become increasingly popular since it allows to distribute an AR application without any further modifications: Electricians do not have to place a handful of markers before they can start repairing a fuse-box and knowledge workers can just pull out their smartphone to see how to clear a paper jam in their office's printer. Plus, there are scenarios where markers are an additional foreign body and constitute a security risk, i.e. if they occlude important control displays. Markerless tracking often relies on the detection and matching of interest points (cf. Explanation in Chapter 2) and derives higher-level structures from them, e.g. planes which allow to establish relationships using homographies and subsequently to derive the pose [69, 58, 53] or non-planar structures obtained by triangulation which allow to compute the pose using 3D–2D matches. The latter method is the approach pursued in this thesis and explained in **Chapter 2**.

1.1 Structure from Motion and SLAM

Structure from Motion (SfM) is a well-established method for reconstructing the environment using a monocular camera. SfM is offline by nature which means that the structure computation is done only after all measurements have been acquired [76, 71]. Early applications of *Online* Structure from Motion (i.e. SLAM) in the field of Computer Vision are [16, 70, 17] and they prepared the ground for autonomous robots and AR applications tracking in a three-dimensional, unknown/unprepared environment with a lower sensibility to drift accumulation compared to SLAM using visual odometry [13, 51]. As addressed earlier, these systems use filter approaches like the Extended Kalman Filter and soon began to integrate more and more sensor types like inertial measurement units (IMUs) [5] and even were combined with object recognition [10].

The PTAM (Parallel Tracking and Mapping) system [37, 38] with its contribution of decoupling mapping and tracking task developed the idea further and is a SLAM system specifically tailored to small AR workspaces. By representing the map as 3D points triangulated from 2D correspondences taken from so-called keyframes and by using an online method for adding new points observed in new keyframes, they get rid of extensive filter techniques which suffer from both an evergrowing state space and the "curse of dimensionality"[15]. Our approach's close relationship with PTAM and our implementation of a keyframe-based SLAM system is explained in **Chapter 3**.

In cases where a more detailed environment reconstruction is needed (e.g. for virtual objects to engage physically correct with their real surroundings), monocular dense [47] or semi-dense [22] approaches have been proposed but our systems sticks with reconstructing a sparse point cloud which are computationally less complex. There are also numerous SLAM systems based on stereoscopic cameras such as [46, 36] but we focus on monocular systems due to their higher availability in AR glasses, smartphones, laptops, etc.

1.2 Augmented Reality for context-aware workflow assistance

Because of its ability to provide live interactive feedback, AR is an intuitive way of presenting procedural knowledge, e.g. how to piece together electronic panels in aircraft manufacturing [11], how to assemble a car's door lock [61] or how to change an engine's combustion chamber [33]. Contradictory to their proven effectiveness and user acceptance [74, 49, 33], Augmented Reality manuals are still not widely available due to AR hardware still being a niche product and a work-intensive content creation process. The first problem is tackled by the continuous spreading of smartphones and smartphone-based AR glasses like Google Glass. But this in turn requires additional efforts to overcome performance and energy limitations like the approach of [31]. The problem of content creation is subject of current research and while there are popular editors like buildAR[1], PLAKAR[2] or the Metaio Creator[3], automatic approaches with little or no user intervention are still rare. One approach using machine learning was proposed by [55, 53] and allows to automatically derive fully operational AR manuals from a single or multiple recordings of a manual workflow. By observing the gains of an image distance function, the recordings can be segmented into meaningful actions and by monitoring the learning rate of a set of nearest neighbor classifiers, those actions can be recognized at runtime. This approach in more detail and how our system adopts the idea of context-awareness is described in **Chapter 4**.

[1] https://www.buildar.com
[2] https://www.plakar.de/
[3] https://creator.metaio.com

1.3 Our contributions

We designed and developed a complete pipeline for tracking a monocular camera with six degrees of freedom in a dynamic environment. The system is able to use prior information obtained from a single or multiple recordings of a manual workflow to provide a good initial estimation of the environment. More notably, dynamic elements in the scene are an explicit part of our tracking model and not simply rejected as noise like in previous dynamic SLAM systems [67, 60, 73].

The contributions of this thesis are

- a terminology for context-aware workflow assistance, developed together with the authors of [55]
- a novel three-step decision process for keyframe insertion
- an $O(1)$ function for ranking the triangulation suitability of two camera views and a view-dependent description of map points
- the derivation of context-dependent SLAM tracking models from reference recordings
- a method of fitting 3D structures based solely on camera trajectories as replacement or initialization of ICP (Iterative Closest Point).

To recapitulate the overall structure of this thesis, explained successively throughout the previous sections: In Chapter 2, we introduce a terminology of concepts required for context-aware SLAM and present a set of tools used throughout the whole thesis, e.g. interest point matching or epipolar geometry. Chapter 3 describes the fundamental tracking and mapping framework and elaborates on map refinement. The adaption of contextual knowledge is explained in Chapter 4. We finish with the evaluation of our system in Chapter 5 and the conclusion in Chapter 6.

2 Foundations

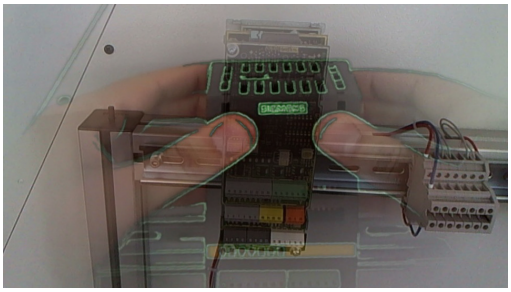
This chapter is divided into two parts. In the first one we elaborate on the proposed system's use case and both develop a terminology and review the underlying mathematical approach. The second part provides explanations of different tools used later in this work and their mathematical background. The principle of the projective camera is taken for granted – the interested reader is referred to Chapter 6 of [29].

2.1 Context-aware AR workflow assistance

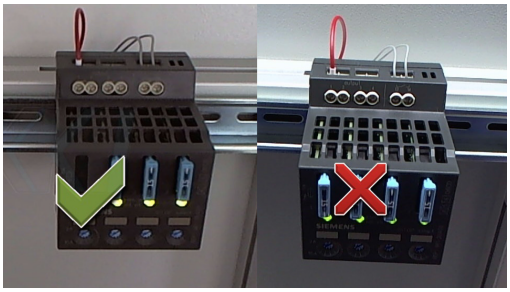
The system at hand is supposed to guide step by step through manual workflows, in most cases (but not necessarily) viewed from first-person perspective. To facilitate this, the approach of [55, 53] contributes two shifts in paradigms:

1. Using one or multiple recordings of the workflow to document, automatically "understand" it in terms of relevant actions and derive meaningful instructions to display. This allows the domain experts themselves (i.e. the electricians, the factory workers, the surgeons, ...) to create AR manuals without knowledge of the underlying mathematics and computer graphics.

2. Monitoring the user's progress and check whether he performed correctly – again based only on the reference recording(s) (cf. Figure 2.1). This increases the efficiency of the worker and reduces the risk of accidents and production failures.



(a) Animated instruction automatically derived from a reference recording



(b) Discrepancy check: visual comparison of target and actual state

Figure 2.1: Context-awareness of the AR handbook. Overlays have been automatically generated and are displayed at the right time. Performed actions are checked for being in accordance with the learned workflow structure. Taken from [53]

2.1.1 Terminology

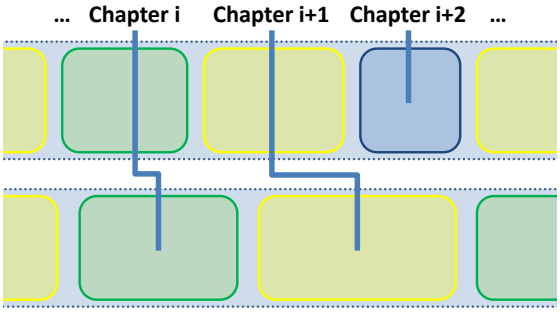
The terminology is developed during the following explanation of how to get a fully functional AR manual from one or multiple recordings by the method of [55]. Figures 2.2 – 2.5 illustrate the used terms starting bottom-up with the lowest level of organization.

A person with expertise in the *workflow* to document puts on a head mounted camera and makes a *recording* showing an exemplary execution from his perspective. Optionally, other experts in the field also record their version of the workflow to enhance the data set with other perspectives or alternatives (i.e. removing one plug before the other and vice versa). Each recording is then loaded into the system and automatically subdivided into *segments* of one of four types: Ignore (negligible footage, e.g. the camera is put on / taken off), *Static* (a relatively steady camera showing one detail in the scenery), *Movement* (the user moves in the scene – preferably between two static segments) and *Action* (the user is interacting with the scene).

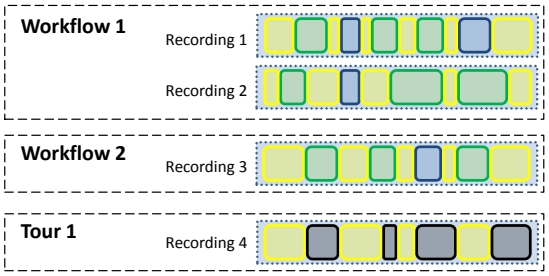
Figure 2.2: Level 0 of our terminology, showing a single recording being subdivided into different kinds of segments. Yellow = static, blue = movement, green = action

So-called *overlays* are then derived from the action segments by using the original footage from the recording and applying visual effects like enhancing edges and using alpha blending. Together with user-defined *annotations* like a manually added arrows, text fields or 3D models these are the possible *augmentations* which can be applied to the image. Static segments are used to learn *target states*: When using the AR manual each action can be verified by comparison of the current state (live camera image) with the following static segment's middle frame, obtained from the recordings (cf. Figure 2.1b).

In order to organize multiple recordings of the same workflow with possibly different segment orders or a varying number of segments, we introduce the concept of *chapters*: A chapter consists of one or multiple segments of the same type but each from a different recording, which comprise the same content. So there may be a chapter "Removing the plug" consisting of three synchronized action segments showing three different people removing the plug. Or there is a chapter "Alternatively switch off electricity" consisting of only one action segment. Of course, chapters do not have to consist of an action, they can also be static or motion chapters. We call any chain of consecutive chapters a *sequence*. Depending on the captured workflow's length and complexity there may be multiple points in time where the user can take an alternative path. This leads to a subdivision of the workflow into a directed acyclic graph of sequences with one starting point (cf. Figure 2.4).



(a) Chapters as interrelations between recordings. Three chapters are shown with chapter i+2 consisting only of one segment from the first recording.



(b) Recordings as parts of workflows or tours. Recordings linked to the same workflow are not obligated to have the same segmentation topology (i.e. the first recording has two motion segments while the second one only has one).

Figure 2.3: Level 1 of our terminology, showing interrelations of recordings

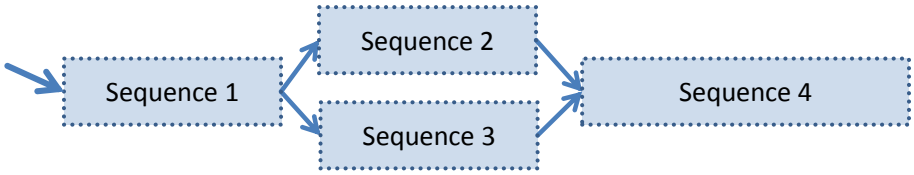


Figure 2.4: Level 2 of our terminology, showing alternatives in a workflow as directed acyclic graph of sequences to model possible variations in the task execution.

Instead of recording a workflow, one can also record a *tour* showing interesting hot spots, e.g. lubrication points in an engine. In this case, only static segments are extracted. If some workflows and tours are semantically linked, i.e. they take place at the same engine, they are subsumed under a *set* and every workflow is assigned its first chapter as entry point. The segments of a tour are all valid entry points (cf. Figure 2.5).

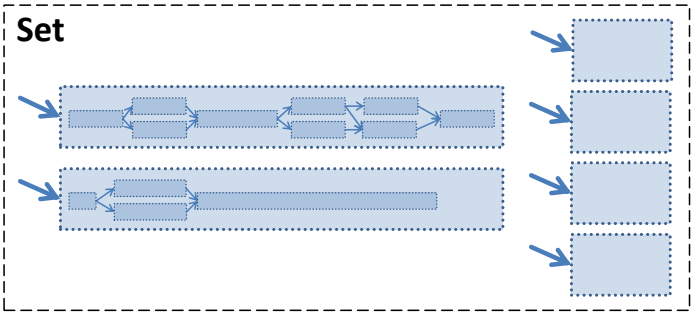


Figure 2.5: Level 3 of our terminology, showing a set consisting of two workflows and four elements of a tour. The arrows indicate possible entry points, i.e. when the user starts the system it detects which of the six alternatives fits to the current camera image.

2.1.2 Mathematical background

The automatic segmentation is achieved by observing the behavior of a robust image distance measurement which is invariant to certain image distortions due to the affine sampling of images (Equation (1) in [55]):

$$d_{\mathcal{T}}(I, J) := \min_{\forall K \in \mathcal{T}(I)} d(K, J) \tag{2.1}$$

$\mathcal{T}(I)$ returns rotated and scaled versions of I, so $d_{\mathcal{T}}(I, J)$ is the distance of J to the best-fitting transformation of I. As a consequence, $d_{\mathcal{T}}(I, J)$ is invariant towards the sampled rotation angles and scaling factors. Rather than operating on this distance function directly, it is encapsulated in a shortest path measurement to cope with repetitive actions (e.g. loosening a screw):

$$D(t_0, t) := \begin{cases} 0 & \text{if } t = t_0 \\ D(t_0, m) + d_{\mathcal{T}}(I_m, I_t) & \text{else if } \exists m \in \mathcal{N}(t_0, t-1) : \\ & \quad \min d_{\mathcal{T}}(I_m, I_t) < T \\ D(t_0, t-1) + d_{\mathcal{T}}(I_{t-1}, I_t) & \text{else} \end{cases} \tag{2.2}$$

where $I_{m/t/t-1}$ is the image at the particular timestamp and $\mathcal{N}$ the set of *novelties*, i.e. the set of frame indices where the pair-wise distance amongst them always exceeds the threshold T. The intuition behind this equation is the following: The distance between two frames t_0 and t is computed by going back in time from t to t_0 searching for a frame m with an image distance $d(I_m, I_t)$ being so small that it is assumed to stem from the same activity. If such an m is found, all frames in between m and t are not part of the shortest path and are therefore omitted. The distance function is then applied recursively on the remaining frames.

The final component of the distance function which makes it truly suitable for recognizing the repetitiveness of actions is a normalization using the sum of all distances and the very smallest distance of any frame m to the frame t with $t_0 < m < t$ (cf. Equation (5) in [55]):

$$\hat{D}(t_0, t) := \begin{cases} 0 & \text{if } D^{sum}(t_0, t) = 0 \\ \dfrac{D(t_0, t) - D^{min}(t_0, t)}{D^{sum}(t_0, t) - D^{min}(t_0, t)} & \text{else} \end{cases} \tag{2.3}$$

This distance function (called normalized shortest path) together with the novelty count is calculated during an one second sliding window and we obtain $\Delta\hat{D}(t_0, t)$ and $\Delta\mathcal{N}(t_0, t)$, respectively. Segments can then be identified by observing $\Delta\mathcal{N}(t_0, t)$: If the value gets saturated, i.e. increases only moderately over a certain period of time, the next segment cut is done with the next increase of $\Delta\hat{D}(t_0, t)$. This also allows to identify segment's type: $\Delta\mathcal{N}$ saturates between two segment boundaries iff this is a static segment. Action and movement segments can be told apart by using homography-based tracking and calculating three scores for translational shift, out-of plane rotation and movement along the optical axis [53].

Finally, a k-Nearest Neighbor (kNN) approach is used for both recognizing the state/action during online use of the manual and to link the segments of multiple recordings together (offline). Each segment i is assigned its own classifier kNN_i which is fed with transformations $\mathcal{T}(I)$ of the own images as positive training samples and those of the previous and upcoming segment as negative ones (cf. Section 4 in [55]). Since the samples are also labeled with their rotation, scale and translation, tracking can be initialized.

2.2 Image features

Extracting data from an image always means to tell unrelated and important things apart. The latter are called *features of an image* and they can be primitive ones like points, edges or circles or more complex ones like motion cues or entire objects. When developing a Computer Vision algorithm or a Machine Learning algorithm in general, the definition of what will be a feature is an important design decision because this determines the parameter space and which methods are applicable[1].

2.2.1 Detection and description of point features

We concentrate on sparse point features, i.e. characteristic points in the image, which are called *interest points* or *keypoints*. One way of defining a keypoint's saliency would be the amount of change which occurs when shifting the image patch around the point in all directions which is the intuition of the Harris corner detector [27]. The FAST detector [62] defines a keypoint as a point which has a continuous arc of pixels in his neighborhood which all have a much higher or much lower pixel intensity than the keypoint itself.

Detecting a keypoint is usually not enough since we also need a method of comparing and recognizing them when given a second image to identify previously seen structures or objects. This is what we use a *descriptor* for: by inspecting a keypoint's local neighborhood, a unique-as-possible description is extracted and stored alongside each detected keypoint. An intuitive choice would be to cut out patches around each keypoint and compare them using the squared sum of pixel-wise distances (SSD), i.e. the fewer neighborhood pixels differ the more similar two keypoints are. A method more robust towards noise and minor image transformations is the BRIEF descriptor [8] which randomly samples point pairs in a keypoint's neighborhood, protocols the result of an intensity comparison and stores them in a binary form which allows for fast descriptor matching using the Hamming distance (as opposed to descriptors using vector representations where matching is computationally more intensive).

Following the recommendations of [52] who examined which keypoint detector is most suitable for a SLAM system, we use **ORB** [63] which is based on the previously discussed

[1] For the sake of completeness: there are also Deep Learning approaches which try to get rid of this design decision by learning which features to operate on [24, 48].

FAST keypoint detector and the BRIEF keypoint descriptor (hence its name: ORB = Oriented fast and Rotated Brief). When designing a keypoint detector, one tries to account for as many image transformations as possible and FAST is invariant to rotation, translation and changes in illumination. This means that the responded keypoint locations should only differ marginally when the image undergoes one of those modifications – a rotated square should still have four keypoints in its corners. Those invariances are even more important for the descriptors and this is a shortcoming of BRIEF since it is not invariant to larger rotations or scale change: While keypoints will be detected at the four corners of the rotated square, their descriptors may differ vastly from the original corners' ones and the matching will fail. ORB solves this problem by using different octaves, i.e. scaled versions of the original image to achieve scale invariance and the determination of a keypoint's orientation to achieve rotation invariance. The invariance towards translation and illumination changes is already given by FAST and BRIEF. By determining scale and orientation of a keypoint, the descriptor can then be computed relative to this affine transformation which means that matching remains a simple bit-wise comparison of two short bit arrays (they have 256 bit in our case).

2.2.2 Feature matching

As discussed previously, keypoints are usually matched by comparing their descriptors. Keypoint matches are also called *correspondences*. To simplify the following explanations, we introduce the distinction between keypoints in **euclidean space** and in **feature space**: A keypoint corresponds to a two-dimensional euclidean point since it has a distinct location in an image. It also corresponds to a higher-dimensional point in the feature space which is represented by the descriptor. While keypoints with a small euclidean distance are not necessarily related to each other, keypoints with a small feature distance are said to be matches. How to exploit the euclidean nature of keypoints is explained in Section 2.2.3, while using the feature space is discussed in the following.

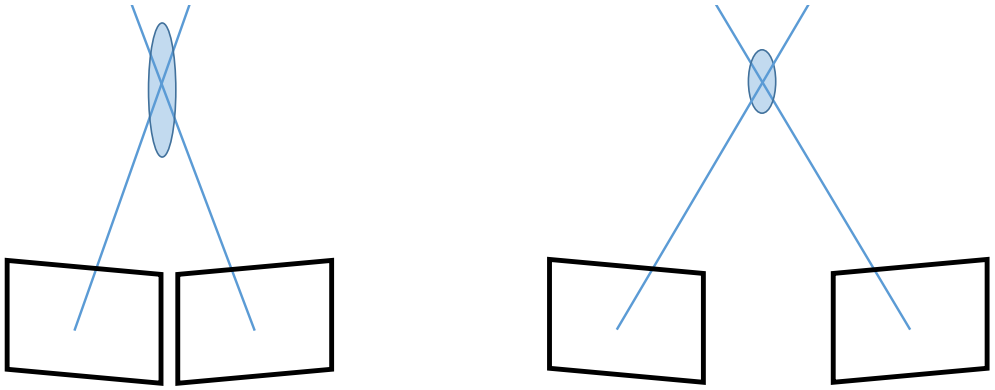
The first choice when deciding which keypoint in one image corresponds to which keypoint in a second image would be to assign each keypoint from the first set the keypoint with the smallest feature distance (i.e. where the descriptors differ the least) in the second set. Since this will assign each keypoint a match, the matching quality has to be examined in order to remove false matches. Either a pre-defined threshold is used to cut of matches with high distances or the ratio test known from SIFT is used [42]:

$$\frac{d(k, k_1)}{d(k, k_2)} < 0.8 \tag{2.4}$$

with k being the keypoint to find a match for and k_1 the best and k_2 the second best candidate. If there is such a k_1 it is a match to k and if not there is no match for k. If (e.g. due to additional matching masks) only one match could be found for k at all, an absolute threshold is used.

As we will see in the next sections, there is a critical anticorrelation between ease of keypoint matching and 3D reconstruction which is called the *small baseline problem*: If two

images differ only slightly in their viewpoint, matching is quite easy since the appearance and thus the descriptors do not differ much. But those obtained matches are nearly useless for the purpose of reconstruction, since little reliable depth information can be obtained from them (cf. Figure 2.6). The next section explains how to tackle this problem by supporting wide-baseline matching with optical flow.



(a) **(b)**

Figure 2.6: Depth uncertainty when using a small baseline (a) compared to a wide baseline (b): The circle indicates the possible location of a triangulated point when accounting for measurement uncertainty.

2.2.3 Sparse optical flow and its application to feature matching

Matching keypoints between two views with unknown camera trajectory can easily be improved in terms of reducing false positives when the optical flow between the two images is available. The optical flow is the two-dimensional measurement of the scene's motion field and due to the loss of information resulting from the projection in 2D it is also referred as *apparent motion* [35].

We use the approach proposed by Lucas and Kanade [43] to calculate the optical flow, implemented in an iterative fashion [7] which quickly converges when the two images' appearances differ only slightly. So in general, the approach performs well only for frame-to-frame tracking.

The use of optical flow for matching is thus limited to a continuous stream of images with piecewise small motion. To use it for matching with a wider baseline we start with a set of keypoints in the first frame, calculate the optical flow frame-to-frame until we reach the desired frame and thus establish feature tracks between the first and second view. Intuitively, a keypoint in the first view and a keypoint in the second view will only match if an optical flow track connects them. In reality, this assumption is way too restrictive since tracks can be lost before they reach the last view (i.e. somewhere on the path the frame-by-frame optical flow could not be computed), the track's end point is most certainly not

the exact position of a keypoint (due to the different mathematical definition of an ORB keypoint and an optical flow point) and optical flow suffers from drift[2].

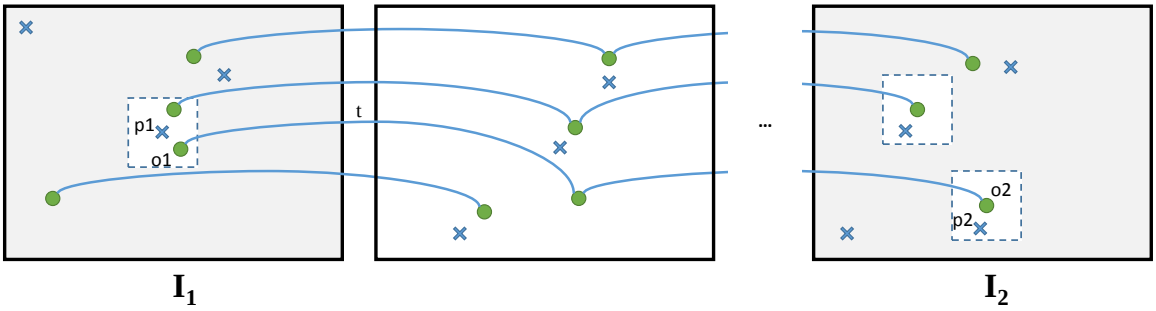


Figure 2.7: Restricting possible matches using optical flow, illustrated for one keypoint $p1$. A window around $p1$ determines which tracks to follow. Each track's endpoint has a window determining which keypoints $p2$ are possible matches with $p1$ (there are two in the illustrated case). To preserve clarity only one optical flow track is annotated with identifiers, i.e. the subscript in $p1_i$ is omitted. Tracks can be lost and keypoints can vanish or new ones appear.

To overcome these problems we weaken the optical flow restriction as follows: Let I_1 be the first view, I_2 the last view and $p1 \in I_1$ and $p2 \in I_2$ the location of a keypoint in these two images (i.e. the norm $\|p1 - p2\|$ resides in the 2D euclidean space, not in the feature space). The optical flow track t is established consecutively from I_1 to I_2 as described above, starting in a point $o1 \in I_1$ and ending in $o2 \in I_2$ (cf. Figure 2.7).

The set of keypoints and the set of start/end points may be disjunct, meaning that the optical flow has been calculated in other points than the keypoints. This is the case when use different keypoint types (i.e. Good Features To Track [65] for optical flow and the scale-invariant ORB [63] for the rest of the pipeline) which happens for example in the offline mapping step explained in Chapter 4.

The application of optical flow for feature matching is summarized as pseudo code in Algorithm 1 — see Figure 2.7 for the used symbols.

2.3 Epipolar geometry

Epipolar geometry is a basic tool for deriving scene structure from two or more image projections. It relates cameras, image points and scene points together and is usually the first concept used in a 3D reconstruction pipeline. The two main mathematical structures in this field are the *fundamental matrix* and its specialization, the *essential matrix*.

[2]The first two problems can be generalized to "there are matching keypoints which do not have an optical flow track connecting them". This subtle rephrasing will come in handy in Chapter 4, where the optical flow computation does not necessarily start in the keypoints to match.

Algorithm 1 Optical-flow-restricted keypoint matching — cf. Figure 2.7 for definitions

```
Start with points in I_1 and compute their optical flow frame-to-frame until I_2.
Remove all optical flow tracks that do not span from I_1 to I_2
Calculate all pair-wise distances ||p1_i − o1_k||                    ▷ euclidean space
Calculate all pair-wise distances ||p2_j − o2_k||
for all keypoints p1 in I_1 do
    Store all o1_k closer to p1 than 8 pixels in its neighborhood N(p1)
    for all tracks t in N(p1) do
        o2 := end point of t
        Store all p2_j closer to o2 than 16 pixels in its neighborhood N(o2)
        Match p1 with every keypoint from N(o2)                      ▷ feature space
    end for
    Keep the best match if it passes the ratio test (cf. Equation 2.4)
end for
```

Definition 1 *The fundamental matrix F is a 3x3 matrix of rank 2 with $x'^{\top} F x = 0$ for two corresponding points x' and x in two different camera views.*

Actually, it can be shown that *any* rank 2 matrix is a fundamental matrix relating two cameras (cf. page 261 of [29]). The intuition for the fundamental matrix is the coplanarity of two camera centers, the two corresponding image points and the 3D scene point (cf. Figure 2.8), a detailed derivation can be found in Chapter 9.2 of [29]. The two most important insights are:

1. F yields **no scale information** because the relative geometry does not change if everything is scaled up or down. This goes even further: Since we are using projective cameras, we are performing a *projective reconstruction* which does reveal neither absolute position, absolute orientation nor scale information (cf. Equation 1.1 in [29]).

2. F can be determined without any reference to the camera poses from a total of **seven correspondences** since it has seven degrees of freedom. This can be derived as follows: The nine matrix elements of F yield eight independent ratios (we only talk about ratios since the scale is ambiguous). Since F also does not have full rank, $det\ F = 0$ holds and one constraint more is released which reduces the nine matrix elements to only seven constraints and thus seven degrees of freedom.

To use the fundamental matrix for reconstruction, we first need the camera's intrinsic parameters which we then can use to derive the essential matrix.

Definition 2 *A camera's linear intrinsic parameters can be expressed by the 3x3 matrix*

$$K := \begin{pmatrix} \alpha_x f & s & x_0 \\ 0 & \alpha_y f & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

with α_x and α_y being the scale factors in x/y direction, f the focal length, s the skew (usually 0) and (x_0, y_0) the camera's principal point.

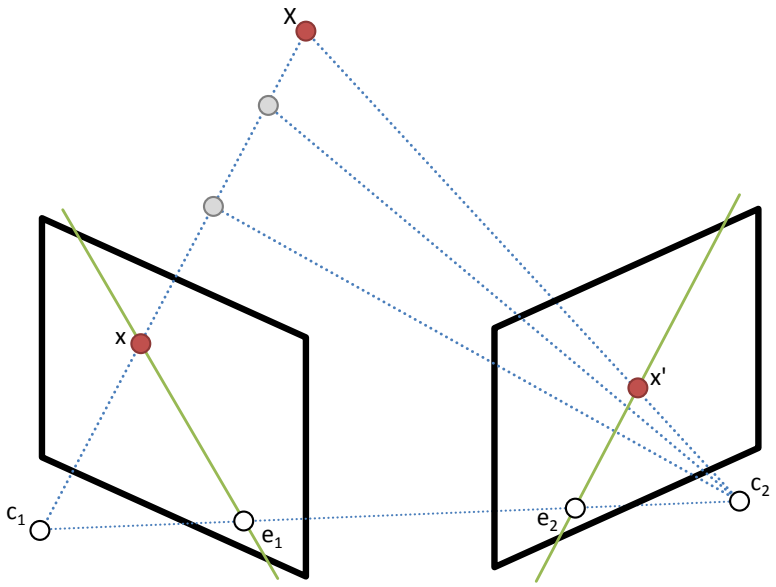
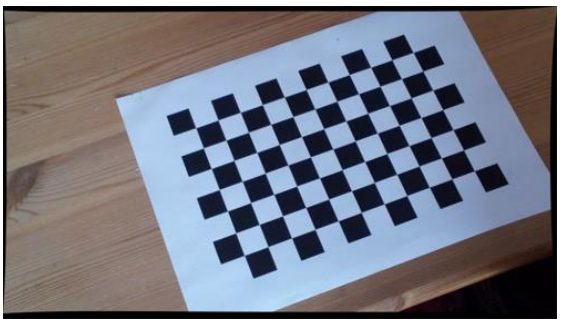


Figure 2.8: Epipolar geometry arising from two camera views with corresponding image points. All pictured points lie on the same plane called epipolar plane. The green solid lines are the epipolar lines connecting image points and epipoles (e_2 is the projection of c_1 into the second view; e_1 analog). Finding a correspondence for x along the epipolar line in the second view is illustrated by the three alternative triangulations. x' is chosen and yields the triangulated 3D point X.

An additional property of a camera cannot be described with a linear transformation and that is the radial and tangential distortion which lead to straight edges not being perfectly straight in the image. Three radial and two tangential distortion parameters are computed in an offline step using several images of a known calibration pattern (cf. Figure 2.9). For more details, see [81]. At runtime, every incoming camera frame is undistorted using this model and thus all modules of our pipeline assume a calibrated, i.e. linear camera model.

(a) The calibration pattern is detected in the reference image



(b) Undistorted version of the same image (not cropped for the purpose of illustration)

Figure 2.9: One of the 12 images of a chessboard pattern used to calculate the radial and tangential distortion of the camera as well as the intrinsics K.

Definition 3 *The essential matrix E is the 3x3 matrix $E := K'^{\top} F K$ with K' and K being the two camera's intrinsic parameters. In our case: $K' = K$ since both views have been taken with the same camera.*

The essential matrix has **five degrees of freedom** and thus can be estimated using the five-point algorithm [50], usually embedded in a RANSAC scheme [25] to be robust towards outliers. All points not satisfying the calculated epipolar constraint are also removed from the whole pipeline — i.e. not considered for triangulation.

2.3.1 Pose estimation

Deriving the relative pose between camera one and two given the essential matrix is only sketched in this paragraph, for the details we refer to Section 9.6.2 in [29]. The first observation is

$$E = R[t]_x \tag{2.5}$$

with $t := (t_x, t_y, t_z)$ being the translation vector[3] and R a 3x3 rotation matrix, both relative to the pose $P_1 := [I|0]$. Using the singular value decomposition of $E = Udiag(1, 1, 0)V^{\top}$ we obtain four possible camera configurations for the second view: $P_2^{1,2} = [UWV^{\top}| \pm u_3]$ and $P_2^{3,4} = [UW^{\top}V^{\top}| \pm u_3]$ with

$$W := \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and u_3 being the third column of U. To determine which of those is the correct solution we perform a *cheirality check*: Only one of the four configuration has the triangulated scene points in front of both cameras (cf. Figure 2.10). In principle, only one point has to be tested [50] but to account for measurement and numerical errors, we triangulate several points and take the configuration having most of them in front of both cameras.

2.3.2 Epipolar constraint for feature matching

Besides the pose reconstruction, epipolar geometry can also be used to speed up and correct keypoint matching. Like the method explained in Section 2.2.3 it also takes place in the euclidean space. As pictured in Figure 2.8, two keypoints can only match if they lie on their corresponding epipolar lines. If the epipolar geometry between two images has been established using initial correspondences, further matching can be improved by disallowing matches with too high distances to the epipolar lines (enforcing correspondences to lie exactly on the epipolar lines will often fail due to measurement inaccuracies).

[3] $[t]_x := \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$

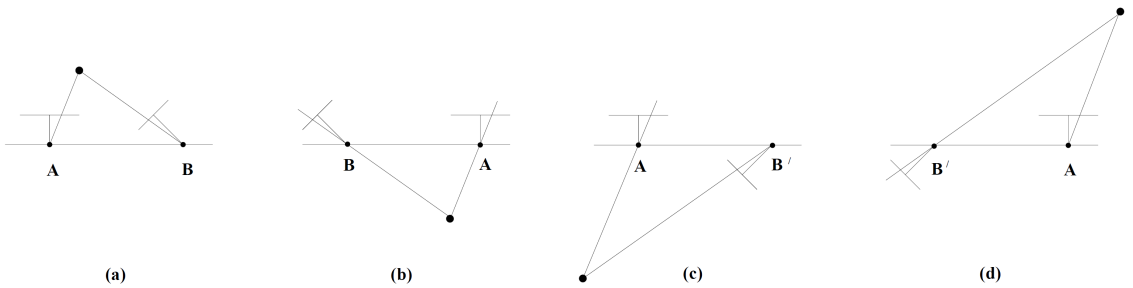


Figure 2.10: The four possible configurations arising from the decomposition of E. Only the first can be true, since it is the only configuration having the triangulated point in front of both cameras. Taken from [29]

If the covariance matrix of the epipolar lines is known, the distance threshold can be replaced by a cone but since this is not the case in our approach, we omit the details and refer to Chapter 11.11 in [29] as well as [72].

2.4 Projective geometry

While we assume the general nature of projective geometry to be known, we want to shortly recapitulate some notations and ideas.

To not confuse projection matrix and camera pose matrix, we state that P denotes the latter one:

$$P = \left[\begin{array}{ccc|c} & R & & \begin{matrix}x\\y\\z\end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.6}$$

The triangulation of a 3D point X by using two points x, x' from two different views (cf. Figure 2.8) is done by deriving the camera poses from the essential matrix and iteratively finding a least-square solution of an overdetermined system of linear equations (four equations per 3D point) [28], an approach which is superior to the simple midpoint triangulation. The keypoints x and x' are also called *measurements* of X.

The last basic tool we discuss in this preliminary is the *Perspective-n-Point (PnP)* solver which answers the question: "Given a set of 3D points and their projections into a 2D image, which pose does the camera have with respect to the 3D scene?". This will be important for our tracking algorithm since our map is represented as a cloud of 3D points, so after matching them with keypoints of the current camera image we need to answer the above question. We use *Efficient PnP (EPnP)* [41] which solves the problem non-iteratively and in $O(n)$ by using an affine combination of in general four non-coplanar virtual points:

The 3D world point p_i can be expressed through the control points c_j by

$$p_i = \sum_{j=1}^{4} \alpha_{ij} c_j, \quad \text{with} \sum_{j=1}^{4} \alpha_{ij} = 1. \tag{2.7}$$

These control points can be chosen arbitrarily (as long as they do not lie on the same plane) but selecting them according to the point cloud's centroid and principal components improves stability. By computing the same control points in the camera coordinate system using $2n$ linear equations with 12 unknowns (3 per control point), a transformation relative to the world coordinate system can be derived.

3 Keyframe-based SLAM

The idea of a keyframe-based SLAM system with a decoupled SfM mapping approach is borrowed from the famous PTAM system (Parallel Tracking And Mapping) [37]. The map to localize in is represented as a cloud of 3D points in euclidean space, each equipped with several descriptors to be recognizable from different viewing angles. This map is used in a parallel fashion to localize in based on a 2D camera image (cf. PnP in Chapter 2). The structure of this chapter is as follows: In the first section (3.1) we will explain the structure of the map. In Section 3.2 we will assume that the 3D map was already created and explain how this information is used to derive the current camera pose. Section 3.3 then elaborates on the mapping step and the chapter finishes with an explanation of map refinement, bundle adjustment in particular (cf. Section 3.4).

3.1 Point clouds as maps for SLAM

Our tracking system's map is represented by a set of 3D points reconstructed from a set of so-called keyframes. *Keyframes* are camera images which have been determined to be characteristic for the scene - e.g. while an image of a fast camera movement would yield few useful information, an image showing substantially new parts of the scene compared to the previous images is desired to be selected as keyframe. Map points are initially triangulated from exactly two keyframes, each contributing one keypoint. Those keypoints are consequently called *measurements* of the map point and since they also yield two descriptors, the map point is equipped with those and the frames' respective poses. This means that a map point has a mapping of descriptors onto camera poses (cf. Figure 3.1).

As already explained in Section 2.2 we do not use image patches for keypoint comparison like PTAM but **ORB** [63] instead which is intrinsic to many image transformations by nature and also capable of realtime[1]. One remark: As a matter of principle, if one can estimate the current camera position, e.g. by knowing the previous one and applying a motion model, one could facilitate keypoint matching by predicting the exact keypoint scale and orientation instead of relying on the heuristics of ORB. However, this is impossible to do for the current camera image's keypoints since due to the different depths of the projected objects the rotation is not uniform along the image [45]. It would be possible to transform the descriptors of all existing map points, though, because here the exact depth is known. But this would contradict the aim of reducing computation time by using easy-to-compare

[1] We use an OpenCL implementation of ORB provided by the OpenCV 3.0 alpha but ORB is also capable of realtime when using a standard CPU implementation.

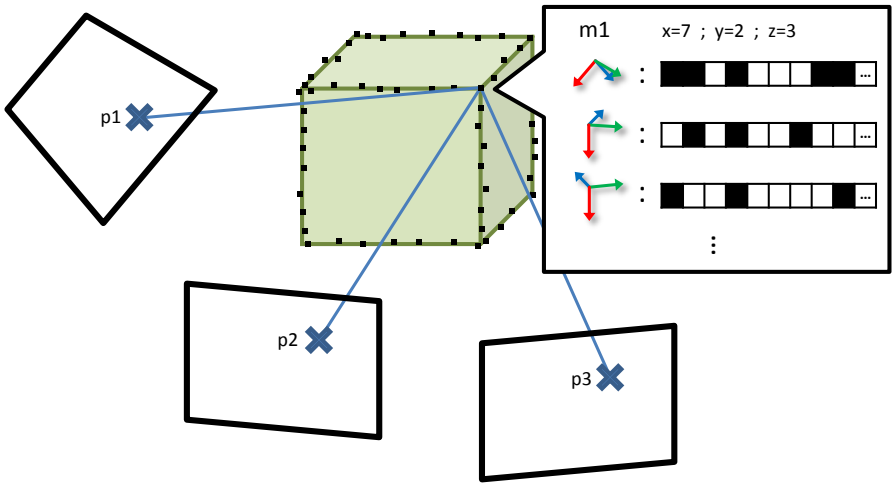


Figure 3.1: Illustration of a point cloud (black dots) and the true object (green box). The map point $m1$ was reconstructed from the keypoints $p1$ and $p2$ and additionally found by reprojection in a third view at the location of the keypoint $p3$. The box on the right shows its descriptor map linking keyframe poses to binary descriptors computed at the respective keypoint positions.

binary descriptors: If for every incoming camera frame all descriptors in the current map would have needed to be recomputed, interactive frame rates would be hard to achieve if not impossible.

While the initial triangulation is always based on two points, n-view reconstruction is implemented by projecting already triangulated points into other keyframes (cf. Section 3.3) and subsequent Bundle Adjustment (cf. Section 3.4). Each keyframe is also stored together with its small blurry image (SBI) and camera pose to aid relocalization which is explained in Section 3.2.1.

3.2 Tracking in a point map

Since tracking and mapping mutually depend on each other, we start the explanation of the tracking procedure by assuming that a point map to localize in already exists and the last camera frame's pose is known as well. The explanation of how both conditions can be met is postponed to the following sections.

The most important technique for our tracking is the PnP solver which allows to infer the camera pose from 3D–2D matches — as already mentioned in Chapter 2, we use EPnP [41] in a RANSAC scheme for this. We obtain those matches with the following pipeline:

1. Retrieve a new image I_t from the camera.

2. Use a motion model to predict the current camera pose P'_t by extrapolating P_{t-1}. In our use case a static motion model is appropriate.

3. Find the five closest keyframes K_i in the map using a pose distance function (cf. Definition 4):

   $$\{K_i \mid i \in [1,5]\} \text{ with } d(P'_t, P_{K_1}) \in \arg\min_i d(P'_t, P_{K_i}) \text{ and } d(P'_t, P_{K_i}) \leq d(P'_t, P_{K_{i+1}}).$$

4. Project only those map points linked to any of those keyframes into the current camera image using the estimated P'_t and discard all of them which are not visible, i.e. whose projections lie outside the image boundaries. This makes sure that even in a large-scale map with many keyframes the search space is reduced to a set of only a few points which we denote with $\{m_i\}$. As a matter of notation, map point projections are also called **reprojections**.

   This general approach of limiting the reprojection effort to the points of only a few keyframes is pursued by other tracking systems like [18] or [60], but our distance function is much easier to implement and faster to compute.

5. Each map point has a map linking poses/keyframes to descriptors. For each m_i, the best-suiting descriptor is selected, i.e. the one stemming from K_i with the smallest i (at least one of the K_i is in the point's descriptor list because of Step 4 but there also may be more than one, so select the closest one). Essentially, this provides a view-point dependent description of each map point and simplifies the matching.

6. Establish matches between $\{m_i\}$ and the set of keypoints in I_t using the descriptors and restrict this process by limiting the search of suitable keypoints to an area around the corresponding reprojection.

7. Take the obtained 3D–2D matches and solve the PnP problem which yields the true pose P_t.

8. Dismiss all previous matches which turned out to be a PnP outlier (their reprojection is more than $4\ px$ away from the assumed keypoint).

Definition 4 *A frame's distance to a keyframe is expressed through the distances of their poses, a measure defined as*

$$d(P_1, P_2) := \alpha \left| P_1^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} - P_2^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right| + \beta \left| P_1^{-1} \begin{pmatrix} 0 \\ 0 \\ q \\ 1 \end{pmatrix} - P_2^{-1} \begin{pmatrix} 0 \\ 0 \\ q \\ 1 \end{pmatrix} \right|$$

with P_1,P_2 being 4x4 pose matrices, $q > 0$ the distance to the so-called point of interest and

$$\alpha + \beta = 1, \quad \alpha > 0,\ \beta > 0.$$

The intuition is that if two keyframes' viewing frustums share a big intersecting volume, they most likely contain many identical map points and thus should be defined to be close to each other. Since this is just a rule of thumb, we don't bother computing the exact intersecting volume and only take a rough guess by computing the combined distance of origins and a point called "Point of Interest" (POI) — the point where we assume most of the scene points reside which is in our use case roughly one arm's length away, i.e. $40\,cm$*. The advantages of this method is that it provides a very fast-to-compute estimate of keyframe interrelation and of (as we will see later) triangulation suitability.*

We have empirically chosen $\alpha = 0.2$ *and* $\beta = 0.8$*. The distance* q *to the point of interest is set to 40 since the world coordinate system is scaled to the units of* $1\,cm$ *(cf. Section 3.3.1) and we assume the area of operations to be an arm's length away.*

The above pipeline is further enhanced by treating PnP inliers of the previous iteration differently than the other map points in Step 6:

6.1 Calculate the sparse optical flow between the images I_{t-1} and I_t linking the location of the previous PnP inlier's reprojections in the first to points in the second image.

a) Search in a very narrow window of only a few pixels around the optical flow results in I_t for a keypoint. If there is one, note it as match to the map point without any comparison of descriptors.

b) If no keypoint was found nearby, perform descriptor-based matching in a wider area around the optical flow result.

c) If the optical flow computation failed, treat this PnP inlier like all the other points and try to find a match according to Step 6.

3.2.1 Relocalization

Whenever using a state-based tracker (like in our case: using the previous pose for reprojection), a problem occurs when tracking is lost. This can happen for example due to camera defocussing, occlusions, fast movement or panning towards unexplored areas[2].

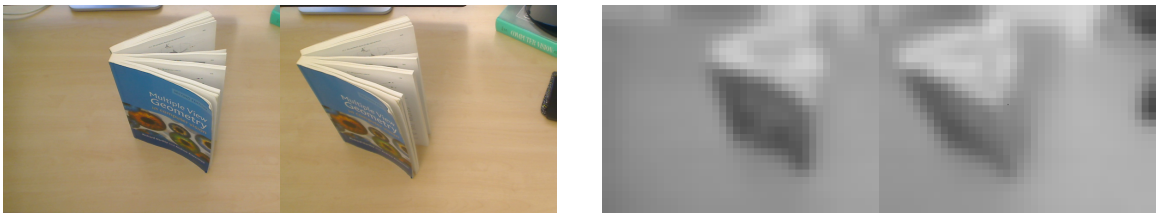
This is why we need a so-called relocalizer which is able to detect known places and set the initial pose guess when the camera points back to the scene. We use small blurry images (SBIs) [38] which derive a descriptor directly from every keyframe's image by

1. Scaling the keyframe's image down to 40×30

[2]While it seems contradictory at first that SLAM could suffer from unexplored areas since after all, this is its primary use case, there are indeed situations where tracking fails because of insufficient map informations. One non-trivial example for this is rotational movement: If the camera turns away from the currently mapped scene in a rotation-only manner, there is zero depth information available rendering impossible the triangulation of new points. Approaches dealing with this problem specifically are e.g. [26, 59] who locate features obtained from rotation at infinity until they can be triangulated properly.

2. Applying a Gaussian blur with $\sigma = 1$
3. Normalizing the intensity by subtracting the image's mean.

Whenever tracking is lost (which is indicated by the number of PnP inliers being below five), the current camera image's SBI is compared to the SBIs of all keyframes by computing the pixel-wise sum of squared distance. The comparison yielding the smallest SSD is taken as match and the tracker's pose guess is set to the corresponding keyframe's pose. Tracking is then attempted as explained before and if it is successful, relocalization is finished. If not, the same procedure is done with the next incoming camera frame.



(a) **(b)**

Figure 3.2: Two pictures of the same scene with a parallax movement (a) and their 40×30 small blurry images (b). For the purpose of printing, the SBI intensity normalization has been omitted — usually the images are much darker.

The first obvious improvement of this approach would be to compute shifted SSDs: Since pixel-wise comparisons are highly sensitive to shifts of only one pixel, one would assume the system to be more robust if it would also try to slide the images. However, we experienced the opposite. Compare the two SBI in Figure 3.2: If the only difference between two SBI is translational, a shifted SSD won't be able to properly differentiate between them and is therefore useless. So in order to not spend processing time for a decrease in differentiability we keep the standard SSD. The effects of small translations are not that grave since the Gaussian convolution already mitigates them and additionally, relocalization does only have to yield a very rough pose guess — the exact alignment is done during tracking.

Relocalization using SBI is used in many visual SLAM systems [38, 59, 26]. Other methods are using for example a Visual Bag of Words [60], vocabulary trees [19] or randomized trees [79].

3.3 Creating a map

This section deals with the process of acquiring environment data and adding it to the map. The first part explains how the system is bootstrapped and the following sections elaborate on map expansion. The section finishes with ways of map refinement. Refining the map is actually not an auxiliary step but an substantial one since deriving 3D structure from only a few 2D measurements is a very error-prone process and relies on a subsequent

non-linear optimization.

3.3.1 Initialization

When starting in a completely unknown environment, there is no 3D information available and thus the point map is completely empty. We initialize the system similar to PTAM [37] by asking the user to perform a translational movement inducing enough parallax to have two view points suitable for an initial triangulation (much like the movement in Figure 3.2). We assume that the user covered a distance of $10\ cm$ and thus can assign the map a meaningful scale. In detail, our approach works as follows:

1. The user is asked to point the camera towards the scene. As he then presses a button, the current image is stored as I_1, Good Features To Track [65] are extracted and frame-to-frame optical flow computation is started as explained in Section 2.2.3.

2. The user performs a movement parallel to the scene, ideally (but not necessarily) with a slight vertical rotation so that the camera's viewing rays intersect each other roughly somewhere around the most salient structures in the scene. Optical flow tracks are established throughout the whole movement.

3. With the second press of a button the user communicates the end of the translational movement which defines the second image I_2.

4. ORB keypoints are extracted from I_1 and I_2 and are matched using the optical flow constraint (cf. Section 2.2.3).

5. Using the ORB correspondences, the epipolar geometry between I_1 and I_2 is exploited and the first set of map points is triangulated (cf. Chapter 2).

6. The scale of the two poses and the map points is normalized so that the distance between the two camera centers is 10 units $\rightarrow$ the map's unit vector has a length of $1\ cm$.

7. Tracking is then started with the pose obtained from Equation 2.5.

Other approaches useful for initialization are disguising the lack of parallax movement until proper triangulation is possible [44] or using the reconstruction approach of [80] who derive depth information from the little shaky movements occurring when a user tries to hold the camera steady. Apart from that, our work on context-aware SLAM in Chapter 4 can also provide an initial map reconstructed in an offline mapping stage.

3.3.2 Identifying distinctive keyframes

In principle, every camera image can become a keyframe after one tracking iteration if its pose is known with adequate precision, i.e. tracking is performing well. The tracking

result is evaluated by counting the PnP inliers: If there are more than 20 or more than half of all visible map points are inliers, tracking is considered to be good. If there are less than five inliers, tracking is considered to be lost.

But the quality of the keyframe's pose cannot be the only constraint, since this would lead to a lot of subsequent frames being added and in consequence to map degeneration (cf. the small baseline problem, Figure 2.6). The first simple restriction is that at least 20 frames have to elapse between two keyframes. On top of that there are three tests: A camera image is added as a new keyframe iff $A \wedge (B \vee C)$. The tests A to C are now explained one after another.

A: Pose test

The current camera pose is compared to the pose of each already existing keyframe to evaluate whether the new keyframe candidate would lead to a small baseline problem. As we have proposed a scale normalization for our map, we now deal with a scale-dependent approach. A scale-independent approach for pose evaluation is discussed in Chapter 4.

The foundation of this technique is the distance function $d(P_1, P_2)$ proposed in Definition 4 together with a modification $d_t(P_1, P_2)$ where $\alpha = 1$ and $\beta = 0$, i.e a measure of translation only. Both measures have to exceed a certain threshold for a camera image with the pose P to be a valid keyframe candidate

$$\forall K_i : d(P, P_{K_i}) > T \wedge d_t(P, P_{K_i}) > T_t \tag{3.1}$$

with $T = 17$ and $T_t = 10$ in our configuration. This makes sure that the camera pose is a substantially new one and the frame will add a previously unseen view point.

B: Histogram of PnP inliers

A camera pose providing a new view of the scene does not guarantee that new information can be extracted from the image. An additional test which also avoids clustering of many map points in small areas is the histogram test, a modified version of the one found in [60]:

The camera image is subdivided into 70×70 tiles and each one is examined for keypoints and PnP inliers: It is counted as *detected bin* if it contains at least one keypoint and as *matched bin* if it either contains the reprojection of at least one PnP inlier or more than five keypoints which have been matched to map points but are not PnP inliers. While a detected bin is counted "pro adding", a matched bin is "contra adding", since the latter indicates that no new information is available: either a salient map point is already available or several map points seem suitable but fail to contribute to the pose computation which is an indicator for matching-uneligible image content. A potential keyframe passes the histogram test iff

$$\frac{|\text{matched bins}|}{|\text{detected bins}|} < 0.5 \tag{3.2}$$

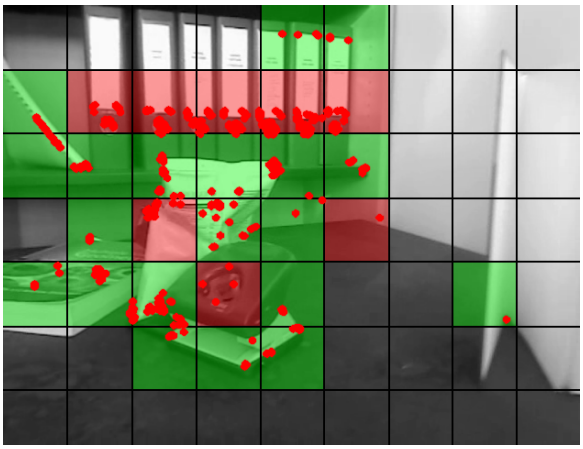


Figure 3.3: Example PnP histogram during keyframe insertion. Green bins contain keypoints but no PnP inliers and red bins contain at least one PnP inlier. This frame passes the histogram test with a ratio of 0.28 and is thus added as a new keyframe.

C: Appearance change detection

While the histogram test avoids adding keyframes showing only already known structures to the map, it conflicts with our robust optical flow tracking: As explained in Section 3.2, the tracking follows PnP inliers by computing the optical flow frame-to-frame. This means that the tracking can cope with gradual appearance changes but once an optical flow track is lost, tracking can not recover the point since the new appearance is not stored in the map. Thus, always preventing the addition of a new keyframe when PnP inliers cover the entire image is too prohibitive. So if the histogram test B fails, every PnP inlier is tested for an appearance change:

$$App := \{p \in pnpInliers \mid \forall \text{ stored descriptor } i \text{ of } p : d(i, c_p) > 1.5 \cdot T\} \tag{3.3}$$

with c_p being the descriptor of the keypoint currently close to the map point p and T an absolute matching threshold for a good match (in our case — ORB with a 256-bit hamming distance — the value is $T = 40$). A potential keyframe passes the appearance change test iff

$$|App| > 20 \vee \frac{|App|}{|pnpInliers|} > 0.5 \tag{3.4}$$

3.3.3 Selecting keyframes for triangulation

If the current camera frame passes these tests, the tracker calls the mapping thread for keyframe insertion. The mapping thread then performs the insertion in the background while the tracking continues. A list is compiled containing the five already existing keyframes which are most suitable to triangulate new points with. This list is also known

from the previously explained tracking and consists of the five closest keyframes in terms of the distance function $d(P, P_{K_i})$. Again, the intuition is that keyframes whose viewing frustums' intersecting volume is large, contain the same parts of the scene. All of those selected keyframes have a sufficient minimum distance due to the thresholding performed in the pose test.

3.3.4 From keypoints to map points

Given a map created from several keyframes and an image with a correctly identified camera pose designated to become a new keyframe, there are three possibilities how to enrich the map:

1. **PnP inlier:** Being an inlier during the PnP solving means that a correspondence between a 3D map point and a 2D keypoint has been established and their distance in the image domain is below a certain threshold. We thus conclude that the 2D keypoint is a measurement of the map point from a previously unseen perspective. It is then added to the map point's set of measurements and attributed with the pose it was observed from. Please note that while a PnP inlier may contribute to a keyframe rejection like explained in the previous section, it is always attributed an additional descriptor if the keyframe was accepted.

2. **Matching inlier:** There are many reasons for a matched map point not being a PnP inlier - the most obvious ones being that it is currently not visible, it is occluded or it vanished from the scene. Another reason is that it is indeed visible but was disregarded during pose computation because of its inaccurately reconstructed position. To account for this and to enable the map point to be refined during bundle adjustment, we try to match the point again. Based on the most recent camera pose estimate, we project the map point back into the image and then restrict the possible matches to those keypoints not farther away from the reprojection than a certain threshold. Matching a matching inlier again has the advantage that now a more accurate constraint can be derived from the computed pose than in the previous matching attempt (where the pose was not yet known but only estimated).

3. **New points:** This is the most important step since it actually allows for map expansion (and not only refinement of previously seen structures). One after another, the keyframes selected as triangulation partners and the current camera frame are examined for entirely new map points as follows:

    1. Establish an epipolar constraint $E_{rel} = R_{rel} \cdot [t_{rel}]_x = R_1^{-1} R \cdot [t_1^{-1} \cdot t_2]_x$

    2. Mask out regions around PnP inliers to avoid clusters of map points.

    3. Establish matches subject to the epipolar constraint and the PnP mask, i.e. only allow matches which are close to their respective epipolar lines and who do not lie next to a PnP inlier.

It is important to note that it is not possible to place a projective constraint on this matching process, since the point depth is not yet known and thus only the weaker constraint arising from the epipolar geometry can be used to reduce false positives.

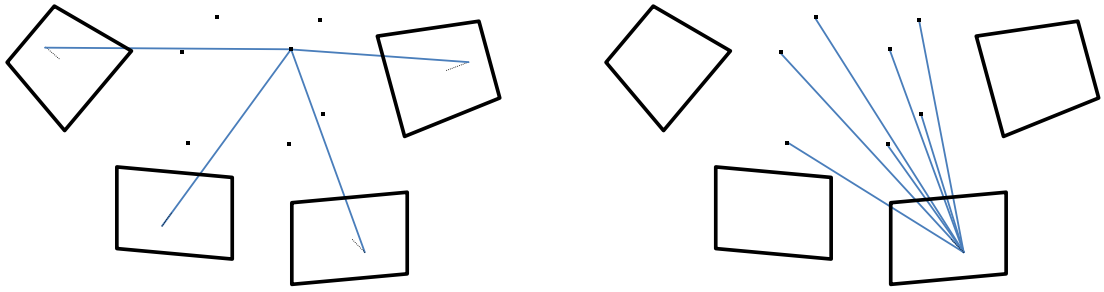
3.4 Refining the map

A prominent measure in the field of projective reconstruction is the *reprojection error*: The quality of a PnP solution is defined through the number of 3D map points whose reprojections are close to the matching keypoints and when triangulating new points, triangulations with a high reprojection error

$$\max\left\{\|x - [K \mid I]P_1 \cdot X\|, \|x' - [K \mid I]P_2 \cdot X\|\right\} > T \tag{3.5}$$

are discarded (x, x' are the two point locations and X is the 3D point. All three are in homogeneous coordinates).

Another application of the reprojection error is in the n-view geometry established by *bundle adjustment*. Up until now a 3D point was always reconstructed from exactly two views and then rediscovered in additional views while the information obtained from these discoveries did not have any effect on the reconstructed 3D coordinates. This is what we introduce bundle adjustment (BA) for: BA not only refines the points' 3D positions but also the camera poses, simultaneously. Its name comes from the fact that two types of bundles are adjusted [29, 78] as illustrated in Figure 3.4: (a) the bundle of rays between each 3D point and the centers of the cameras it was observed from and (b) the bundle of rays between each camera center and the 3D points visible in this image.



(a) Bundle for one map point connecting it to all camera centers

(b) Bundle for one camera center connecting it to all map points

Figure 3.4: The two types of bundles subject to optimization during bundle adjustment

In general, the only fixed parameters are the measurements taken in the images and the cost function to minimize is that of the reprojection errors as a function of the poses and map points:

$$\min_{\{P_i\},\{X_j\}} \sum_{ij} \|x_{i,j} - [K \mid I]P_i \cdot X_j\|^2 =: \min_{\{P_i\},\{X_j\}} \sum_{ij} \|f(P_i, X_j)\|^2 \tag{3.6}$$

with P_i being the different views parameterized by R_i and t_i, $x_{i,j}$ the measurements therein and X_j the map points. The function $f(\cdot)$ is also called *cost function*. An important modification we have to apply on this familiar non-linear least squares problem is that of a *loss function* $\rho(\cdot)$ which limits the influences of matching outliers:

$$\min_{\{P_i\},\{X_j\}} \sum_{ij} \rho(\|f(P_i, X_j)\|^2). \tag{3.7}$$

We use the Huber loss function for this (cf. Figure 3.5) which is piecewise defined to be quadratic for small values and linear for large ones:

$$\rho_\delta(a^2) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta \cdot \left(|a| - \frac{\delta}{2}\right) & \text{else.} \end{cases} \tag{3.8}$$

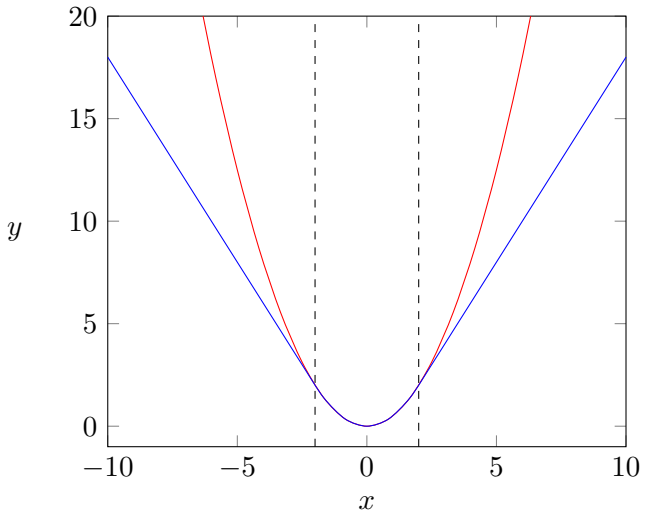


Figure 3.5: Huber loss $\rho_2(x)$ (lower blue curve) as compared to a square loss (upper red curve). The dashed lines mark the transition points from quadratic to linear behavior in $\rho_2(x)$.

To find the minimum in Equation 3.7, we use a sparse Levenberg–Marquardt algorithm (cf. Appendix A6.6 in [29] and [1]). The most important feature of this approach is that in each iteration the original non-linear problem is approximated linearly using the Jacobian matrix $J(x)$ and a step size Δx:

$$f(x + \Delta x) \approx f(x) + J(x)\Delta x. \tag{3.9}$$

The resulting linear least square problem can be solved using standard methods like QR factorization. However, in larger-scale problems like bundle adjustment a more efficient scheme should be deployed and indeed, BA exhibits special characteristics which allow the application of the Schur complement trick [2]:

Definition 5 *The **Schur complement trick** is based on the observation that no term $f(P_i, X_j)$ includes more than one camera parameter block. As a consequence, the computation of the regularized Hessian matrix H used for solving the LS-problem can be reformulated such that it does no longer depend on the points and cameras, but on the cameras alone: Instead of solving the so-called normal equations*

$$H \begin{bmatrix} \Delta P \\ \Delta X \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}, \text{ with } H = \begin{bmatrix} B & E \\ E^\top & C \end{bmatrix},$$

we solve

$$\underbrace{\left[B - EC^{-1}E^\top\right]}_{\text{Schur complement } S} \Delta P = v - EC^{-1}w$$

using sparse cholesky factorization. Because S only depends on the cameras, it is also called "reduced camera matrix".

A detailed derivation of the mentioned linearization and the Schur complement can be found in [2].

On one hand, BA is quite tolerant towards missing data (i.e. points that should be visible in one view but could not be measured correctly) but on the other hand, due to its non-linear nature, it is quite dependent on a good initial configuration. This is why we pose additional constraints on the minimization of the reprojection error:

1. **Fasten the first camera:** As explained in Chapter 2, projective reconstruction is ambiguous towards projective transformations. This is why the first camera pose was arbitrarily defined to be at the origin: $P_1 = [I|0]$.

   This fixation is also kept when performing bundle adjustment. But we found out that it is beneficial to also modify the first camera during BA and only afterwards transform the whole coordinate system so that $P_1 = [I|0]$ again. This is due to the pronounced dependency upon a good initial configuration: If the only error in the reconstruction would be that the first camera is slightly less rotated, fastening it would mean that every other camera and map point would have to change. Since this is more likely to fail, the pose normalization is done afterwards.

2. **Fasten the first $n - 10$ cameras, if more than 10 are available:** Despite the heavy optimization of BA (reduction of computational complexity as discussed above and using a concurrent implementation), it suffers from an ever-increasing parameter space — each newly acquired keyframe and map point increases the runtime. In order to remain realtime-capable we limit BA to the last 10 keyframes and map points therein:

1. Take the at most 10 most recent keyframes $\{K_i\}$ and collect all map points $\{m_j\}$ which have been measured in at least one of them.

2. Start bundle adjustment with the parameters $\{P_i\}$ and $\{m_j\}$ (i.e. those are the only parameters subject to optimization).

3. During BA, take also all of the fastened keyframes in which at least one of the map points $\{m_j\}$ was measured and include these reprojection errors in the optimization. This means that the remaining keyframes are not allowed to change their pose but still influence the minimization.

As [23] has shown, this recurrent application of bundle adjustment even on only a small subset of the map decreases the tracking failure rate significantly. Please note that contrary to the fixing of the first camera described above, we now actually keep the cameras fix. This is because fixing retroactively by transforming all other map entities after optimization does not work with multiple cameras (the interrelations in between the set of fixed cameras may have changed) and we are safe to assume that the initial camera poses are now estimated with a sufficient quality.

One can also optimize the camera's intrinsic parameters by evaluating $f(K, P_i, X_j)$ but since we have them calibrated, we keep them fixed to decrease the parameter space and to prevent disimprovement of actually correct parameters.

4 Context-aware SLAM

As clarified in the previous chapters, quality of mapping (and hence quality of tracking) highly depends on the quality of interrelations between the different angles of view obtained from the same scene. However, due to the use of a single monocular camera, these different views are necessarily taken at different points in time. This places another constraint on the scene: Rigidity — when the scene changes while the camera is moving, it is hard to differentiate which appearance changes originate from the change in perspective and which from the altered scene. Several SLAM approaches tackle this problem [6, 67, 73] but they all differ from our system described here as they dismiss scene dynamics simply as noise while it is an explicit part of our model.

We use the knowledge about scene dynamics gained from the workflow analysis (cf. Section 2.1) as follows: Like the planar tracker of [53], we establish a tracking reference frame for each chapter during the offline segmentation. During online use the current state (i.e. the chapter the user is in) is determined using kNN classifiers [55] and the associated tracking model is loaded — in the case of the planar tracker this is a so-called relevance plane, in our case it is a 3D point cloud. Those tracking models have to be acquired beforehand and this is why we perform Structure from Motion (SfM) on the available workflow recordings to reconstruct the 3D structure.

It is important to note that common, straight-forward SfM approaches have some problems with reconstructing a scene from the given workflow recordings. As [78] pointed out, not all recordings are suitable for reconstruction problems and a session should usually be planned in advance with the goal of appropriate camera placements in mind for the bundle adjustment to converge. As a matter of fact, this is also true for keyframe-based SLAM — the authors of PTAM recommend a "sinusoidal or zig-zag camera motion" [38] when exploring the scene. In our case, especially the lack of redundancy is a major problem since the recordings have been made with a different intention: the documentation of a step-wise workflow. Placing another constraint on the recording setting would vastly impair the user experience and is contrary to the idea of generating recordings as a daily work's byproduct. Additionally, camera placements that are beneficial for reconstruction may impair the automated workflow analysis.

We try to overcome some of these problems by using information gained from the workflow analysis, the segmentation in particular. This chapter explains how to derive information about the scene's 3D structure from the different segment types and finally, how to stitch those maps together. This stage is called *offline mapping* since pre-recorded sequences of a workflow are first divided into chapters, multiple 3D point maps are created as explained below and then used as initial map which is then extended following the

SLAM scheme from Chapter 3. Each map is created by an initial triangulation taking two views with a sufficiently large baseline as explained in Section 3.3.1. The only difference is that now on the one hand this has to be done without any user interaction but on the other hand, we are able to incorporate the entire sequence since we have the full recordings available. The next sections explain how to derive the two initial frames for map initialization depending on the segment's type. After this we elaborate on how to expand these initial maps and finally, how to merge them to establish interrelations between them.

4.1 Static chapters

The static chapter analysis is done for all static chapters having more than one recording available, thus consisting of multiple segments. Since each of those segments have been classified as static, finding two frames with sufficient baseline in one single segment alone is rather rare. As a consequence, we try to find such a pair in between two frames of different segments with the premise that their view points may differ sufficiently due to the recording persons having different body heights (vertical parallax) or looking directions (horizontal parallax). To find the best pair of images for triangulation, all recordings are

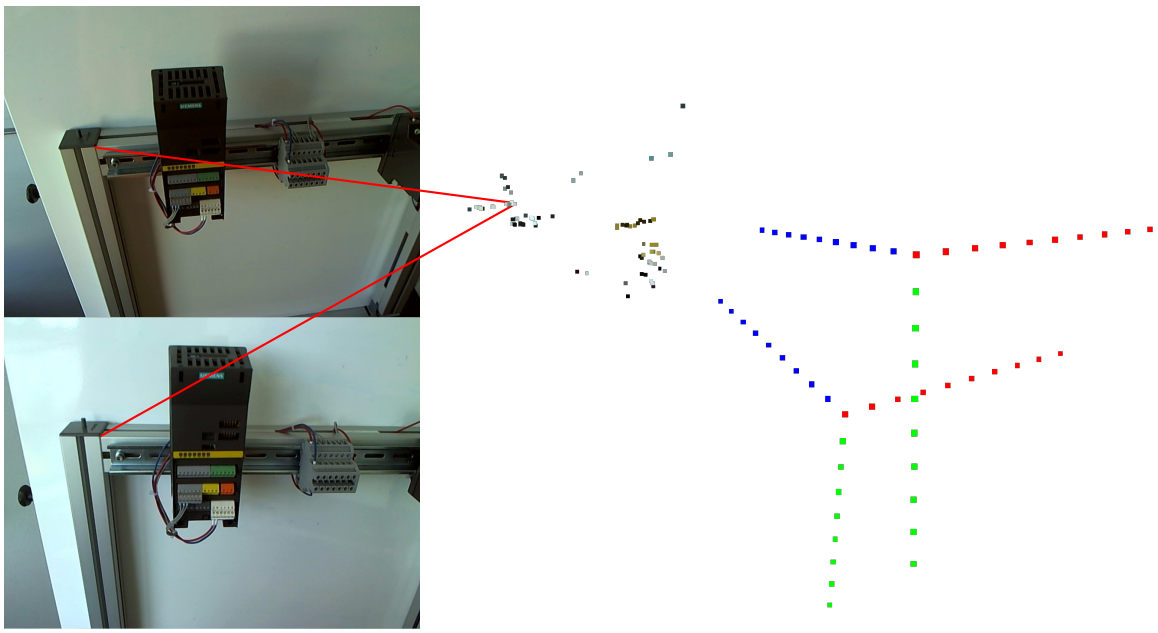


Figure 4.1: Two frames from different recordings but of the same chapter and the reconstructed point map with the camera poses. The red lines are for guidance only and exemplarily link a reconstructed point to its two keypoints

sampled by taking every fifth frame and then evaluate the pair-wise computed essential matrix in terms of the inliers satisfying the epipolar constraint: Every pair having more than 15 epipolar inliers is added to a candidate list which is then processed in descending order of the inlier count until a configuration produced a map with at least 20 map points:

1. Find the image pair with the highest number of epipolar inliers in the candidate list
2. Using this pair, find additional keypoint matches using the epipolar constraint
3. Compute the pose and triangulate map points
4. If more than 20 points have been successfully triangulated, abort and add the triangulated points together with the two frames as keyframes to the map. If not, remove the image pair from the candidate list and start with step 1 again.

The most expensive part of this procedure — the computation of the pair-wise epipolar geometry — is implemented concurrently since the estimations are independent of one another.

4.2 Movement chapters

In theory, the content of movement chapters is most suitable for SfM since the camera is traveling through the scene, so we can operate on an individual segment of the chapter. However, initializing the map has to be done without any user interaction which means that the two frames which have been selected by the user for initialization in Section 3.3.1 have to be identified automatically. Additionally, movement segments may not have distinct start and stop frames picturing the same scene — e.g. if the recording user traveled down a hallway, start and end frame may not have any image features in common. As a consequence, just taking the first and the last frame of a segment may yield a wide baseline but the image content may be totally unrelated.

The advantage that we have in this offline mapping step compared to the user-driven map initialization is that we have the ability to jump back and forth in time, i.e. initialization does not have to take place at the chapter's beginning but can also be done somewhere in the middle and then be carried back to the starting frame (cf. the region growing in Section 4.4). Hence, our goal is to identify two frames in a motion segment which

- establish a sufficiently wide baseline,
- share a substantial part of the scene and
- contain map points of different depths.

The first two requirements mutually affect each other while the third one is a weak but desirable constraint as it simplifies the epipolar geometry estimation (having only coplanar points yields only three degrees of freedom and thus the resulting fundamental matrix is degenerated — see Section 11.9.2 in [29]).

The decision process is once again based on a scoring function, this time based on the *apparent motion*, i.e. the optical flow (OF): The frame-to-frame OF is a fast-to-compute first estimation of camera movement and scene structure, e.g. if two frames do not have any

visual structures in common, there will hardly be any OF tracks connecting them and if the camera does not move much, the tracks' length will be rather short. The OF tracks are established similarly to the method explained in Section 2.2.3 but with the modification that new tracks are started throughout the segment:

1. Start with the first frame of a movement segment and extract 500 Good Features To Track [65].
2. Compute the optical flow frame-to-frame starting in this features to obtain OF tracks. Whenever the optical flow could not be computed for a feature, the corresponding track is terminated.
3. Whenever the number of currently active tracks falls below 200, at most 300 new tracks are obtained. In order to assure a sufficient distribution of tracks in the whole image, an image mask is used which prevents the detection of new keypoints closer than 10 px to an already existing track.

This procedure allows to obtain a set of optical flow tracks for every pair of images $(I_s, I_t)_{s<t}$ in a segment:

$$tracks(I_s, I_t) := \{t_i \in ofTracks \mid startFrame(t_i) \leq s \wedge endFrame(t_i) \geq t\} \tag{4.1}$$

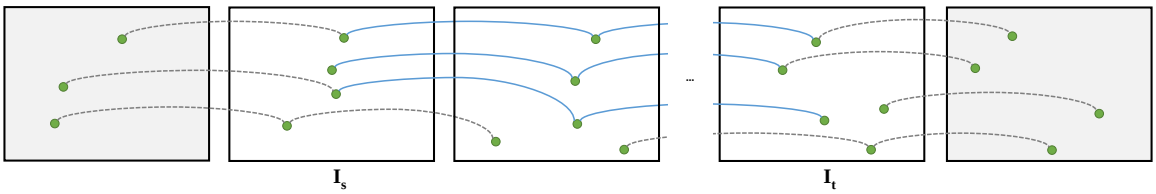


Figure 4.2: Illustration of optical flow tracks being established between the two frames I_s and I_t. Only the blue solid lines are part of the resulting set $tracks(I_s, I_t)$.

Notice that tracks that terminate before I_t as well as tracks starting after I_s are not part of this set (cf. Figure 4.2). While the optical flow computation itself must not be sampled, the evaluation of image pairs is sampled with a step size of five like the static chapters and a minimal distance of five frames: $(I_s, I_t) \quad \text{with} \quad s < t \wedge s \bmod 5 = 0 \wedge t \bmod 5 = 0$.

Each image pair is ranked according to its calculated variation:

$$var(I_s, I_t) := (\mu_{dist} + \sigma_{dist} + \sigma_{starts}) \cdot \frac{N}{500} \tag{4.2}$$

where $N := |tracks(I_s, I_t)|$ is the number of optical flow tracks,

$$\mu_{dist} := \frac{1}{N} \sum_{t \,\in\, tracks(I_s, I_t)} \|start(t) - end(t)\|$$

the mean traveled distance of all tracks and σ_{dist} the corresponding standard deviation. These values give a rough estimate of the baseline and scene disparity. A rough measure of feature coverage is σ_{starts}, which is simply the variance in the starting points' positions.

The image pairs are sorted in descending order according to the measure $var(I_s, I_t)$ and as for the static chapters, we go trough this list and try to reconstruct 3D points:

1. Find the image pair with the highest $var(I_s, I_t)$ in the candidate list
2. Establish keypoint matches using the optical flow tracks (cf. Section 2.2.3)
3. Compute the epipolar geometry, derive the pose and triangulate map points
4. If more than 20 points have been successfully triangulated, abort and add the triangulated points together with the two frames as keyframes to the map. If not, remove the image pair from the candidate list and start with step 1 again.

4.3 Action chapters

Action chapters are the chapters with the most image dynamics unsuitable for reconstruction: In most cases, the user fixates the scene and only performs rotational head movements while his hands occlude most of the scene and the 3D scene structure is likely to change. This is why we do not attempt to initialize a map directly from an action chapter, but instead try to carry forward the maps of the two surrounding static chapters. This is achieved by a region growing approach which will be explained in the next section.

One advantage we can draw from action chapters is that the hand posture is available [54, 56, 57] which we can use to mask out areas were keypoints should be neither acquired nor matched.

4.4 Region Growing

The last sections described how to derive an initial triangulation from a chapter. For static chapters, this is enough in most cases but for movement and action chapters we would like to take the other available frames into account as well. We employ a region growing approach, meaning that we start from the second frame I_t used for triangulation and run the SLAM algorithm on all subsequent frames till we reach the chapter's end. Analogously, we start with the first frame I_s and traverse the frames backwards until we reach the chapter's first frame. Eventually, this will yield additional keyframes and — after performing bundle adjustment — refine the reconstructed map.

Unfortunately, there is no scale information available as in the approach explained in Chapter 3: During the offline mapping step, there is no camera movement available for which we can assume a translation of a fixed distance. This means that our pose distance function (Definition 4) is only able to provide a keyframe ranking for triangulation suitability, but is not useful for rejection of unsuitable poses as in Section 3.3.2. One method would be to manually define the scale or using a database of objects with known dimen-

sions but we employ an automatic, generic approach for identifying distinctive keyframes based on *apical angles* [77].

The apical angle α_X in the 3D point X is the angle between the two rays connecting it with the camera centers:

$$\alpha_X = cos^{-1}\left(\frac{\overrightarrow{Xc_1} \cdot \overrightarrow{Xc_2}}{\|\overrightarrow{Xc_1}\|\|\overrightarrow{Xc_2}\|}\right) \cdot \frac{180}{\pi}. \tag{4.3}$$

A high apical angle means that the camera baseline is high enough for this specific 3D point to be reconstructed with an appropriate precision. Please note that it would also be possible to measure the angle of the cameras' viewing rays. But this yields many false negatives, i.e. rejections of perfectly fine camera poses: A purely translational movement may yield enough disparity despite the angle between the viewing rays being zero.

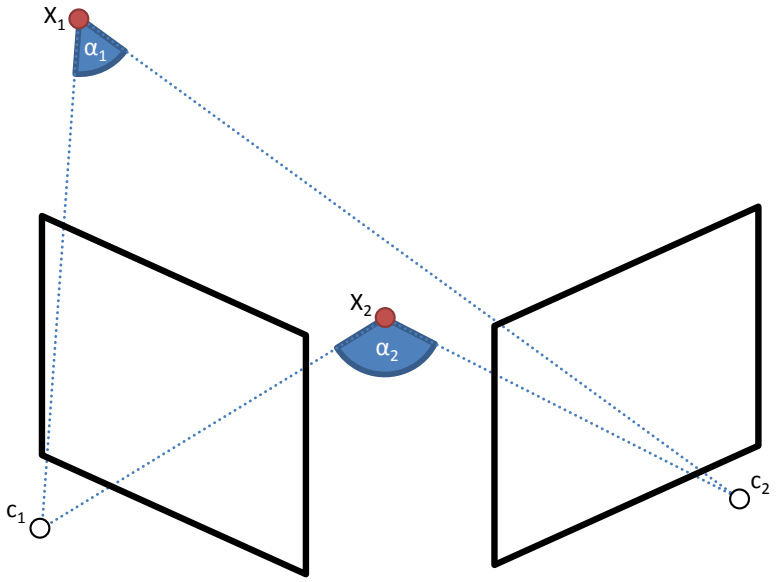


Figure 4.3: Apical angles as a measure for triangulation suitability. A wide angle yields a higher accuracy and thus a reconstructed scene with many large angles has lower spatial jitter.

Measuring the apical angle is obviously an a-posteriori measure since it is only applicable to already triangulated points. In our case we can transform this to an a-priori measure (i.e. applying it before triangulating any new points) since we already have triangulated points available. We keep the decision rule $A \wedge (B \vee C)$ from Section 3.3.2 but replace the pose test A with the following A_2. The rest of the keyframe insertion pipeline remains unmodified, i.e. selecting existing keyframes for triangulation is done with the proposed distance function again where the POI distance $q = \|c_1 - \mu\|$ with c_1 being the first keyframe's camera center and μ the 3D map points' mean.

A_2: Scale-independent pose test using apical angles

The current camera pose is compared to the pose of each already existing keyframe to evaluate whether a new keyframe should be added. In contrast to the publications cited in the following we have transformed the apical angle to an a-priori measure since we calculate it for the PnP inliers before any new point is triangulated. For each existing keyframe K_i :

1. Calculate the dominant apical angle at the map points which are currently PnP inliers like this:
    1. Connect each PnP inlier with the camera center of the current pose and the center of K_i.
    2. Compute each apical angle and dismiss the 5th and the 95th percentile to exclude outliers [32].
    3. Instead of performing a kernel density estimation [68] in order to compute the dominant apical angle as suggested by [77], stick to the finding of [32] that a simpler quality assessment is sufficient. We use the mean of all apical angles:

$$q = \frac{1}{n} \sum_{X} \alpha_X \tag{4.4}$$

2. Dismiss the frame (i.e. let the pose test fail) if $q < 20$, i.e. if the mean apical angle is below 20°. This value turned out to be a reliable cutoff in our evaluation.

4.5 Map Merging

Dealing with multiple maps in Augmented Reality has already been proposed by [9] but they do not attempt to merge the created maps but rather enforce a context switch. Their method of context determination and switching is similar to the relocalization method explained in Section 3.2.1: As soon as tracking degrades, they try to identify which of the other available maps fits better based on the keyframes' SBIs (small blurry images).

Our system on the other hand benefits from a more sophisticated context determination algorithm and allows for less erroneous map switching. This is due to the fact that context-determination based on SBI only works when the maps' keyframes are sufficiently distinct since a small change in the scene like flipping a switch or removing a fuse will most likely not change the SBI at all. As a conclusion, SBI relocalization is appropriate for detection of spatial context switches but is not the method of choice concerning procedural context switches. For the latter case, we benefit from the offline mapping stage: Each chapter (speak: context) was linked with a 3D point map. If at runtime (i.e. when the system is used live) a switch to a different chapter has been detected, the corresponding point map is automatically loaded and tracking continues.

As described earlier, our system has reconstructed an individual 3D map for each chapter. Up until now they are by no means interrelated to each other since they originate from completely independent triangulations. This bears the following challenge: Since 3D reconstruction from projective images is always up-to-scale and each map is placed at the origin, we cannot simply put all maps in one reference frame. This has implications on both the augmentations and the recognition of already visited locations with subsequent map merging. In the following section we will explain these effects and will then continue with our proposed solution.

4.5.1 The effects of scale ambiguity on context-aware SLAM

A context switch not only results in a change of the reference frame but also in a scale change. As a result, scene augmentations that are meant to be maintained over the course of several chapters cannot easily be pinned to the same location in 3D (cf. Figure 4.4).

Figure 4.4: Illustration of scale ambiguity and its effect on scene augmentations. On the left: correctly placed augmentations in the first map. On the right: The same augmentations in the second map, from left to right: No correction of the reference frame, reference frame corrected but incorrect scale, corrected reference frame and correct scale.

The simplest solution to this problem would be to require all augmentations to be defined per chapter and indeed this is feasible in most cases - e.g. after a fuse is removed we no longer have to highlight its position. However, when the workflow is in a certain chapter and the user wanders around visiting locations of a different chapter we want to maintain the tracking by switching to a different map but still maintain the current chapter's augmentation. Differently speaking, we want to maintain the processual context while being able to change the spatial context.

Moreover, if we are able to do this, we can also provide a feedback to the context-determination algorithm by telling him that the user did not advance in the process but just left the scene and guide him back.

The second aspect is that we want to merge maps originating from the same locality: e.g. when the user flips a switch this results in up to three chapters (the initial state, the action and the resulting state) and apart from the switch the scene structure stays the same. Merging the map reduces redundancy, results in a more accurate reconstruction and allows for tracking in the same reference frame (i.e. the camera can be continuously tracked throughout the whole sequence, not only for a single chapter each).

4.5.2 Establishing interrelations between different maps

The most crucial part of map merging is to establish correspondences between two maps and then merge them by minimizing an energy function which is based on those correspondences. There are several types of correspondences one could consider, the most obvious one being correspondences between 3D map points. In this case, an Iterative Closest Point algorithm (ICP) can align two point clouds by minimizing the cumulative distances between corresponding points:

$$\sum_i \|X'_i - (sRX_i + t)\|^2. \tag{4.5}$$

The energy is minimized depending on a transformation with seven degrees of freedom: three translation parameters t, three rotation parameters R and one scale parameter s. This minimization often fails if maps have few common points or the matching is faulty[1]. This is the reason why we propose a different approach which is

- independent of the scene structure (and also does not depend on the used reconstruction algorithm — it can also operate in dense [47] or semi-dense [22] environments and does not require any descriptors)
- invariant towards dynamic scenes (i.e. even works when the structure has changed between the two point clouds)
- non-iterative (except for an optional RANSAC scheme).

We call it *Structure Fitting using Trajectories (SFT)*. It can replace ICP were it fails (e.g. with little redundancy between two maps or distinct scene dynamics) or serve as its initialization.

SFT is based on a modification of the previously discussed region growing: A chapter's map is now not only extended between its boundaries but the tracking continues until it reaches the initialization frames of the neighboring maps. As a result, we have tracked the same camera relative to two different maps which yields two overlapping camera trajectories (cf. Figure 4.5). By aligning them, the two maps can be put into the same coordinate frame without taking their 3D structure into account.

Having two overlapping trajectories means that we have a list of corresponding camera poses. Using this list we can compute a coordinate system transformation (s, R, t) mapping the first onto the second trajectory as follows:

1. Remove all pose pairs where one of the camera poses could not be determined with a good quality (i.e. does not have enough PnP inliers). Abort the whole procedure if less than 10 pairs are left.

[1] "Pure" ICP operates only in the euclidean space, i.e. on the points' coordinates alone. Here, matching means to take the closest point (hence the name: Iterative Closest Point) and is very error-prone if the initial alignment is not already pretty close. There are also variants of ICP using additional point descriptors for matching (like the ones our point map stores alongside the 3D points) but this fails with repetitive structures or viewing angles that differ too much.

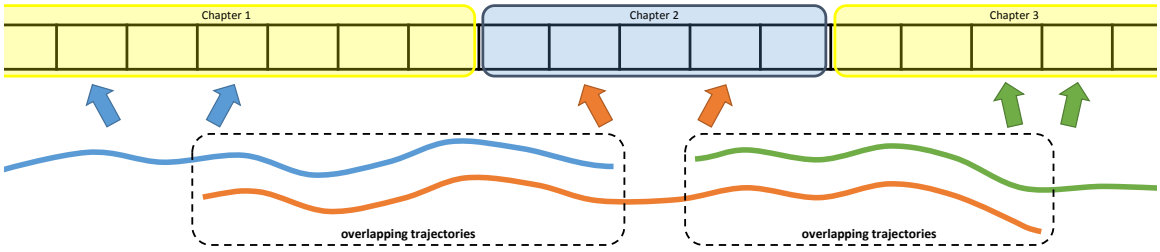


Figure 4.5: Three consecutive chapters with overlapping camera trajectories usable for establishing map interrelations. The two frames used for each of the three initial triangulations are marked with two arrows of the same color.

2. Generate three *virtual points* per pose, the first being placed in the origin: $P^{-1}(0,0,0,1)^\top$ and the others on two axes $P^{-1}(1,0,0,1)^\top$ and $P^{-1}(0,1,0,1)^\top$. These three points parameterize the pose's six degrees of freedom. Since they are synthesized points, we have three exact point matches for each pose pair and thus need no ICP scheme.

3. The transformation aligning the two trajectories is computed similar to many ICP approaches. We use a singular value decomposition of a distance correlation matrix [3, 21] to solve the given least-squares problem and derive the scale from the ratio of standard deviations [34]:

    1. Compute the mean μ_1 of all camera centers from the first trajectory and the mean μ_2 of the second trajectory plus the corresponding standard deviations σ_1, σ_2. It is important to only use the centers for this, since the other virtual points currently have an arbitrary scale — their distance to the center is set to 1 regardless of the true scale (cf. Figure 4.6).

    2. The scale is $s = \sigma_1/\sigma_2$.

    3. Adjust the all virtual points which are not a camera center according to the scale factor.

    4. Calculate the 3×3 distance correlation matrix with $\{p_i\}$ being all virtual points from the first trajectory and $\{q_i\}$ the corresponding ones from the second trajectory:

    $$H := \sum_i (q_i - \mu_2) \cdot (p_i - \mu_1)^\top. \tag{4.6}$$

    5. Reconstruct R and t by a singular value decomposition of H. The details can be found in [3] and we also account for an additional special case mentioned in Section 3.1.4 of [21].

4. The first trajectory can now be fitted to the second one by applying (s, R, t). As a consequence, the first map can be fitted to the second map using the same transformation.

Mathematically, this transformation could be derived from a single pair of corresponding cameras. However, in any real application it is necessary to take as many pairs into account as possible, since the tracking may be inaccurate. As an additionally modification, step 3 can be wrapped in a RANSAC scheme to be more robust towards outliers, i.e. wrongly estimated camera poses. Similar to LO-RANSAC [14], we operate on a non-minimal consensus set to alleviate the effect of noisy inliers.

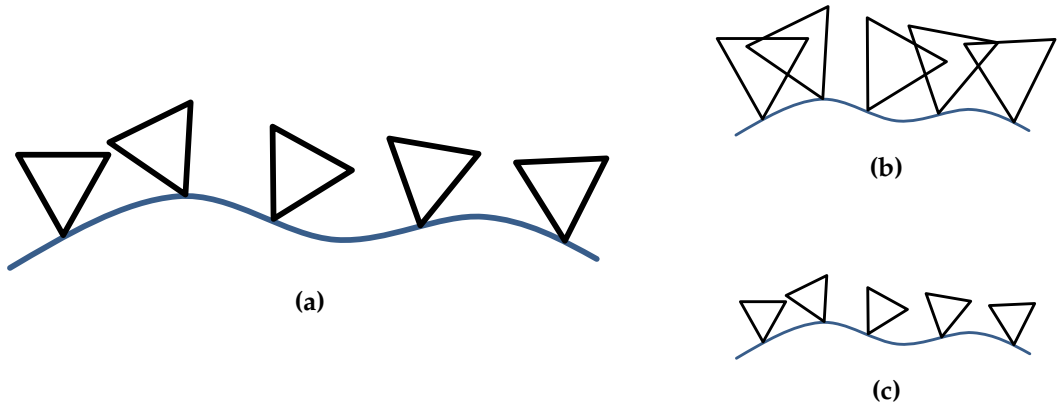


Figure 4.6: Parameterization of a camera trajectory (blue line) using three points per camera (black triangles; the point on the blue line is the camera's origin). The three virtual points are generated with arbitrary scale — although the trajectory (b) is much smaller than (a), only the distances of the camera centers reflect this fact. A correct down scaling of (a) would look like (c) and an can only be achieved after the scale has been determined using the camera centers only.

4.6 From offline SfM to SLAM

In this section we will give a brief résumé of how the offline techniques discussed in this chapter relate to the online SLAM system. The inverse direction was taken in the region growing step where SLAM helped to perform SfM.

Like the workflow segmentation, the discussed offline mapping is done after the recordings have been made. There are little time constraints since the results do not have to be available immediately. This why the reconstruction can be performed with higher accuracy (more features, higher resolutions, loop closures, ...) providing a more precise map. The SLAM tracking goes hand in hand with the context determination since the offline mapping step established a surjective mapping between workflow chapters and point maps and thus the map can be switched whenever a context switch has been determined. This removes the need of a manual initialization by the user and also results in a better relocalization method: As explained in Chapter 2, the context classifier already provides a rough pose estimate and additionally, the set of SBI to search in can be restricted to those attributed to the current chapter. As changes in the environment are explicitly modeled as the result of an action chapter, the SLAM tracking can adapt to them by switching to the

map attributed to this chapter and — after the action was carried out — changing to the next map.

In order to adapt to unforeseen scene changes and unknown parts of the environment, the offline reconstructed maps are not used as fixed, unchangeable tracking model but rather extended in the traditional SLAM way which allows to maintain overlays even in unknown environments (e.g. when the user walks around an object an looks at it from an angle of view not present in any given recording). Additionally, we can guide the user back to the scene even if he wandered off to a previously unknown environment: Since the camera trajectory is known, one could automatically generate an AR overlay indicating the way back like a mini-map in a computer game. Deriving spatial relations is also important if a workflow contains several similar-looking structures which are only distinguishable by their location, e.g. four fuse boxes each in one corner of a room or two similar circuit boards in a switch cabinet. A purely appearance-based approach would fail in this scenario while a combination of appearance knowledge and spatial knowledge solves the problem.

5 Evaluation

To understand the impact of the different modules on the overall performance, we evaluate the system by divided it into its single modules. In the end, show typical tracking results. The module evaluation is done with synthetic images where we know the exact structure, camera poses and keypoint matches and thus can restrict our tests to exactly one of those variables each (e.g. when using the ground truth for keypoint matches, the quality of triangulation can be measured without any interferences of the keypoint matching quality). Except for a qualitative assessment in the end of this chapter, we do not evaluate the quality of our chosen keypoint descriptor but refer to the evaluation performed by its authors [63] and a SLAM-specific evaluation [52], both being the basis of our decision to use ORB.

The reprojection error is given in pixels and is in relation to a 640x480 image. The reconstruction error (euclidean distance between the reconstructed 3D point and the true position) is given as dimensionless quantity since the scale is arbitrary. All data points in the diagrams are the mean value obtained over 100 iterations with the same parameters.

5.1 Synthesis of evaluation data

To have ground truth available and eliminate interferences e.g. from the keypoint matching, we synthesize data for our evaluation. We synthesize scene structures for the evaluation of triangulation and epipolar geometry estimation and pairs of matching camera trajectories for the evaluation of SFT.

We have different types of scene structures which will be explained in the upcoming sections. They all have the following in common: The camera images are 640x480, the camera is linear (i.e. no distortion) and its intrinsics are

$$K = \begin{pmatrix} 100 & 0 & 320 \\ 0 & 100 & 240 \\ 0 & 0 & 1 \end{pmatrix}.$$

3D points are created by a specific creation rule (explained in the next section) and two cameras are placed in a way that every point is visible in both of them. The 3D point is projected in both camera images yielding two matching keypoints. Their position is then modified according to the noise model (a Gaussian distribution with the true location as its mean) to simulate inaccuracies in the keypoint detection.

5.1.1 Centered points

For an accurate evaluation one has to know that in the strict sense the reprojection error depends on the distance to the image center: 5px error at the border means a more accurate reconstruction than 5px right in the center. To account for this, one could measure the reprojection error in angles or — since our images are already synthetic — create all 3D points in one place and rotate both cameras around it so that the point's true projections lie in the centers of both images. All 3D points are 10 units away from each of the cameras and if not mentioned otherwise, the angle between the camera's viewing rays is 10 degrees. When applying Gaussian noise to the projections in order to simulate noise, the keypoints are spread around the center (cf. Figure 5.1). Since we do not want to evaluate the descriptor matching itself, we can take the known matches for granted in this otherwise physically impossible situation[1].

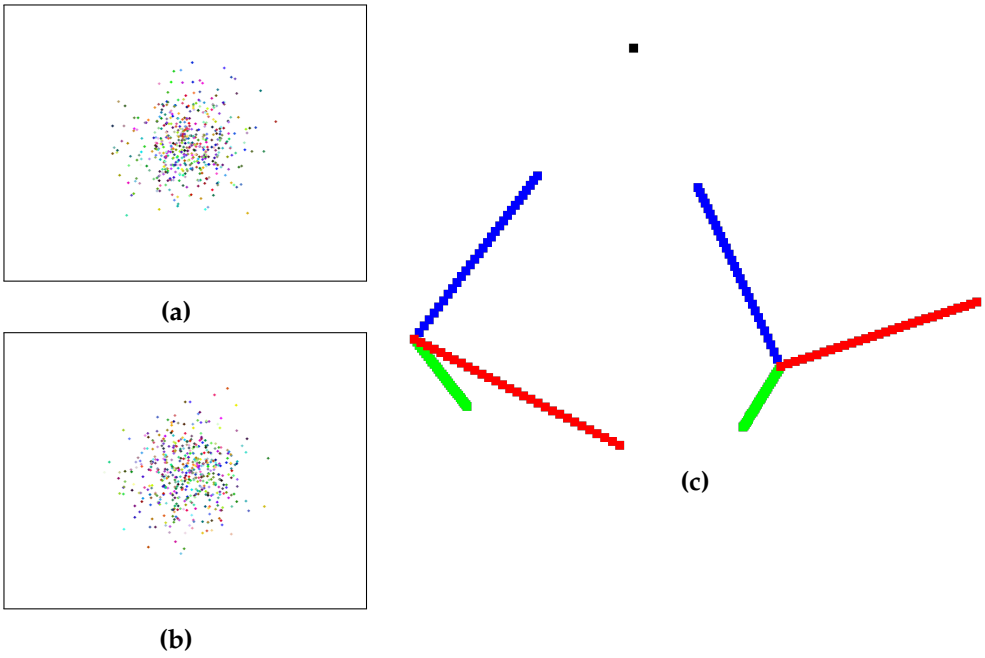


Figure 5.1: Synthetic images used for the evaluation of the triangulation. A scene with 500 points in one spot (black dot in (c)) is projected into the two camera images (a) and (b) with Gaussian noise ($\sigma = 50$ for the purpose of illustration). Both cameras' viewing rays enclose an angle of 45° and meet in the scene points.

[1] In reality, multiple points having the exact same 3D location can not yield different projections, regardless of the noise model. Nevertheless, we use this synthesization approach since generating 500 of these points in one image is the same as generating 500 images with one point each.

5.1.2 Book scene

Estimating the epipolar geometry is not possible from points on a single plane. For this purpose, we also synthesize a scene consisting of two planes as pictured in Figure 5.2.

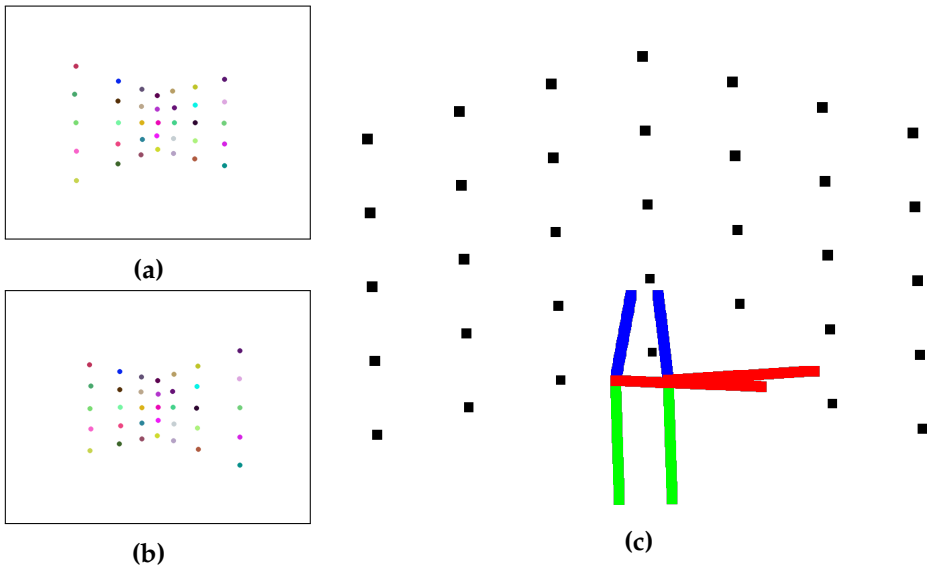


Figure 5.2: Synthetic images used for the evaluation of the epipolar geometry estimation. The points are arranged in two planes forming a book-like structure (c). The two camera images (a) and (b) exhibit a keypoint noise of $\sigma = 1$ and the matches are known (indicated by color). Both camera's viewing rays enclose an angle of 10° and meet in the book's center.

5.1.3 Pairs of matching camera trajectories

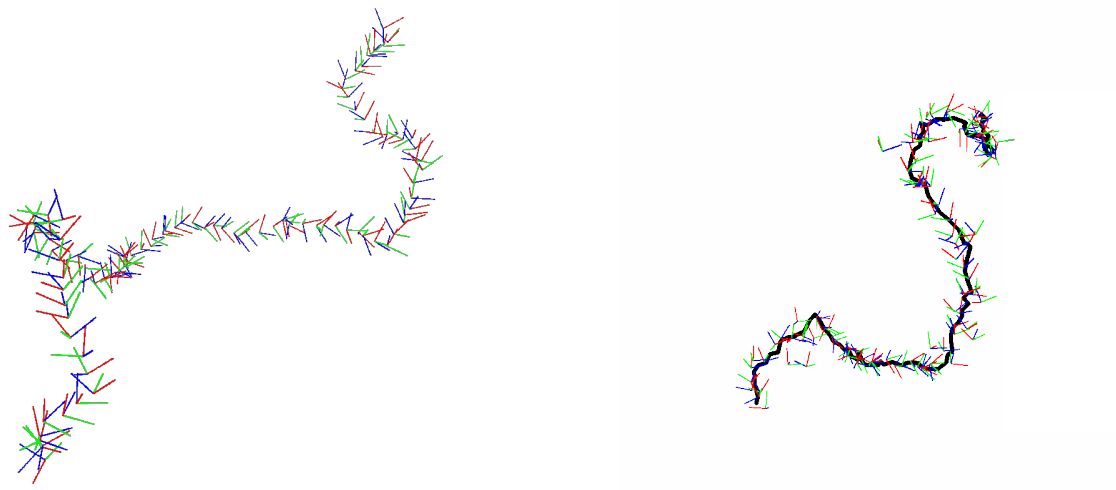
To evaluate *Structure Fitting using Trajectories (SFT)*, we synthesize pairs of camera trajectories: The first trajectory is generated by a random walk starting in the origin and by applying random rotations to the camera orientation. The second one is a by (s, R, t) transformed copy of the first and subject to Gaussian noise both in the camera positions and in the orientations. The noise (R_σ, t_σ) added to each camera pose is modeled as follows, using the Rodrigues formula [29]:

$$\begin{aligned} R_\sigma &:= I + sin\theta\,[r]_x + (1 - cos\theta)\,[r]_x^2 \quad \text{with} \\ \hat{r} &:= \left(\mathcal{N}(0,\sigma^2), \mathcal{N}(0,\sigma^2), \mathcal{N}(0,\sigma^2)\right)^\top, \quad \theta := \|\hat{r}\|, \quad r := \frac{\hat{r}}{\theta} \end{aligned} \tag{5.1}$$

and

$$t_\sigma := \left(\mathcal{N}(0,\sigma^2), \mathcal{N}(0,\sigma^2), \mathcal{N}(0,\sigma^2)\right)^\top. \tag{5.2}$$

Both trajectories contain 100 camera poses and an example can bee seen in Figure 5.3. As pictured in Figure 5.4, already small σ result in distinct noise.



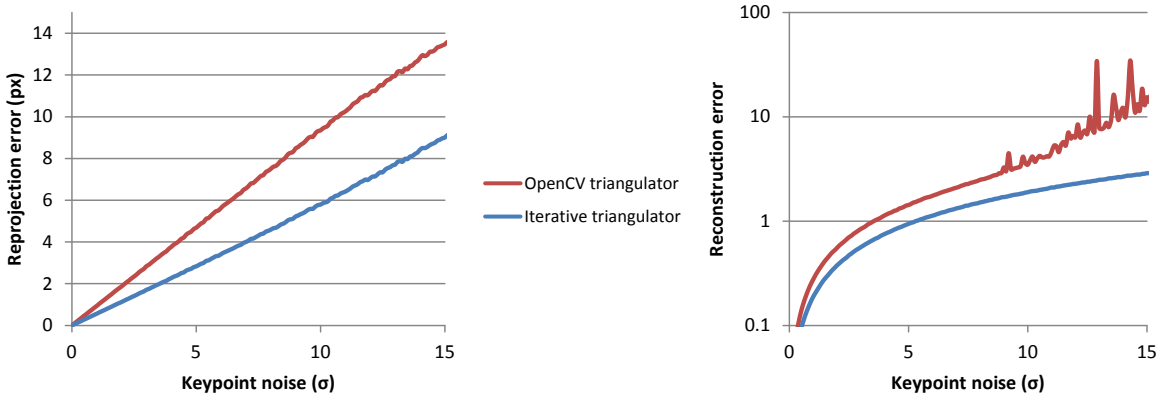
(a) (b)

Figure 5.3: Synthetic trajectories used for the evaluation of SFT. (a) Synthesized trajectory and (b) transformed version with noise of $\sigma = 0.02$. SFT finds the transformation (s, R, t) mapping the first trajectory onto the second one. The solid black line is the trajectory from (a) fitted to the second one.

Figure 5.4: Synthetic trajectories used for the evaluation of SFT. The black line is the trajectory from Figure 5.3a, mapped onto a randomly transformed trajectory with noise of $\sigma = 0.2$.

5.2 Triangulation accuracy in presence of noise

An integral part of our SLAM approach is the triangulation component. As already mentioned, we use an iterative least-squares method [28] and at this point we also want to refer to [66] who provide a basis for our implementation. In the following, we examine the behavior of reprojection error and reconstruction error and compare our module against the implementation of OpenCV 3.0-alpha (non-iterative least-squares).



(a) (b)

Figure 5.5: Our used iterative triangulator vs the OpenCV triangulator, evaluated using the reprojection error (a) and the reconstruction error (b). Please note the logarithmic scale in the latter one and how the OpenCV triangulator becomes unstable. The reconstruction error is given as euclidean distance with an arbitrary scale.

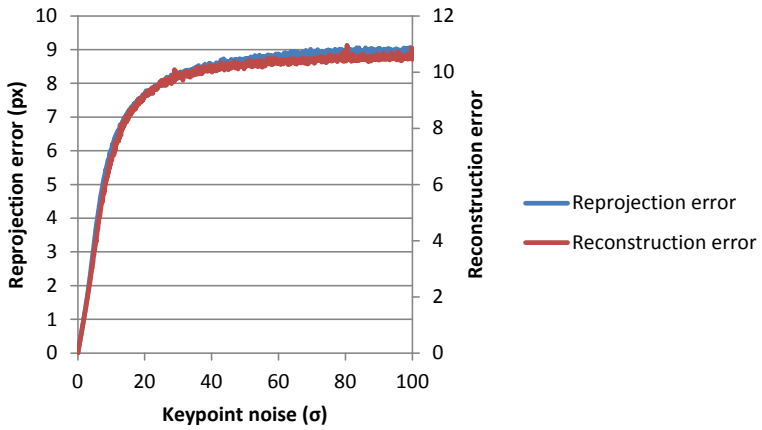


Figure 5.6: Behavior of the reprojection error and the reconstruction error during triangulation when the keypoint noise increases.

When the exact camera positions are known, the reprojection error is an appropriate estimate of the reconstruction error as they correlate (cf. Figure 5.6). This is important because in a real application, the reconstruction error cannot be measured since the true structure is not known. We have chosen a reprojection cutoff in this experiment pictured in Figure 5.6, i.e. all triangulations yielding a reprojection error of more than 20 px are discarded to examine whether the reconstruction error is also influenced by such a threshold. The reconstruction error is dimensionless but it is clear that it follows the same tendency as the reprojection error. In reality, a much lower cutoff is chosen (usually 2 px) but since

we examine the system's boundary areas by using positional noises of up to $\sigma = 100$, we use a rather high one.

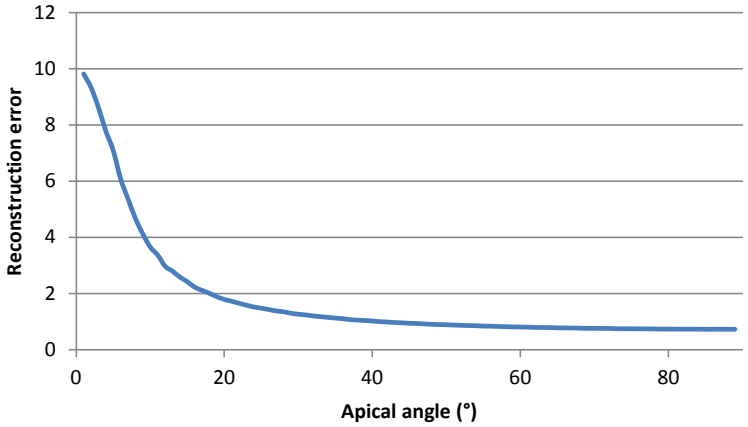


Figure 5.7: Triangulation of 500 points with a keypoint location noise of $\sigma = 5$: Reconstruction error as a function of the apical angle. The data indicates that 20° is a good threshold — this is why we used it for the pose test A_2.

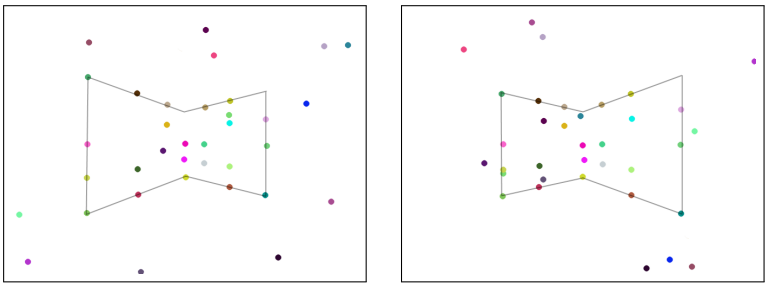
As the last aspect of triangulation, we take a short look at the small baseline problem and plot the reconstruction error as a function of the angle between the camera's viewing rays. In our synthetic scene, this is the same as the apical angle of any of the 3D scene points. Figure 5.7 shows that the closer the angle is to 90°, the more reliable the reconstruction. We have chosen a Gaussian keypoint noise of $\sigma = 5$ in these experiments.

5.3 Epipolar geometry in presence of noise and outliers

One level higher on top of the triangulation lies the estimation of the epipolar geometry which gives us the camera poses needed for the already evaluated triangulation. We now use the synthetic book scene and besides the Gaussian noise in the keypoint locations we also introduce matching outliers as second type of unwanted effects to evaluate their influence on the essential matrix computation. Matching outliers are generated by taking both reprojections of a 3D point and move them to a random location within the image borders.

We first estimate the essential matrix given our synthesized keypoints (cf. Figure 5.5) with the five-point algorithm [50] wrapped in a RANSAC scheme (with a 99.9% confidence and a threshold of 1). We then derive the two camera poses and finally triangulate the keypoints to reconstruct the scene. The keypoint positions are subject to Gaussian noise and the previously explained matching outliers (cf. Figure 5.8).

The error measure used is the mean reconstruction error, i.e. the mean distance of all reconstructed points to their respective ground truth points. Since scale and position of the



(a) (b)

Figure 5.8: The book scene with a keypoint error of $\sigma = 1$ and roughly half of the points being outliers. The outlines have been manually added to illustrate the true structure. The evaluation shows that the reconstruction provides acceptable results with these noise parameters and can also deal with a higher keypoint location noise.

origin are arbitrary in projective reconstruction, we have to align the ground truth points and the reconstructed scene. Again, this done by computing a singular value decomposition of a distance correlation matrix (cf. Section 4.5.2 and [3, 21]) and done non-iteratively since the true 3D-3D matches are known because we synthesized them. As a consequence, the reconstructed scene points $\{X_i\}$ and the true points $\{X'_i\}$ are transformed by (s, R, t) so that

$$\sum_i \|X'_i - (sRX_i + t)\|^2. \tag{5.3}$$

is minimal and is used as our error measure. Again, this measure is dimensionless, since the reconstruction is of arbitrary scale. As the keypoint noise increases, the reconstruction error does as well (cf. Figure 5.9). Of particular note is the ratio of failed attempts to establish the epipolar geometry by estimating the essential matrix: Up until $\sigma = 1$, every single attempt succeeds but then the fail ratio rapidly increases. In the vast majority of cases the reason for failing was that none of the four possibilities in the cheirality check had at least 70% of the triangulated points in front of both cameras, a threshold which we also apply in our actual application.

As a last aspect of the epipolar geometry, we evaluate the robustness towards outliers. We take a fixed keypoint noise parameter of $\sigma = 1$ and plot the reconstruction error as a function of the outlier count. We stop at the value of 30 outliers, since we have 35 scene points and need at least five correspondences for the estimation. The data (cf. Figure 5.10) indicates that the estimation is unaffected by a small number of outliers but that for a reconstruction of appropriate quality, much more than the mathematical minimum of five correspondences are needed.

As a conclusion, the estimation of the epipolar geometry provides acceptable results but would be the first point to improve when a higher accuracy of our system is needed: While the triangulation's error increases steadily with increasing keypoint noise, the estimation of the essential matrix quickly becomes unstable (cf. Figure 5.9) and thus the triangulation results are bad. This is of particular interest, since the initial 3D map — whether acquired

with the help of the user or automatically in the offline mapping step — is always triangulated based on the poses obtained from the essential matrix.

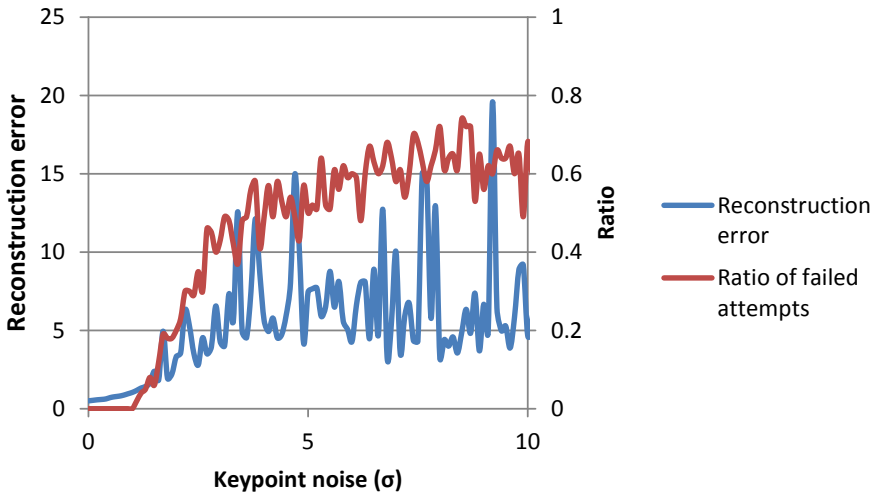


Figure 5.9: Estimation of epipolar geometry and subsequent triangulation of an outlier-free scene with increasing positional noise in the keypoint locations. As the noise increases, the ratio of failed estimation attempts of the epipolar matrix increases, too.

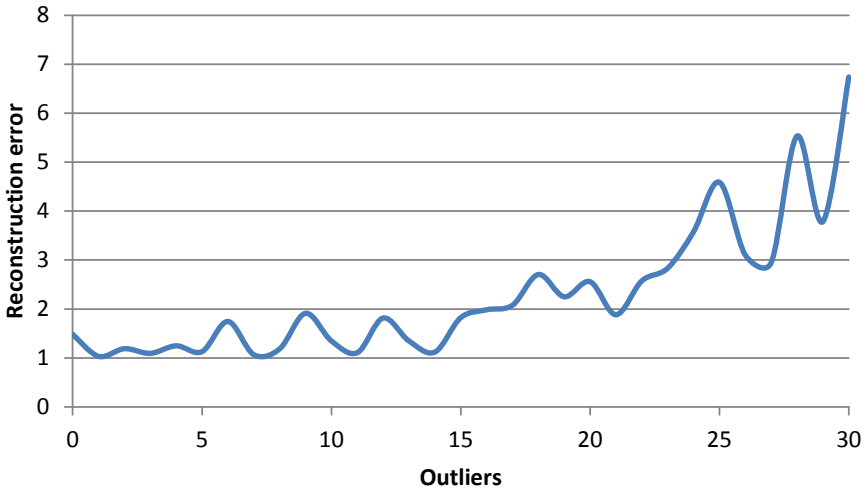


Figure 5.10: Estimating the epipolar geometry and triangulating points with a keypoint noise of $\sigma = 1$. The whole scene has 35 points, so we stop at 30 outliers where the used five-point algorithm is sill applicable.

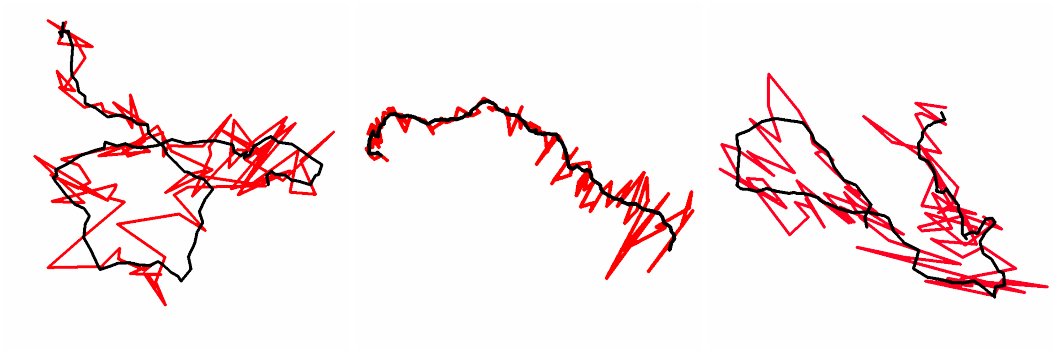
5.4 Structure fitting using trajectories in presence of noise

After synthesizing a camera trajectory c_1 and transforming it by a random transformation T (scale, rotation and translation) and applying noise $\mathcal{N}(0, \sigma)$ as explained in Section 5.1 we obtain a trajectory c_2. We use SFT to compute a transformation T' mapping the first trajectory c_1 to c_2. We then investigate the quality of T' by comparing it to the true transformation T. This comparison is done by computing the transformation $T_\epsilon := T(T')^{-1}$ and computing its rotation angle θ_ϵ and the translation vector's length $\|t_\epsilon\|$. Since the scale is once again arbitrary we divide the translation $\|t_\epsilon\|$ by the length of the trajectory c_1' which is the by T transformed c_1, i.e. the optimal alignment:

$$\|\hat{t_\epsilon}\| = \frac{\|t_\epsilon\|}{length(c_1')}. \tag{5.4}$$

We compute 100 random transformations T per data point and report the mean value for θ_ϵ and $\|\hat{t_\epsilon}\|$ of the SFT result. As we can see in Figure 5.12, two noise-free trajectories ($\sigma = 0$) can be mapped with an error of $\theta_\epsilon = 0$ and $\|\hat{t_\epsilon}\| = 0$ since there is an unique mathematical solution for (s, R, t). When the noise increases, the error does too but stays at acceptable levels until the noise of $\sigma = 0.2$ where the rotational error is below 8° and the translational error around 9% of the ground truth trajectory's length. Figure 5.11 shows some examples of SFT aligning two trajectories with the noise of $\sigma = 0.1$.

Figure 5.13 shows the behavior of the cost function which is minimized by SFT: the sum of distances between all corresponding virtual points. Here, we normalize the scale by using the length of the *noisy* trajectory since SFT does not operate on the ground truth but minimizes the distance to the noisy data. This length increases with increasing noise, since the cameras lie further and further apart. We notice that with a noise higher than $\sigma = 0.2$ the distance function becomes meaningless because the noise leads to the cameras being distributed randomly in space (cf. Figure 5.4 where it is already hard for a human to recognize the noisy trajectory with $\sigma = 0.2$).



(a) **(b)** **(c)**

Figure 5.11: SFT results with a trajectory noise of $\sigma = 0.1$. The red line is the noisy second trajectory and the black line the fitted first one.

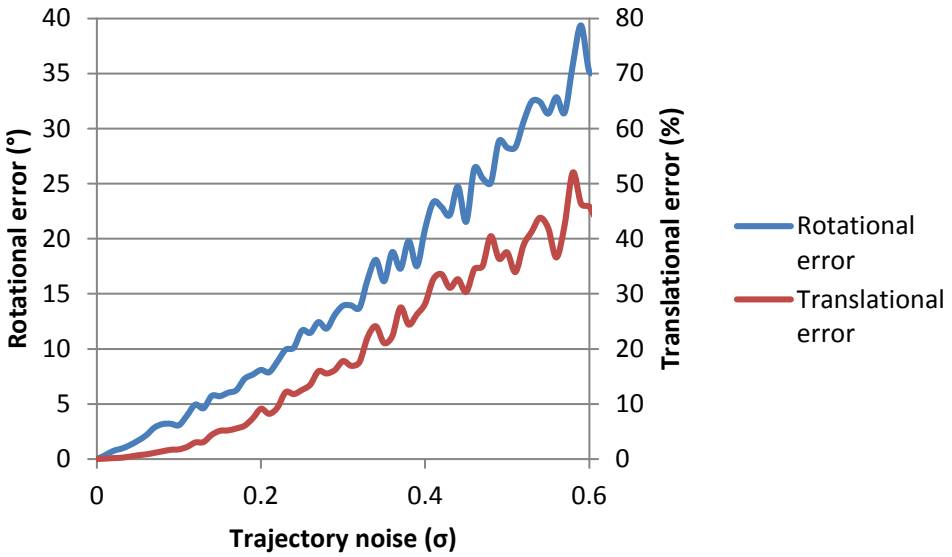


Figure 5.12: Pose error of Structure Fitting using Trajectories as a function of camera pose noise.

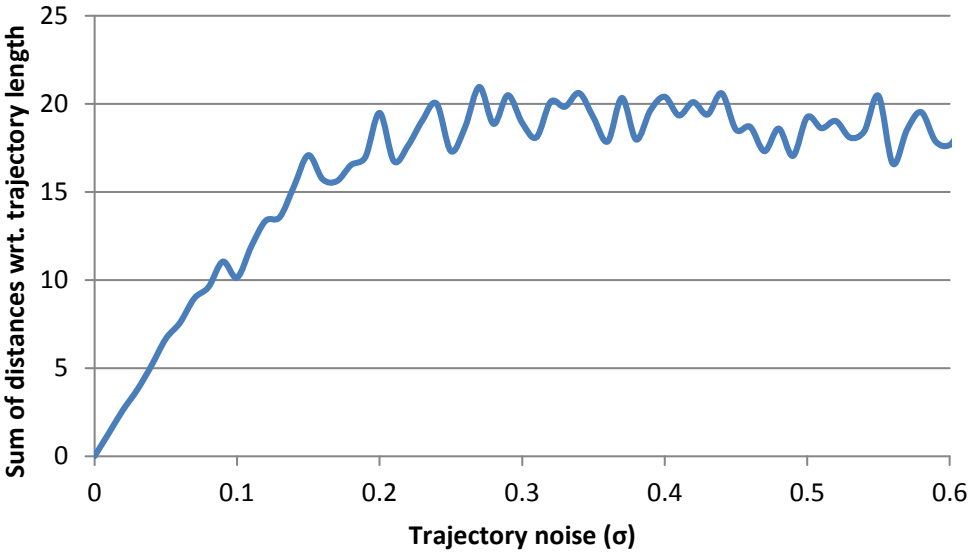
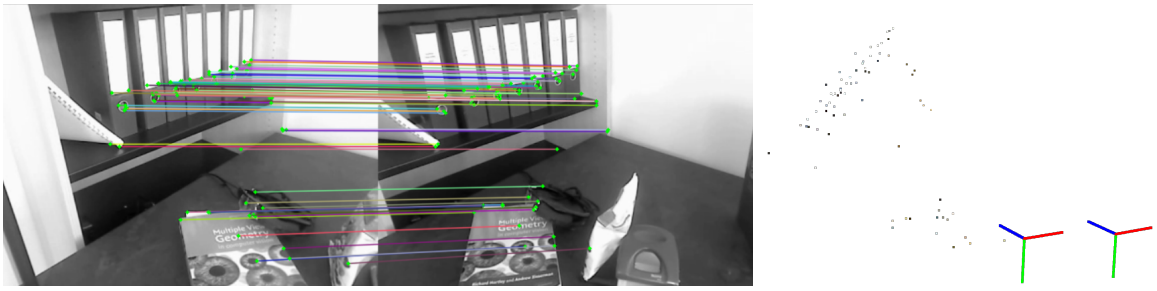


Figure 5.13: Error function used by SFT: The sum of all distances between corresponding virtual points normalized by the ground truth trajectory length. The axis of ordinate is given as relation, i.e. 1 = 100%.

5.5 Tracking results on real-life images

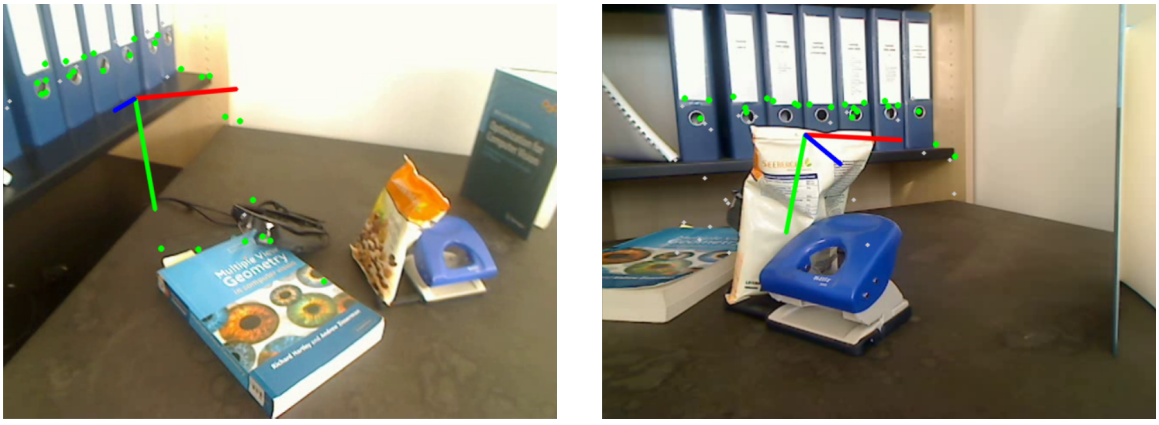
In this section, we show images of our system taken at runtime. Figure 5.14 pictures a small scene initialized by a horizontal parallax movement next to its reconstructed point cloud. Figure 5.15 shows two images taken later in the same sequence with the computed pose and the map point reprojections.

Results of the offline mapping are shown in the following figures: Figure 5.16 shows tracking during an action chapter with a map which was reconstructed from the previous static chapter. The two keyframes have been automatically selected by the method explained in Chapter 4. Figure 5.17 pictures tracking in a slightly modified scene — most of the previously triangulated structure, e.g. the corner on the left, is not available resulting in a tracking based on poorly distributed points. Figure 5.18 shows half of the keyframes in a map which was obtained from merging four different chapters with SFT.



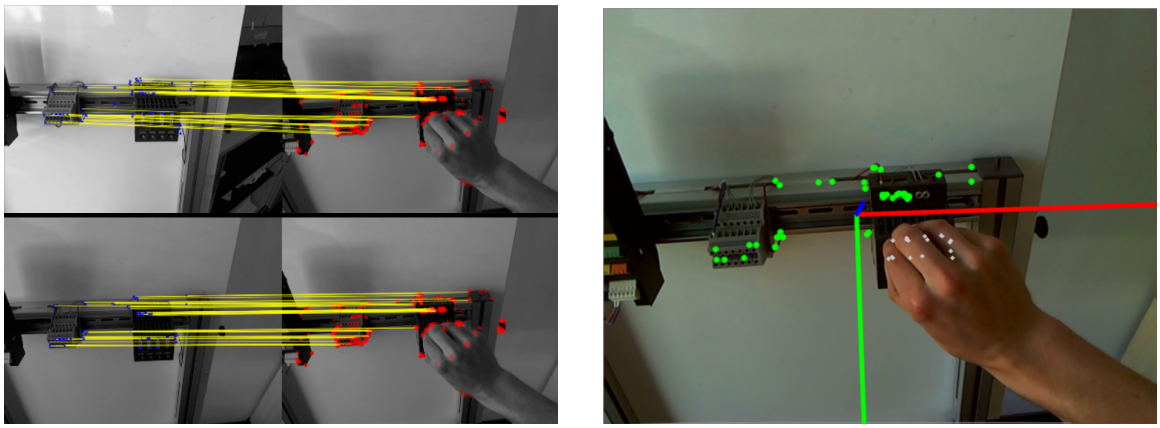
(a) (b)

Figure 5.14: An initial map of a small office scene obtained from a parallax movement. (a) shows the two frames and their keypoint matches used for triangulation. (b) shows the resulting sparse point cloud with the line of folders in the background and the book in the foreground clearly distinguishable.



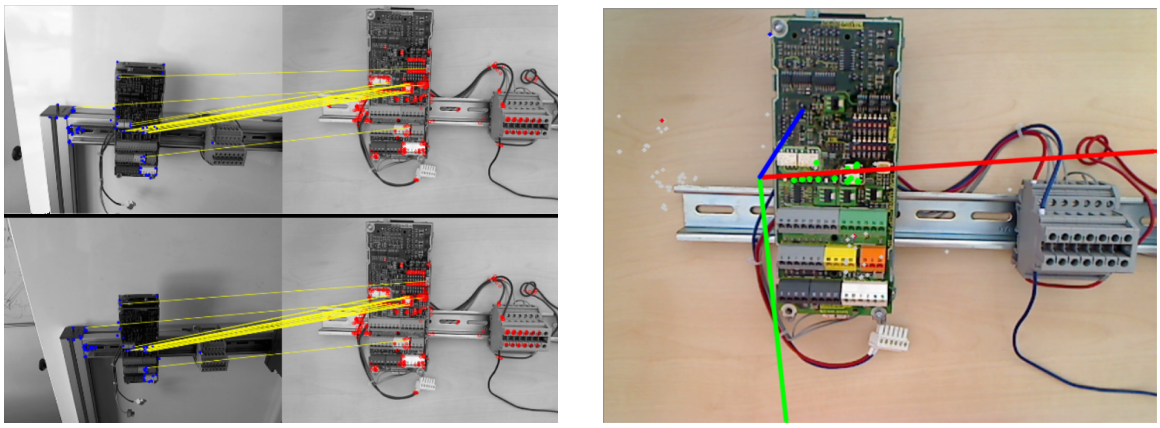
(a) (b)

Figure 5.15: Tracking results in the map initialized in Figure 5.14. The green dots are reprojections of map points which are PnP inliers and the gray ones are outliers.



(a) (b)

Figure 5.16: Tracking during an action chapter based on a map obtained from a static chapter. (a) The keyframes are pictured in the left column and the matches of 3D map points with the current keypoints in the right one. (b) pictures the tracking result — notice the gray outliers due to the occlusion by the hand.



(a) (b)

Figure 5.17: Tracking result in a slightly variated scene based on an offline reconstructed static chapter. (a) The offline selected keyframes are in the left column, the 3D-2D matches in the right one. Notice the isolated matching errors. (b) Tracking result

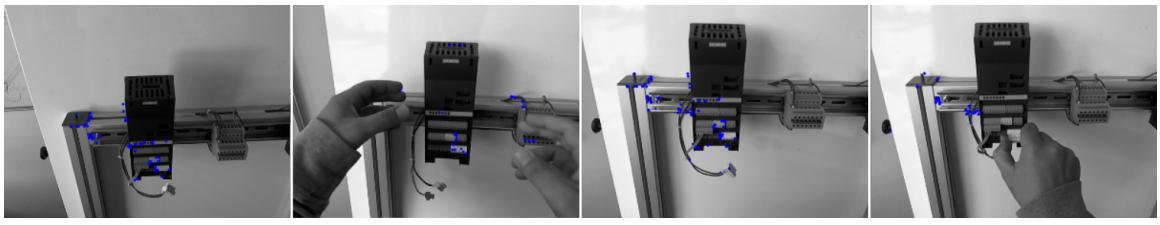


Figure 5.18: Four of the eight keyframes obtained from four different chapters merged into a single map. The blue dots are the map points' reprojections.

6 Conclusion

We have proposed the integration of contextual workflow knowledge into a SLAM tracking system for the purpose of procedural assistance using Augmented Reality.

We have shown that context-dependent SLAM tracking models can be automatically derived from one or multiple reference recordings in a step called *offline mapping*. At runtime, the map used for tracking and mapping can be automatically exchanged whenever a context-switch is detected and the initialization of the SLAM system can be done without any user interaction. We presented different strategies for the extraction of tracking models depending on a segment classification (static, action, movement): Exploiting multiple recordings for static segments, analyzing the optical flow in movement segments and using hand masks in action segments. This extraction could be further improved for the static segments by deriving 3D structure even from minimal movement [80].

The overall approach highly depends on the correct triangulation of 3D points and the computation of the epipolar geometry. The evaluation of both modules showed that a significant improvement can be achieved with the right choice of the triangulator (an iterative one significantly outperforms a non-iterative) and that the essential matrix estimation is as error-prone as it is vital for a good reconstruction. An elaborate matching and outlier rejection scheme before the essential matrix is computed is therefore very important.

Because of the small baseline problem, carefully selected keyframes are crucial for a good reconstruction, too. We proposed an easy-to-compute distance measure integrated into a three-step decision process for keyframe insertion. This procedure ensures that triangulation is only done with two images exhibiting a wide-enough baseline and that the redundancy between keyframes is reduced for the benefit of a faster bundle adjustment and tracking. In the future, this process could be made more restrictive towards clusters of map points: Although the histogram approach already reduces them a lot, they still appear — predominantly in areas where the tracking is not yet stable enough. In addition to a reduction of such clusters by enforcing a minimal distance between points one could also evaluate the effects of a more robust RANSAC consensus set selection like the one from [73] proposed for a robust SLAM system.

The evaluation of our developed technique SFT (Structure Fitting using Trajectories) indicated that SFT is robust towards noisy data and that it is a good complementary approach to ICP in situations where the camera trajectories are available. Because of its nature, SFT can be used as map merging technology for any SLAM approach and its complexity is independent from the used scene structure (a dense reconstruction [47] is computationally no more demanding than a sparse point cloud). Further work concerning SFT

would be to use it as drift-correcting component: If we have two overlapping camera trajectories obtained relative to two different maps, there may also be keyframes which have been added to one of those maps during this overlap. Instead of adding such a keyframe based on only one tracking pose, one could first compute a coordinate system transformation using SFT and then refine the keyframe's pose using both available tracking results. One could also investigate the suitability of SFT for other tracking algorithms, e.g marker tracking: Usually, each marker has its own, predefined coordinate system and two markers do not have any reference to each other. Instead of manually defining this relation beforehand, it can be acquired at runtime as soon as both markers are visible in the same image. Again, one single correspondence would be mathematically sufficient but using multiple ones may yield better results in real applications due to tracking inaccuracies.

The research on context-aware SLAM systems is by no means complete and there are three general aspects we want to work on in the future:

1. Integration of other sensors: e.g. by using an IMU (Inertial Measurement Unit), one could establish a metrical relationship between two keyframes and thus solve the problem of scale ambiguity.

2. A framework for the integration of map extensions created by users. A new view point or a refined map can be feed back into a global database so that the system improves continuously as it is used. This of course requires sanity checks and the protection of the user's privacy.

3. In addition to using context-awareness for SLAM we also want to use SLAM for context-awareness: The 6 DOF camera tracking could aid the classification of movement chapters, new view points could be synthesized to improve the workflow chapter classification and discrepancy checks (checking whether the user performed an action correctly) could be done based on the reconstructed 3D geometry.

List of Figures

Bibliography

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres Solver.

[2] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Bundle adjustment in the large. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 29–42, 2010.

[3] K. Somani Arun, Thomas S. Huang, and Steven D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 9(5):698–700, 1987.

[4] Ronald Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[5] Gabriele Bleser. *Towards Visual-Inertial SLAM for Mobile Augmented Reality*. PhD thesis, Technical University Kaiserslautern, 2009.

[6] Gabriele Bleser, Harald Wuest, and Didier Stricker. Online camera pose estimation in partially known and dynamic scenes. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 56–65. IEEE, 2006.

[7] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker. Technical report, Intel Corporation, Microprocessor Research Labs, 2000.

[8] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 778–792, 2010.

[9] Robert O. Castle, Georg Klein, and David W. Murray. Video-rate localization in multiple maps for wearable augmented reality. In *Proceedings of the International Symposium on Wearable Computers (ISWC)*, pages 15–22, 2008.

[10] Robert O. Castle, Georg Klein, and David W. Murray. Combining monoSLAM with object recognition for scene augmentation using a wearable camera. *Image and Vision Computing*, 28(11):1548–1556, 2010.

[11] Thomas P. Caudell and David W. Mizell. Augmented reality: an application of heads-up display technology to manual manufacturing processes. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*, volume 2, pages 659–669, 1992.

[12] Frédéric Chenavier and James L. Crowley. Position estimation for a mobile robot using vision and odometry. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2588–2593. IEEE Comput. Soc. Press, 1992.

[13] Alessandro Chiuso, Paolo Favaro, Hailin Jin, and Stefano Soatto. 3-d motion and structure from 2-d motion causally integrated over time: Implementation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science, pages 734–750, 2000.

[14] Ondřej Chum, Jiří Matas, and Josef Kittler. Locally optimized RANSAC. In *DAGM Symposium*, pages 236–243, 2003.

[15] Fred Daum. Nonlinear filters: beyond the Kalman filter. *IEEE Aerospace and Electronic Systems Magazine*, 20(8):57–69, August 2005.

[16] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the International Conference on Computer Vision (ICCV)*, volume 2, pages 1403–1410. IEEE, 2003.

[17] Andrew J. Davison, Ian Reid, Nicholas Molton, and Olivier Stasse. MonoSLAM: real-time single camera SLAM. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–67, 2007.

[18] Zilong Dong, Guofeng Zhang, Jiaya Jia, and Hujun Bao. Keyframe-based real-time camera tracking. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1538–1545, 2009.

[19] Zilong Dong, Guofeng Zhang, Jiaya Jia, and Hujun Bao. Efficient keyframe-based real-time camera tracking. *Computer Vision and Image Understanding*, 118:97–110, 2014.

[20] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, June 2006.

[21] D.W. Eggert, A. Lorusso, and R.B. Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6):272–290, March 1997.

[22] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[23] Christopher Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. In *Photogrammetric Computer Vision*, 2006.

[24] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why Does Unsupervised Pre-training Help Deep Learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[25] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm

for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[26] Steffen Gauglitz, Chris Sweeney, Jonathan Ventura, Matthew Turk, and Tobias Hollerer. Live tracking and mapping from both general and rotation-only camera motion. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 13–22. IEEE, 2012.

[27] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, pages 147–151, 1988.

[28] Richard I. Hartley and Peter Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.

[29] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[30] Philipp Hasper and Nils Petersen. SLAM for dynamic AR environments. 2014.

[31] Philipp Hasper, Nils Petersen, and Didier Stricker. Remote Execution vs. Simplification for Mobile Real-time Computer Vision. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 156–161, 2014.

[32] Michal Havlena, Akihiko Torii, and Tomas Pajdla. Efficient structure from motion by graph optimization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 100–113, 2010.

[33] Steven J. Henderson and Steven K. Feiner. Augmented reality in the psychomotor phase of a procedural task. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 191–200. IEEE, October 2011.

[34] Berthold K. Horn, Hugh M. Hilden, and Shahriar Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7):1127, July 1988.

[35] Berthold K. Horn and Brian G. Schunck. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, November 1981.

[36] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual SLAM for RGB-D cameras. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 2100–2106. IEEE, November 2013.

[37] Georg Klein and David W. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 1–10. IEEE, November 2007.

[38] Georg Klein and David W. Murray. Improving the Agility of Keyframe-Based SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV) 2008*, 2008.

[39] Johannes Köhler, Alain Pagani, and Didier Stricker. Detection and Identification Techniques for Markers Used in Computer Vision. In *Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling and Engineering (IRTG 1131 Workshop)*, pages 36–44, 2011.

[40] Dieter Koller, Gudrun Klinker, Eric Rose, David Breen, Ross Whitaker, and Mihran Tuceryan. Real-time vision-based camera tracking for augmented reality applications. In *ACM symposium on Virtual reality software and technology (VRST)*, pages 87–94, New York, New York, USA, 1997. ACM Press.

[41] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal of Computer Vision*, 81(2):155–166, July 2008.

[42] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[43] Bruce D Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 130, pages 674–679, 1981.

[44] Alessandro Mulloni, Mahesh Ramachandran, Gerhard Reitmayr, Daniel Wagner, Raphael Grasset, and Serafin Diaz. User friendly SLAM initialization. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 153–162, 2013.

[45] Raúl Mur-Artal and Juan D Tardós. Fast Relocalisation and Loop Closing in Keyframe-Based SLAM. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

[46] Richard A. Newcombe, Andrew J. Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew W. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[47] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. DTAM: Dense tracking and mapping in real-time. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2320–2327, 2011.

[48] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, and Caroline Suen. Unsupervised Feature Learning and Deep Learning Tutorial, 2013.

[49] Susanna Nilsson and Björn Johansson. Fun and usable: augmented reality instructions in a hospital setting. In *Proceedings of the 2007 conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artifacts and environments - OZCHI '07*, page 123, 2007.

[50] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–77, June 2004.

[51] David Nistér, O Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, number C, pages 652–659, 2004.

[52] Zhen Peng. *Efficient matching of robust features for embedded SLAM*. Diploma thesis, University of Stuttgart, 2012.

[53] Nils Petersen, Alain Pagani, and Didier Stricker. Real-time modeling and tracking manual workflows from first-person vision. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 117–124. IEEE, October 2013.

[54] Nils Petersen and Didier Stricker. Adaptive Search Tree Database Indexing for Hand Tracking. In *Proceedings of the International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing (CGVCVIP)*, 2012.

[55] Nils Petersen and Didier Stricker. Learning task structure from video examples for workflow tracking and authoring. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 237–246, 2012.

[56] Nils Petersen and Didier Stricker. Morphing billboards for accurate reproduction of shape and shading of articulated objects with an application to real-time hand tracking. In *Proceedings of CompImage*, 2012.

[57] Nils Petersen and Didier Stricker. Morphing billboards: an image-based appearance model for hand tracking. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, January 2014.

[58] Christian Pirchheim and Gerhard Reitmayr. Homography-based planar mapping and tracking for mobile phones. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 27–36, 2011.

[59] Christian Pirchheim, Dieter Schmalstieg, and Gerhard Reitmayr. Handling Pure Camera Rotation in Keyframe-Based SLAM. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 229–238, 2013.

[60] Katrin Pirker, Matthias Rüther, and Horst Bischof. CD SLAM - continuous localization and mapping in a dynamic world. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pages 3990–3997. IEEE, September 2011.

[61] Dirk Reiners, Didier Stricker, Gudrun Klinker, and Stefan Müller. Augmented Reality for Construction Tasks: Doorlock Assembly. In *Proceedings of the International Workshow on augmented Reality (IWAR)*, pages 31–46, 1998.

[62] Edward Rosten and Tom W. Drummond. Machine Learning for High-Speed Corner Detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 430–443, 2006.

[63] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011.

[64] Gerhard Schall, Daniel Wagner, Gerhard Reitmayr, Elise Taichmann, Manfred Wieser, Dieter Schmalstieg, and Bernhard Hofmann-Wellenhof. Global pose estimation using multi-sensor fusion for outdoor Augmented Reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 153–162. Ieee, October 2009.

[65] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[66] Roy Shil. Simple triangulation with OpenCV from Hartley & Zisserman. http://www.morethantechnical.com/2012/01/04/simple-triangulation-with-opencv-from-harley-zisserman-w-code/ (visited 20.6.2014).

[67] Jun Shimamura, Masashi Morimoto, and Hideki Koike. Robust vSLAM for dynamic scenes. In *Proceedings of the 12th IAPR Conference on Machine Vision Applications*, pages 344–347, 2011.

[68] Bernard. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[69] Gilles Simon, Andrew W. Fitzgibbon, and Andrew Zisserman. Markerless tracking using planar structures in the scene. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, pages 120–128, 2000.

[70] Iryna Skrypnyk and David G. Lowe. Scene Modelling, Recognition and Tracking with Invariant Image Features. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 110–119, 2004.

[71] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the World from Internet Photo Collections. *International Journal of Computer Vision*, 80(2):189–210, December 2007.

[72] Aleksandar Stojanovic and Michael Unger. Robust Detection of Point Correspondences in Stereo Images. *Acta Polytechnica*, 47(4-5), 2007.

[73] Wei Tan, Haomin Liu, Zilong Dong, Guofeng Zhang, and Hujun Bao. Robust Monocular SLAM in Dynamic Environments. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 209–218, 2013.

[74] Arthur Tang, Charles Owen, Frank Biocca, and Weimin Mou. Comparative effectiveness of augmented reality in object assembly. In *Proceedings of the Conference on Human factors in computing systems (CHI)*, page 73, 2003.

[75] Sebastian Thrun. Probabilistic Algorithms in Robotics. *AI Magazine*, 21(April):93–109, 2000.

[76] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, November 1992.

[77] Akihiko Torii, Michal Havlena, Tomas Pajdla, and Bastian Leibe. Measuring camera translation by the dominant apical angle. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7, 2008.

[78] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment—a modern synthesis. *Vision algorithms: theory and practice*, pages 298–372, 2000.

[79] Brian Williams, Georg Klein, and Ian Reid. Real-Time SLAM Relocalisation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[80] Fisher Yu and David Gallup. 3D Reconstruction from Accidental Motion. In *Proceedings of the onference onC Computer Vision and Pattern Recognition (CVPR)*, 2014.

[81] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.