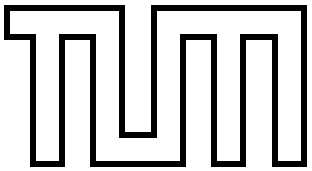


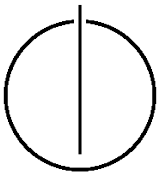
FAKULTÄT FÜR INFORMATIK

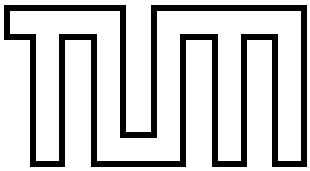
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Semi-dense visual SLAM on mobile devices

Thomas Schöps





FAKULTÄT FÜR INFORMATIK

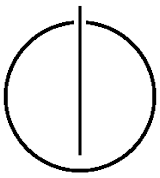
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Semi-dense visual SLAM on mobile devices

Semi-dichtes visuelles SLAM auf mobilen Endgeräten

Author:	Thomas Schöps
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	M.Sc. Jakob Engel
Date:	May 15, 2014



I assure the single handed composition of this master's thesis only supported by declared resources.

Munich, May 15, 2014 Thomas Schöps

Abstract

In this Master's Thesis a visual SLAM (simultaneous localization and mapping) system for a monocular camera is developed based on a novel visual odometry system [16], which is able to run in real-time on current-generation Android smartphones.

In contrast to traditional approaches, the proposed system uses dense image alignment with the Lucas-Kanade method to estimate relative video frame poses based on estimated semi-dense depth maps. Due to using a monocular color or grayscale camera only, the estimated scale of the scene may drift in addition to translational and rotational drift. The system therefore estimates loop closures as similarity transforms in the $\mathrm{Sim}(3)$ group, i.e. consisting of relative position, orientation and also scale.

The base components of the odometry system are pose tracking and mapping for estimating depth maps. The SLAM system adds keyframe selection to keep a sparse set of good frames in memory, constraint search with pose-based and appearance-based candidate finding for building up a pose graph, and optimization to refine frame poses based on the pose graph constraints.

Real-time operation on Android devices is achieved by operating on a lower image resolution and SIMD optimization with the NEON instruction set used in assembler code. For demonstrating the capabilities of the system, a VR (virtual reality) and an AR (augmented reality) demo were implemented for Android devices. The latter takes advantage of the semi-dense depth maps estimated by the system by allowing simulated objects to collide with the real scene.

In evaluations with real and synthetic benchmark sequences, the numerical accuracy of the system is shown. Additionally, especially the ability to cope with large scale changes is shown in a qualitative way by tests with further sequences.

Contents

1 Introduction

Figure 1.1: Augmented reality demo running on a Sony Xperia Z1 smartphone. The system continuously estimates a point cloud of the scene geometry which is visible in a camera image. For this screenshot, the point cloud is fixed at a certain position marked by the white camera model and shown from a different perspective. The point color depends on the distance to the original camera position.

Modern smartphones contain a variety of different sensors to record information about their environment. Among the most common ones are cameras, an accelerometer, a magnetometer and a gyroscope. However, with today's software these devices are still limited to having very simple sense of their environment: they can for example record photos and videos, but in general do not "understand" what can be seen on them. They use other sensors to roughly estimate their position and orientation, but this limited precision also limits the applications: smartphones typically adjust their screen orientation to the orientation the device is held in, can display an approximate compass, or determine their approximate position via GPS or wireless networks for e.g. navigation. The accuracy of the latter method is usually not better than a few meters, and often worse.

More possibilities become available if the precise 3D pose of the device can be determined in real-time. It is a fundamental requirement for example for augmented reality

(AR), as shown in Fig. 1.1, where artificial content is rendered into a camera image. This precision can be accomplished by exploiting image information recorded by the device's cameras. A basic method for doing this is the use of well distinguishable visual markers. They are placed in the scene and recognized in the device's camera images, allowing to easily estimate the relative pose of the device to the markers. Using image templates as markers, this is a primary method used today by existing commercial AR applications on smartphones such as String [48], Wikitude [58], Layar [33] or junaio [24]. While this works well for its purpose (and markers on products like packaging or magazines act nicely as triggers to show the augmentation), it requires to alter the scene and is limited to small spaces where markers can be applied.

A more sophisticated approach is visual odometry: with a video sequence as input, the relative device pose of each frame to a previous frame is determined. Accumulating these poses up allows to get a global pose estimate relative to the first frame. With time however, such systems also accumulate drift: as each relative pose estimate contains an error, however small it may be, the estimated pose deviates more and more from the real pose. Thus, while visual odometry systems are well applicable to e.g. pose tracking for virtual reality (VR) systems where only local correctness is necessary, they fail as soon as global correctness is required.

To address this, structure-from-motion respectively simultaneous localization and mapping (SLAM) systems have been invented. They estimate the device motion (localization) at the same time as a persistent model of the environment (mapping). This goes hand-in-hand as one is required to estimate the other: given an environment model and a new sensor reading, the location from which the new sensor reading was determined can be estimated. On the other hand, knowing the device position, the model can be updated with new sensor readings. Using the environment model, pose drift can be determined and accounted for with help of loop closures: in addition to pose estimates for new frames, relative pose estimates are done between other overlapping frame pairs. This allows to globally optimize the world model, removing drift. Furthermore, the world model may be used for additional reasoning about the world, e.g. in robotics applications.

Early visual odometry and visual SLAM approaches were feature-based, i.e. they first abstract information from an image into features, most importantly keypoints or lines. Based on such a sparse representation, they then estimate poses using the features only. This discards valuable information, in general leading to less robustness. This is most apparent for the case of images which do not contain the specific type of feature being detected at all, e.g. for a keypoint-based system and images containing predominantly (curved) lines only. Recently, direct respectively dense methods became popular which use raw image information without relying on features.

1.1 Problem statement

In this thesis, a semi-dense visual SLAM system is developed which only uses information from a single, ordinary camera. One of the main challenges is the scale ambiguity inherent to this approach: neglecting effects such as blurring, the information measured by such cameras for each image pixel only consists of the color in the direction of the light hitting the pixel's sensor(s). Thus, a smaller or larger but otherwise identical scene would result

in the exact same image being taken - an effect which was used for early special effects in movies, for which small-scale models of scenes were built in order to e.g. simulate destruction of buildings by destructing the toy house models. For SLAM systems, this effect means that in addition to pose drift in position and orientation, deviation in scale is an additional type of drift which has to be accommodated for.

Furthermore, an additional requirement is the ability of the system to run in real-time on a modern smartphone. While smartphone processors quickly become more powerful, they are still much slower than their desktop counterparts, requiring some adaptions to be able to run the system in real-time.

In order to show the capabilities of the system, a VR demo and an AR demo are developed. The AR demo shows how the information given by the semi-dense depth maps estimated by the system can be taken advantage of for physical interactions of simulated objects with the real scene.

1.2 Outline

Chapter 2 first explains the specific background knowledge required to understand the remainder of this thesis. The chapter presents the camera model used, allowing to associate 2D pixel coordinates in an image with their observation directions respectively observed 3D points. Furthermore, it contains a practical introduction to the $\mathrm{SE}(3)$ and $\mathrm{Sim}(3)$ Lie groups for pose representation, which are well suited for performing optimization on them. Non-linear optimization methods, uncertainty propagation and epipolar geometry are addressed shortly in addition.

Chapter 3 gives an overview of related work regarding SLAM approaches, divided up into feature-based methods and direct methods.

Chapter 4 first presents an overview of the complete proposed SLAM system before describing each of its components in detail: pose tracking, keyframe selection, depth mapping, constraint search and graph optimization.

Chapter 5 is about the system's implementation with focus on specifics of mobile devices. In particular, it presents several methods for camera calibration and describes the NEON optimizations which were performed for ARM processors.

Chapter 6 describes the two demos which were developed to demonstrate the system. It especially details how a collision model is generated from semi-dense depth maps in the AR demo.

Chapter 7 presents the results of various evaluations, both numerical evaluation conducted with benchmarks and qualitative evaluation with additional scenes.

Chapter 8 sums up what is presented in this thesis and gives a list of ideas for future enhancements of the system.

2 Background

This chapter lays out the theoretical foundations required to understand the proposed system. First it describes homogeneous coordinates in Sec. 2.1 and the camera model employed in the system in Sec. 2.2. In Sec. 2.3 it then explains the $\mathrm{SE}(3)$ and $\mathrm{Sim}(3)$ representations used for pose representations and optimization of poses. The next section, 2.4, shows how this optimization is done in general. Finally, the chapter briefly introduces uncertainty propagation in Sec. 2.5 and the principle of stereo comparisons, a tool to estimate depth in camera images, in Sec. 2.6.

2.1 Homogeneous coordinates

Geometrical calculations are commonly done by use of linear algebra, which means representing points or directions as vectors and transforming them by matrix multiplication. Regarding notation, vectors and matrices are denoted in **bold** in this thesis. Linear algebra however lacks the ability to represent translations which are non-linear transformations, or projective transformations which occur e.g. when projecting 3D points onto the image plane of a camera. To remedy this, homogeneous coordinates are introduced: each vector is defined to be equivalent to a set of vectors augmented with one further non-zero dimension z as follows (for the example of two dimensions):

$$\begin{pmatrix} x \\ y \end{pmatrix} \hat{=} \begin{pmatrix} xz \\ yz \\ z \end{pmatrix} \tag{2.1}$$

The right hand side in above equation is then called a **homogeneous vector**. A vector can be converted to a homogeneous vector by setting an arbitrary value for z. For efficiency and numerical reasons, 1 is most suited so the other components need not to be changed. A homogeneous vector can be converted back to an ordinary vector by dividing all components by z and dropping the last component.

Calculations with homogeneous vectors are performed exactly as with ordinary vectors, the only difference being the higher dimensionality. This effectively allows to represent translations in matrix-vector multiplications by setting the matrix elements being multiplied with the last component of the vector non-zero. It also enables to defer the division by z until the end of calculations to the step in which vectors are converted back to a non-homogeneous representation. As all translations are multiplied by z and divided by it again at the end, they still act as expected even if z is different from 1. As an example, this shows how a homogeneous matrix-vector multiplication results in a translation and

division when going back to a non-homogeneous representation:

$$\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ q_x & q_y & q_z \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \triangleq \begin{pmatrix} \frac{x + d_x}{q_x x + q_y y + q_z} \\ \frac{y + d_y}{q_x x + q_y y + q_z} \end{pmatrix} \tag{2.2}$$

A further advantage of this is that directions can be naturally represented by setting the homogeneous coordinate to 0, which can be seen as moving the point infinitely far away. The same results when subtracting two points, assuming that their homogeneous components are normalized to the same value. Effectively such vectors are not affected by translations, which is to be expected from directions.

Because of the equivalence of differently scaled homogeneous vectors, the resulting degrees of freedom of transformations in this space are one less than what may be expected. For example, projective transformations in a 3-dimensional homogeneous space, represented as 3×3 matrices, have a resulting 8 degrees of freedom. Fig. 2.1 visualizes the equivalence of vectors for the example of a 2-dimensional homogeneous space.

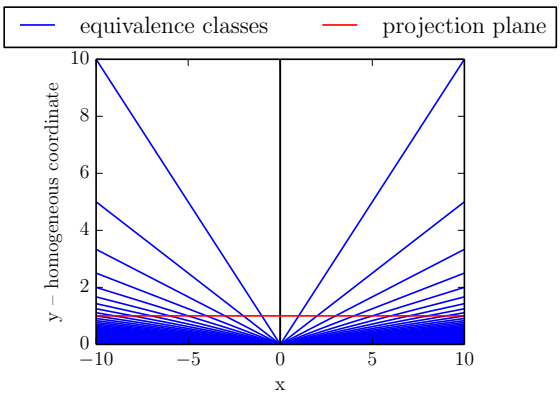


Figure 2.1: Example of equivalence classes for a 2-dimensional homogeneous space. Only positive y are shown. Each (blue) homogeneous line is equivalent to its non-homogeneous intersection point with the (red) projection plane at $y = 1$. The lines correspond to the transformation of a non-zero point on them with scaled identity matrices $s\mathbf{I}, s \in \mathbb{R}$.

2.2 Camera model

2.2.1 Theory

The **pinhole camera model** used in this thesis alludes to early pinhole cameras, which basically consisted of a box with a small hole in its front side (see Fig. 2.2, left). Light rays entering the box through the hole hit the image plane on the opposite box side where a light sensitive film is placed. The light reacts with the film, darkening the spots where light hits it. By lighting the film for some time, an inverted image (negative image) is created on the film, which can later be inverted again to create the final photo.

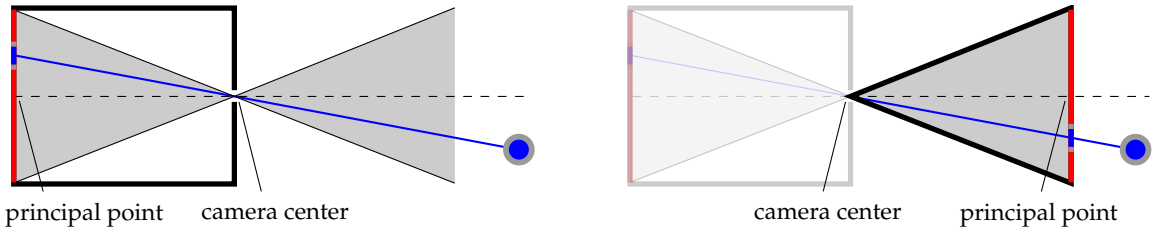


Figure 2.2: **Left:** Pinhole camera sketch. The blue object is projected along the blue line onto the red image plane inside the camera box, creating an inverted image. All rays pass through a single point, the camera hole, called the camera center. The point at which a ray (dashed) passing the camera center perpendicularly to the image plane intersects it is called the principal point. The field of view (shaded in gray) is determined by the size of the image plane and its distance to the camera center. **Right:** Camera model sketch after flipping the image plane to the front of the camera center, creating a virtual image plane. In this model, objects are projected to their final position in the image.

The position of the box hole is called the **camera center**. Ideally it is a single point through which all light rays contributing to the image pass. The **principal point** is the point on the image plane which is hit by the ray going through the camera center perpendicularly to the image plane. The **focal length** is the physical distance between image plane and camera center.

Virtually flipping the image plane to the front side of the camera eases imagination. With this, light rays hit the **virtual image plane** at their final position in the image: see Fig. 2.2, right.

Real pinhole cameras required long lighting times (up to some hours) to allow enough light to enter the box and darken the film. Because of this, today's cameras use lenses to gather and focus the light. However, lenses introduce a set of distortions. Apart from effects which are not modeled here (e.g. depth of field, which means that objects in front or behind the focal distance become blurred), most importantly lenses cause radial image distortion depending on the distance of an image point to the principal point on the image (see Fig. 2.3). This means that circular image regions centered on the principal point are radially moved inwards or outwards relative to the un-distorted image.

To account for this, different models can be applied, which are based on warping between distorted and un-distorted image coordinates with some function. This warp can be applied as a pre-processing step on all incoming images. A warped image is no longer rectangular, but if fitting a rectangle around it, invalid image parts can easily be ignored or alternatively the image cropped to the largest rectangular region containing only valid pixels. As an independent pre-processing step, this warp is neglected in the following. Sec. 5.2 presents some existing methods which may be used to compensate for it.

2.2.2 Projection

In practice, the most common operations related to the camera model are calculating the **projected** pixel coordinates of a 3D point and its inverse, determining the **un-projected**

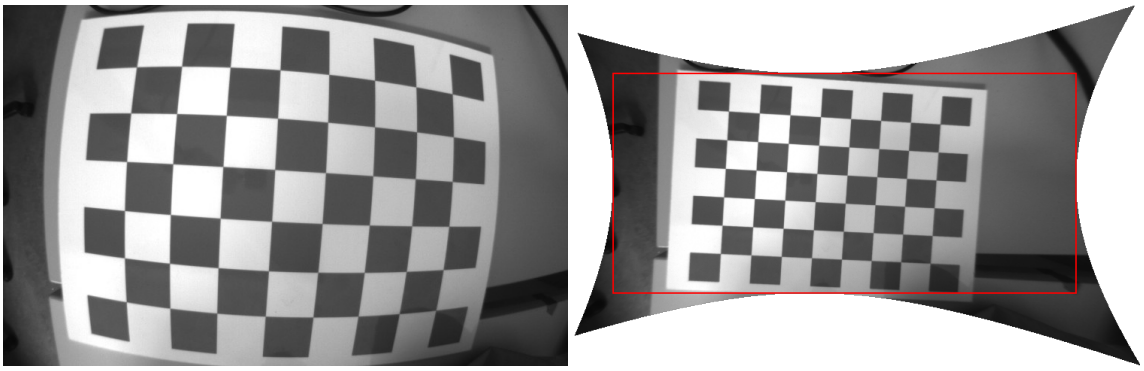


Figure 2.3: **Left:** Photo of a rectangular checkerboard which exhibits strong radial distortion. Lines which are originally straight become curved. **Right:** The same photo after correcting radial distortion. If the image is cropped to the largest rectangular subregion (red), large parts of the original image are cut away.

3D point seen at a given image position in a given distance. These operations are derived here.

Digital raster images are most often represented in a coordinate system with the origin at the top-left image corner, with the x axis pointing to the right and the y axis pointing downwards. To fit to this, the 3D **camera coordinate system** is defined such that the x axis points to the right, the y axis downwards, and the z axis forwards in the view direction to complete a right-handed system. The origin is defined to lie on the camera center. For the following, it is assumed that 3D points are in the camera coordinate system.

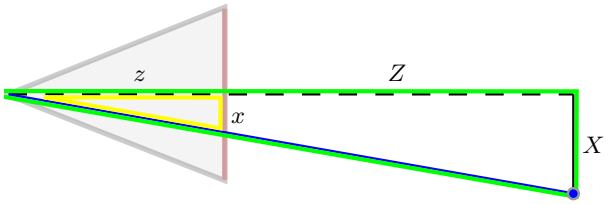


Figure 2.4: Sketch showing how the blue point at $(X, Z)^T$ is projected onto the image plane at $(x, z)^T$. The intercept theorem or, equivalently, the concept of similar triangles is applied to the triangles marked in green and yellow to determine x.

Projecting a point $\mathbf{X} = (X, Y, Z)^T$ onto the image plane to a point $\mathbf{x} = (x, y, z)^T$ can be described by two similar triangles (see Fig. 2.4 for a two-dimensional sketch), relating camera coordinates and image coordinates:

$$\begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix} = \begin{pmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{pmatrix} \tag{2.3}$$

The image plane distance to the camera center is defined to be 1, resulting in:

$$\mathbf{x} := \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{pmatrix} \tag{2.4}$$

$\mathbf{x}$ is now said to be in **normalized image coordinates**, which are independent of the used sensor parameters and have their origin at the principal point. To further determine the position in pixels $\mathbf{x}'$, the sensor's pixel size and configuration needs to be accounted for by first multiplying with a conversion factor and then adding an offset to shift the origin to the top-left corner:

$$\mathbf{x}' = \begin{pmatrix} f_x x + c_x \\ f_y y + c_y \end{pmatrix} \tag{2.5}$$

f_x and f_y can be interpreted as the focal length divided by the pixel width respectively height. c_x and c_y can be interpreted as the pixel coordinates of the principal point (which may be different from the image center).

Points outside the view frustum can be discarded by checking if their projected pixel coordinates lie withing the image bounds. 3D points *behind* the camera may also be calculated to be inside the image and therefore must be checked for in addition by looking at the sign of their Z coordinate in camera coordinates.

In order to be able to write above equation in matrix form, homogeneous coordinates are used to implement the projection and define the **intrinsic calibration matrix $\mathbf{K}$** as follows:

$$\mathbf{x}' = \mathbf{K}\mathbf{x} := \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \triangleq \mathbf{K}\mathbf{X} \tag{2.6}$$

If required, $\mathbf{K}_{1,2}$ may hold the **skew s** as additional parameter. However, for digital cameras this kind of distortion usually does not occur, so the skew is set to zero.

$\mathbf{K}$ as the intrinsic calibration matrix encapsulates all constant intrinsic camera parameters which can be represented by a matrix. They are usually calibrated in a setup step before running the actual system, see Sec. 5.2.[1] (Note that for cheap cameras, the intrinsic parameters may change, e.g. due to temperature differences.) The final projection function $\pi(\mathbf{X})$ is thus a matrix multiplication, followed by conversion to non-homogeneous coordinates:

$$\pi(\mathbf{X}) := \mathbf{K}\mathbf{X} \tag{2.7}$$

2.2.3 Un-projection

The process of determining the 3D point coordinates which are projected to a given pixel position is called un-projection. Using the intrinsics matrix $\mathbf{K}$, homogeneous pixel coordinates $\mathbf{x}'$ are first easily transformed to normalized image coordinates $\mathbf{x}$ by inversion:

$$\mathbf{x} = \mathbf{K}^{-1}\mathbf{x}' \tag{2.8}$$

$$\text{with} \quad \mathbf{K}^{-1} = \begin{pmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

[1] For completeness: in contrast to this, the so-called **extrinsic camera parameters** may change at any time and consist of the camera position and orientation.

As second step, eq. 2.4 is inverted to find the camera space coordinates X and Y. This step requires Z to be given. Assuming that $z = 1$:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} xZ \\ yZ \end{pmatrix} \tag{2.10}$$

The final un-projection function $\pi^{-1}(\mathbf{x}, Z)$ thus depends on the pixel coordinates and a known depth value:

$$\pi^{-1}(\mathbf{x}, Z) := Z\mathbf{K}^{-1}\mathbf{x} \tag{2.11}$$

2.3 SE(3) and Sim(3) Lie groups

Up to now, 3D points were assumed to be in camera space coordinates. However, in practice many different coordinate systems are in use and points need to be transformed between systems, e.g. from a global world coordinate system into a camera coordinate system. For this, a way to represent coordinate system transformations is required. In addition to rigid body transformations which are able to describe camera poses with position and rotation, similarity transformations incorporating scaling are also required in order to cope with scale drift.

While there exist many different representations for the rotational part, for optimizing over the set of transformations as done in SLAM systems a minimal representation (with no more parameters than degrees of freedom) should be chosen. This prevents optimization from creating invalid configurations or performing inefficiently and minimizes the computational load when calculating and computing with Jacobians.

One kind of representation which fulfills this is the SE(3) Lie group and algebra for rigid body transforms and the Sim(3) Lie group and algebra for similarity transforms. Here, first Lie groups and algebrae in general will be explained shortly under a practical point of view, followed by the specific SE(3) and Sim(3) groups. A more thorough explanation, on which this is based on, may be found in [47].

2.3.1 Lie groups and algebrae

Lie groups are defined as mathematical groups which are at the same time a smooth manifold, i.e. a manifold which can be locally approximated with an Euclidean tangent space. A simple and easy to visualize example for such a manifold which is also a Lie group is the circle group SO(2). It represents rotations in a plane and can be visualized as a 2-dimensional sphere. Locally, the circle can be approximated by a line, while globally it is not Euclidean; see Fig. 2.5. However, keep in mind that the circle group is a special example of a Lie group because it is commutative, while Lie groups in general are not.

While formally introducing Lie groups based on above definition is difficult, restricting oneself to matrix Lie groups eases the process while not excluding most interesting Lie groups (according to [47]). The most general matrix Lie group is the group of all invertible matrices $n \times n$, the general linear group GL(n), as it is the largest subset of matrices fulfilling the group axioms; on why it is a smooth manifold, see pages 19f. of [34].

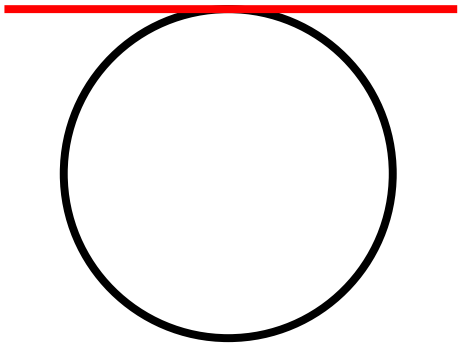


Figure 2.5: Illustration of the circle lie group SO(2) as the set of points covered by applying all possible elements of the group to a point (black). In addition, a part of the tangent space of the topmost circle point obtained by applying $R(0)+\frac{\delta R(t)}{\delta t}\Big|_{t=0} t$ to this point is shown in red. Locally, the tangent space smoothly approximates the circle.

The circle group may be represented in matrix form as the following set of matrices with $t \in \mathbb{R}$:

$$R(t) := \begin{pmatrix} \cos(t) & -\sin(t) \\ \sin(t) & \cos(t) \end{pmatrix} \tag{2.12}$$

For introducing the tangent space (or Lie algebra) of a Lie group, first the concept of smooth paths is introduced. A smooth path in $X \subseteq \mathbb{R}^{n\times n}$ is defined as a differentiable function $P : [a, b] \to X$ with $[a, b]$ being a real interval. Note that here and in the following, matrices in $\mathbb{R}^{n\times n}$ are also equivalently seen as vectors in $\mathbb{R}^{n^2}$ by reordering the components. For the example of the circle group, Eq. 2.12 also defines a smooth path for any interval $[a, b]$, $a < b$. Now a tangent vector $\mathbf{x}$ at a point $\mathbf{y}$ of a smooth path is defined as:

$$\mathbf{x} := \frac{\delta P(t)}{\delta t}\bigg|_{t=0} \quad \text{with} \quad P(0) = \mathbf{y} \tag{2.13}$$

Such a vector is also said to be a tangent vector of the space X. The set of all existing tangent vectors at a point defines a vector space and is called the tangent space of this point. For a Lie group G, the tangent space *at the identity* is called the **Lie algebra** corresponding to the group, denoted (in Gothic letters) $\mathfrak{g}$. For the example of the circle group, the (single) tangent vector at the identity is:

$$\frac{\delta R(t)}{\delta t}\bigg|_{t=0} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{2.14}$$

and thus it also one of the possible single vectors acting as generators of the tangent space there.

2.3.2 Minimal representation

While the matrix representation of tangent vectors is convenient, it is in general overparametrized. For example, the circle group has only one degree of freedom, but the tangent space is a subset of $\mathbb{R}^{2\times 2}$. Thus a minimal representation of tangent space vectors as

a linear combination of a tuple of n generating tangent vectors $\mathbf{G}_i$, $i \in [1..n]$, is defined, together with a mapping between minimal ($\mathbf{x}'$) and normal ($\mathbf{x}$) representation by the **hat** ($\widehat{\cdot}$) and **vee** ($\cdot^\vee$) operators:

$$\widehat{\mathbf{x}'} := \sum_{i=1}^{n} x_i \mathbf{G}_i = \mathbf{x} \tag{2.15}$$

$$\mathbf{x}^\vee := \mathbf{x}' \quad \text{with} \quad \widehat{\mathbf{x}'} = \mathbf{x} \tag{2.16}$$

For the example of the circle group, the operators resolve to the following:

$$\widehat{x} = \begin{pmatrix} 0 & -x \\ x & 0 \end{pmatrix} \tag{2.17}$$

$$\mathbf{x}'^\vee = x'_{2,1} = -x'_{1,2} = \frac{1}{2}(x'_{2,1} - x'_{1,2}) \tag{2.18}$$

2.3.3 Mapping between Lie group and algebra

To associate elements of the Lie algebra with elements from the Lie group and vice versa, the **exponential map** and **matrix logarithm** are used, respectively. The exponential map for matrices is defined analogously to the series representation of the exponential:

$$\exp(\mathbf{X}) = \sum_{k=0}^{\infty} \frac{\mathbf{X}^k}{k!} \tag{2.19}$$

As can be shown [47], it maps the set of matrices to the set of invertible matrices. At the same time, it maps a Lie algebra to its corresponding Lie group. The inverse operation to this is the matrix logarithm, mapping elements of a Lie group to the corresponding Lie algebra. For the example of the circle group, evaluating above series gives:

$$\exp(\widehat{x}) := \begin{pmatrix} \cos(x) & -\sin(x) \\ \sin(x) & \cos(x) \end{pmatrix} \tag{2.20}$$

There are infinitely many valid logarithm values to this because of the wrapping-around of rotation angles. By limiting the codomain in order to obtain a single-valued function, it may be defined for example as follows:

$$\log(\mathbf{X}) = \arctan\left(\frac{\mathbf{X}_{2,1}}{\mathbf{X}_{1,1}}\right) \cdot \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{2.21}$$

2.3.4 Adjoint

Sometimes it is necessary to transform tangent vectors between different tangent spaces. This can be done using the **adjoint** of a Lie group element $\mathbf{A}$ for a tangent space element $\mathbf{V}$, which is defined as [47]:

$$\mathbf{Adj}_\mathbf{A}(\mathbf{V}) := \mathbf{AVA}^{-1} \tag{2.22}$$

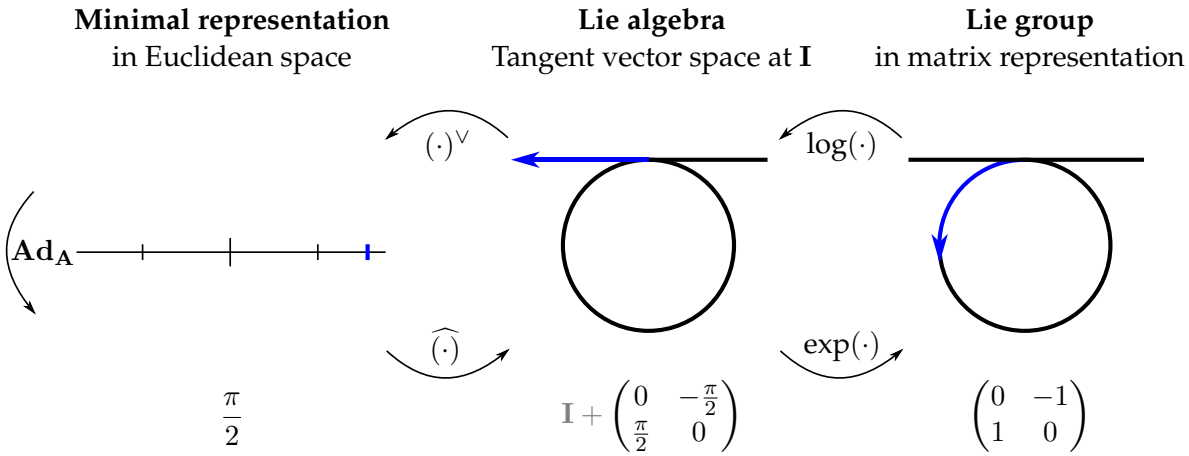


Figure 2.6: Relationship between Lie groups, Lie algebrae and their minimal representation in Euclidean space, with an example from the $SO(2)$ group. Multiplication with the adjoint $\mathbf{Ad_A}$ is used to change the tangent space of a minimal algebra element.

As this is a linear operator, there exists a matrix $\mathbf{Ad_A}$ such that, for all minimal Lie algebra elements $\mathbf{x}$:

$$\widehat{\mathbf{Ad_A x}} = \mathbf{A}\widehat{\mathbf{x}}\mathbf{A}^{-1} \tag{2.23}$$

Its important property is:

$$\mathbf{A}\exp(\widehat{\mathbf{x}}) = \exp(\widehat{\mathbf{Ad_A x}})\mathbf{A} \tag{2.24}$$

Thus, in calculating the composition of $\mathbf{A}$ and $\mathbf{x}$, the multiplication order can be switched:

$$\log(\mathbf{A}\exp(\widehat{\mathbf{x}})) = \log(\exp(\widehat{\mathbf{Ad_A x}})\mathbf{A}) \tag{2.25}$$

Fig. 2.6 shows an overview of the Lie group concept together with its Lie algebra (tangent space), minimal tangent space representation and the operations to convert between the different representations. In practice, for implementation it is convenient and effective to combine hat and exp, respectively log and vee into single operations $\exp(\text{hat}(\cdot))$ and $\text{vee}(\log(\cdot))$.

2.3.5 SE(3) Lie group and algebra $\mathfrak{se}(3)$

The special Euclidean group $SE(3)$ is the group of rigid body transformations consisting of a rotation and a translation. Group elements can be represented as 4×4 matrices consisting of a rotational part and a translational part, which are left-multiplied onto homogeneous vectors for applying the transformation:

$$\begin{pmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_{3\times1} \\ 1 \end{pmatrix} \tag{2.26}$$

In effect, this first rotates the vector $\mathbf{x}_{3\times1}$ around the origin and then applies a translation. Fig. 2.7 shows how this can be seen as change of coordinate system or transformation of the scene.

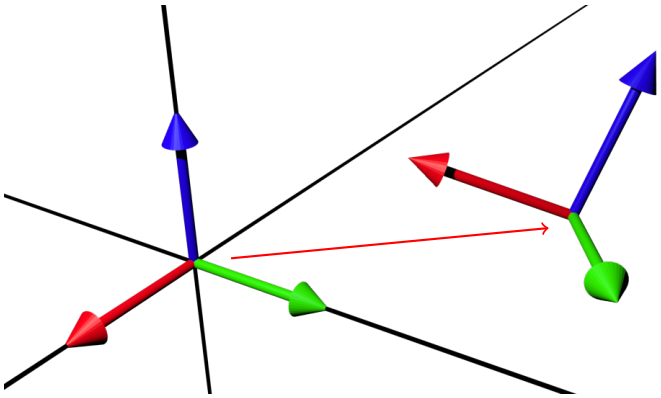


Figure 2.7: Illustration of a rigid body transformation as a change of coordinate system with fixed scene. The original and resulting coordinate systems are displayed as three orthogonal axes. Equivalently, rigid body transformations can also be thought of as transforming the scene with fixed coordinate system. The transformation changing the system A to B is equal to the transformation moving the scene from B to A in system B. Transformations can be easily concatenated by matrix multiplication to transform between more than two systems.

Rigid body motions have 6 degrees of freedom (3 in rotation and 3 in translation), thus elements of $\mathrm{SE}(3)$ map to a 6-dimensional minimal tangent space vector in $\mathfrak{se}(3)$. Formulas for the hat-exp and log-vee operations are derived in [47] and given here for reference. Note that it is wrong to assume in general that the rotation and translation components are stored separately in the minimal tangent space vector. With the minimal tangent space vector defined as:

$$\boldsymbol{\xi}_{6\times 1} \in \mathfrak{se}(3), \quad \begin{pmatrix} \boldsymbol{\upsilon}_{3\times 1} \\ \boldsymbol{\omega}_{3\times 1} \end{pmatrix} := \boldsymbol{\xi} \tag{2.27}$$

The hat-exp operation is defined as:

$$\exp(\widehat{\boldsymbol{\xi}}) = \begin{pmatrix} \exp([\boldsymbol{\omega}]_\times) & \mathbf{V}\boldsymbol{\upsilon} \\ \mathbf{0} & 1 \end{pmatrix} \tag{2.28}$$

$$\text{with} \quad \left[\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right]_\times = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}, \tag{2.29}$$

$$\mathbf{V} = \begin{cases} \mathbf{I} + \frac{1}{2}[\boldsymbol{\omega}]_\times + \frac{1}{6}[\boldsymbol{\omega}]_\times^2 & \text{for } \theta \to 0 \\ \mathbf{I} + \frac{1-\cos(\theta)}{\theta^2}[\boldsymbol{\omega}]_\times + \frac{\theta-\sin(\theta)}{\theta^3}[\boldsymbol{\omega}]_\times^2 & \text{else} \end{cases} \text{with } \theta = ||\boldsymbol{\omega}||_2, \tag{2.30}$$

$$\exp([\boldsymbol{\omega}]_\times) = \begin{cases} \mathbf{I} + [\boldsymbol{\omega}]_\times + \frac{1}{2}[\boldsymbol{\omega}]_\times^2 & \text{for } \theta \to 0 \\ \mathbf{I} + \frac{\sin(\theta)}{\theta}[\boldsymbol{\omega}]_\times + \frac{1-\cos(\theta)}{\theta^2}[\boldsymbol{\omega}]_\times^2 & \text{else} \end{cases} \text{with } \theta = ||\boldsymbol{\omega}||_2 \tag{2.31}$$

The log-vee operation is defined as:

$$\log\left(\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}\right)^\vee = \begin{pmatrix} \mathbf{V}^{-1}\mathbf{t} \\ \log(\mathbf{R})^\vee \end{pmatrix} \tag{2.32}$$

$$\text{with} \quad \log(\mathbf{R}) = \begin{cases} \frac{1}{2}(\mathbf{R} - \mathbf{R}^T) & \text{for } d \to 1 \\ \frac{\arccos(d)}{2\sqrt{1-d^2}}(\mathbf{R} - \mathbf{R}^T) & \text{for } d \in (-1, 1) \end{cases} \quad \text{with } d = \frac{1}{2}(\text{trace}(\mathbf{R}) - 1), \tag{2.33}$$

$$(\boldsymbol{\Omega})^{\vee}_{\mathfrak{so}(3)} = \frac{1}{2} \begin{pmatrix} \boldsymbol{\Omega}_{3,2} - \boldsymbol{\Omega}_{2,3} \\ \boldsymbol{\Omega}_{1,3} - \boldsymbol{\Omega}_{3,1} \\ \boldsymbol{\Omega}_{2,1} - \boldsymbol{\Omega}_{1,2} \end{pmatrix} \tag{2.34}$$

2.3.6 Sim(3) Lie group and algebra $\mathfrak{sim}(3)$

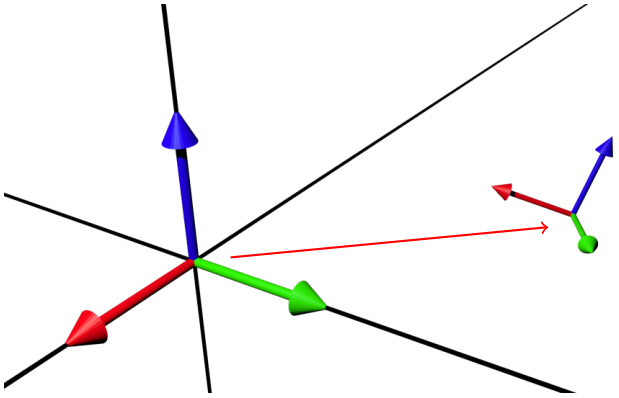


Figure 2.8: Analogous figure to 2.7 for Sim(3). The scale change is illustrated by a differing size of the coordinate system frame. Sim(3) transformations can be used analogously to SE(3) transformations for changing between different coordinate systems or transforming the scene.

For coping with scale drift in monocular SLAM, rigid body motions are not sufficient, as the estimated scale of the whole scene may vary. For this reason, the group of similarity transformations Sim(3) in three dimensions is also introduced which in addition allows to include an isotropic scaling. Elements of this group may be represented by 4×4 matrices consisting of a rotational part, including positive scaling $s \in \mathbb{R}^+$, and a translational part. As for the SE(3) group, for applying the transformation the matrix is left-multiplied onto a homogeneous vector:

$$\begin{pmatrix} s\mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_{3\times1} \\ 1 \end{pmatrix} \tag{2.35}$$

In effect, this first uniformly scales and rotates the vector $\mathbf{x}_{3\times1}$ around the origin and then applies a translation. Fig. 2.8 illustrates this kind of transformation.

Scaling introduces another degree of freedom over rigid body motions, so the minimal tangent space vectors in the Lie algebra $\mathfrak{sim}(3)$ have 7 components. Formulas for the hat-exp and log-vee operations are derived in [47] and given here for reference. With the minimal tangent space vector defined as:

$$\boldsymbol{\xi}_{7\times1} \in \mathfrak{sim}(3), \quad \begin{pmatrix} \boldsymbol{\upsilon}_{3\times1} \\ \boldsymbol{\omega}_{3\times1} \\ \sigma \end{pmatrix} := \boldsymbol{\xi} \tag{2.36}$$

The hat-exp operation is defined as:

$$\exp(\widehat{\boldsymbol{\xi}}) = \begin{pmatrix} e^{\sigma}\exp([\boldsymbol{\omega}]_{\times}) & \mathbf{W}\boldsymbol{\upsilon} \\ \mathbf{0} & 1 \end{pmatrix} \tag{2.37}$$

$$\text{with} \quad \mathbf{W} = \left(\frac{e^{\sigma}-1}{\sigma}\right)\mathbf{I} + \frac{A\sigma + (1-B)\theta}{\theta(\sigma^2+\theta^2)}[\boldsymbol{\omega}]_{\times} + \left(\frac{e^{\sigma}-1}{\sigma} - \frac{(B-1)\sigma + A\theta}{\sigma^2+\theta^2}\right)\frac{[\boldsymbol{\omega}]_{\times}^2}{\theta^2}, \tag{2.38}$$

$$A = e^{\sigma}\sin(\theta), \quad B = e^{\sigma}\cos(\theta), \quad \theta = ||\boldsymbol{\omega}||_2 \tag{2.39}$$

The log-vee operation is defined as:

$$\log\left(\begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}\right)^{\vee} = \begin{pmatrix} \mathbf{W}^{-1}\mathbf{t} \\ \log(\mathbf{R})^{\vee} \\ \ln(s) \end{pmatrix} \tag{2.40}$$

2.4 Non-linear optimization

Many tasks in computer vision involve defining a **cost function** or **energy** depending on a number of parameters and optimizing the parameter values to find a minimum (or equivalently, a maximum) of the function. For example, for registering some images taken from the same viewport to each other to form a panorama, the parameters would define the relative placement of images and the cost function would measure the similarity of overlapping image regions. Cost functions are often defined as a sum of squared residuals r_i. Here, $\mathbf{p}$ denotes the vector of parameters:

$$E(\mathbf{p}) := \sum_i r_i(\mathbf{p})^2 \tag{2.41}$$

For example, in image registration a summation over all pixels of an image may be used. In case the residuals are defined to be linear in the parameters to optimize, this would be a **linear least squares** problem for which a closed-form solution could be computed. However, often they are not; for example, accessing an image at a parameterized position is always non-linear for a general image. For these **non-linear least squares** problems, solutions are usually found iteratively. Starting from an initial parameter estimate and repeatedly computing a linearized approximation to the problem, small update steps are calculated for the parameters. If the initial estimate is close to the solution, the parameter estimate will then converge towards it; if the initial estimate is too far away and the function is non-convex, it will diverge and no optimum or a local optimum may be found instead. Convergence may be checked for e.g. by thresholding the relative reduction in cost. The following subsections present two well-known algorithms for calculating update steps in non-linear least squares problems.

2.4.1 Gauss-Newton method

This method approximates in each step k the residuals with a first-order Taylor approximation at the current parameter estimate $\mathbf{p}_k$ to find a parameter update $\Delta\mathbf{p}$:

$$r_i(\mathbf{p}_k + \Delta\mathbf{p}) \approx r_i(\mathbf{p}_k) + \left.\frac{\partial r_i}{\partial \mathbf{p}}\right|_{\mathbf{p}_k} \Delta\mathbf{p} \tag{2.42}$$

This is a linear expression in $\Delta\mathbf{p}$, so by setting in into the cost function, a linear least squares problem is created which is solved to find $\Delta\mathbf{p}$:

$$E(\mathbf{p}_k + \Delta\mathbf{p}) \approx \sum_i \left(r_i(\mathbf{p}_k) + \left.\frac{\partial r_i}{\partial \mathbf{p}}\right|_{\mathbf{p}_k} \Delta\mathbf{p} \right)^2 \tag{2.43}$$

Stacking these terms as matrices yields:

$$E(\mathbf{p}_k + \Delta\mathbf{p}) \approx ||\mathbf{y}_k - \mathbf{J}_k \Delta\mathbf{p}||^2 \tag{2.44}$$

Differentiation by $\Delta\mathbf{p}$ and setting to zero to find the optimum finally results in:

$$\Delta\mathbf{p} = -(\mathbf{J}_k^T \mathbf{J}_k)^{-1} \mathbf{J}_k^T \mathbf{y}_k \tag{2.45}$$

2.4.2 Levenberg-Marquart method

The Gauss-Newton method uses a second-order approximation to the cost function to compute update steps. Here, $\mathbf{J}_k^T \mathbf{J}_k$ is the Gauss-Newton approximation to the Hessian matrix, the matrix of second derivatives. If the real cost function is not well enough modeled by this approximation, divergence may result.

The Levenberg-Marquart method is an alternate optimization method which solves this problem by smoothly blending between the Gauss-Newton method and gradient descent (updating parameters in the direction of highest cost reduction). It replaces the used Hessian approximation by $\mathbf{J}_k^T \mathbf{J}_k + \lambda \mathbf{I}$, $\mathbf{I}$ being the identity matrix. λ is an adaptive scalar parameter which controls the update step: if it is close to zero, the updates are essentially equal to Gauss-Newton updates as derived above. For larger values of λ the updates converge to gradient descent steps, directly updating the parameters in the direction of largest residual reduction.

Before applying an iteration, the Levenberg-Marquardt algorithm checks whether the corresponding update decreases the residual. If yes, the update is accepted and λ decreased; if no, the update is rejected and λ increased. For example, λ is multiplied by or divided by 10. This effectively prevents doing Gauss-Newton steps which increase the residual and falls back to the gradient descent method for these cases.

2.5 Uncertainty propagation

Uncertainty propagation is the process of estimating the covariance of a distribution after being transformed by a function. It is often required when working with distribution covariances, as for example used to represent uncertain estimates in SLAM systems, to transform them into a common frame. If for example $\mathbf{x}$ is transformed by $f(\mathbf{x})$, one wants to determine the covariance of $f(\mathbf{x})$ from the covariance of $\mathbf{x}$, denoted $\mathbf{\Sigma}_\mathbf{x}$.

An easy way to approximate this is using first-order Taylor approximation. Then the estimate is calculated as follows, with $\mathbf{J}_f$ being the Jacobian of f at $\mathbf{x}$:

$$\mathbf{\Sigma}_f \approx \mathbf{J}_f \mathbf{\Sigma}_\mathbf{x} \mathbf{J}_f^T, \tag{2.46}$$

There are other alternatives which may be better suited for non-linear transformations such as the unscented transform [57], which however require more processing time.

2.6 Epipolar geometry

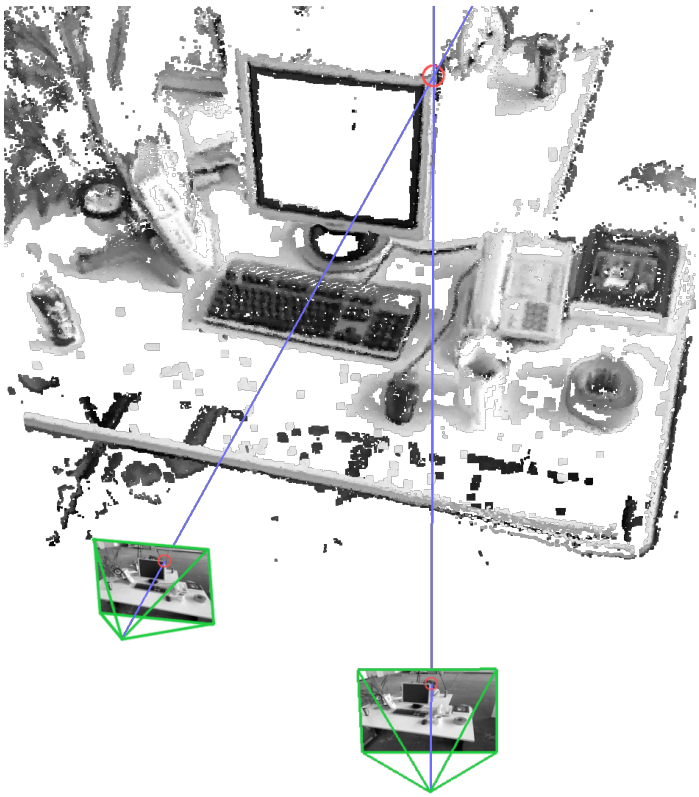


Figure 2.9: Principle of depth estimation by stereo comparisons: in two or more images, the projected pixel coordinates of the same 3D location are found. Here it is a corner of the computer monitor (circled red). Then the 3D position is determined by intersecting the rays through the camera center and found pixel coordinates in the image plane (blue).

The process of estimating depth from color or grayscale images utilizes the geometrical relations, called **epipolar geometry**, between two or more pinhole-model cameras. For the scope of this thesis, only **stereo comparisons** are of interest, i.e. the way depth is estimated in RGB or grayscale images by finding identical image regions.

This is shown in Fig. 2.9: the ray originating from the camera center and passing through a point on the image plane is the set of all possible world locations of this point. To estimate the depth of the point along this ray, the projected point coordinates of the same world location in at least one other camera image are found. By intersecting the resulting rays, a unique 3D location estimate for the point is determined.

In practice the rays will not intersect exactly, so the point with minimum distance to both lines can be used as an average. An approach to limit the search space is to project the line of all possible point locations determined from one camera into the other camera image. This line is then called an **epipolar line** corresponding to the point in the first image. By constraining the search for the same world location in the second image to the

epipolar line, it is guaranteed that the rays to find the point's world position will intersect. However, the assumes that the relative placement of the cameras is correctly estimated – if not, the correct location might be outside the search range.

3 Related work

This chapter presents an overview of existing monocular SLAM and visual odometry approaches with focus on methods which run in real-time on mobile devices.

3.1 Feature-based methods

Common to all feature-based methods is a two-step processing pipeline:

1. **Feature extraction:** First, a feature extraction component scans the image – typically with no prior information – and determines a set of feature observations. As a feature is found, it calculates a descriptor vector for it based on its appearance.

2. **Geometry calculation:** With fast matching methods and outlier detection schemes like RANSAC, the feature observations are matched to each other between different images. From this derived information only, camera poses and scene geometry are estimated. There exists a variety of methods to do this, including bundle adjustment based approaches such as the popular PTAM [29] or filtering-based approaches [11, 36].

Doing sequential image-based (photometric) estimation as first and geometric estimation as second step simplifies the problem. However, *it limits the system to using information contained in the chosen feature types*. Most approaches use keypoints as their feature type; in this case, information contained in edges (which are common in man-made environments) is lost.

It has been attempted to model such observations as features too, in particular [30, 13] describe edge-based features while [8] uses region-based features. As estimation of such high-dimensional feature spaces is difficult, they are rarely used in practice.

Keypoint-based approaches operate on a sparse representation of the scene consisting of the keypoint cloud. One often wants to estimate a denser 3D reconstruction, so many approaches have been made where an additional reconstruction step is added to a keypoint-based system, operating on the raw images and camera poses returned by the system [51]. However, the information obtained this way cannot be directly used in the SLAM system.

Today, there are several keypoint-based monocular visual odometry and SLAM methods which run in real-time on mobile devices [36, 31].

3.2 Direct methods

Instead of estimating features, direct methods track camera poses and create depth maps directly on the raw images. *This way all image information may be used*, which leads to higher accuracy and robustness. Indoor environments with few keypoints particularly

benefit from this; in addition, it provides significantly more information about the scene geometry. Image alignment is usually done based on the Lucas-Kanade method [3].

Earlier direct methods use planar patches to model the scene. [46] presents such an approach for monocular cameras which simultaneously estimates the correspondences, camera pose, scene structure and illumination changes. [25] also relies on planar patches and especially discusses issues of local optimums during initialization.

New methods use per-pixel depth estimates. This is well established for RGB-D or stereo sensors [27, 7]. For monocular color or grayscale cameras, these depth estimates need to be determined by stereo comparisons with previous video frames. Depending on the density of the estimated depth maps, the methods can be divided into two classes:

- **Dense methods:** These compute fully dense depth maps for selected keyframes with variational regularization. The popular DTAM system [39] is one of the examples using this method, others are [49, 41]. A disadvantage of this is its computational demand which requires to use a powerful GPU to run it in real-time.
- **Semi-dense methods:** These are motivated by the goal to reduce processing complexity, and merge depth observations iteratively into distribution estimates with filtering. [16], on which this thesis is based on, creates semi-dense depth maps in this way in real-time on a CPU. To achieve high frame rates even on embedded platforms, [19] combines direct tracking with keypoints.

These methods are all visual odometries, and to our knowledge this is true for all existing direct monocular methods.

4 Monocular Semi-Dense SLAM

This chapter describes the developed system. First, Sec. 4.1 presents an overview of the whole system to give an idea of the main concepts, components and their dependencies. Following this, each of the components is described in detail in Sec. 4.2 to Sec. 4.6.

4.1 SLAM system overview

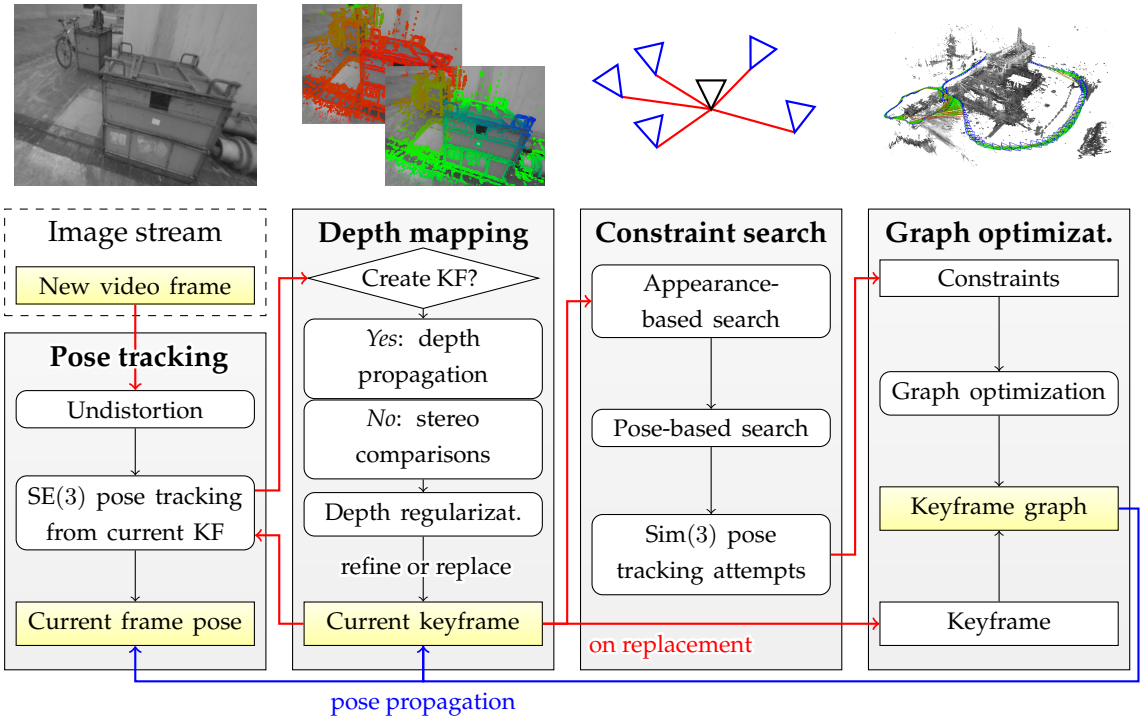


Figure 4.1: Overview of the system components and control flow. New frames are first processed by the **pose tracking** component to determine the camera pose using the current keyframe as reference frame. Based on the tracking result, the **depth mapping** component decides whether to create a new keyframe from this frame or not and replaces or refines the current keyframe with the new frame, respectively. When the current keyframe is replaced, the old keyframe is passed on to the **constraint search** component which determines relative poses to other keyframes. Finally, constraints and old keyframe are passed on to the **graph optimization** component which inserts them into the keyframe graph and optimizes it. The optimized poses propagate back to the active frames (blue).

This section describes the overall architecture of the developed SLAM system. Fig. 4.1 shows a graphical overview of its components. The system is based on the visual odometry system of [16], which provided the initial implementation of the tracking and mapping components; especially the latter is taken over mostly unmodified.

4.1.1 Concepts

(Key)frame

A (video) frame is one image from the input image stream. Here, all derived information is also considered to belong to a frame, such as its pose or depth map. Some frames are promoted to keyframes, which stay in memory after their initial processing and are later inserted into the keyframe graph.

Pose

This term is used for the $\mathrm{SE}(3)$ or $\mathrm{Sim}(3)$ transformation between a frame and the world coordinate system, giving the frame's position, orientation and possibly scale.

Semi-dense inverse depth map

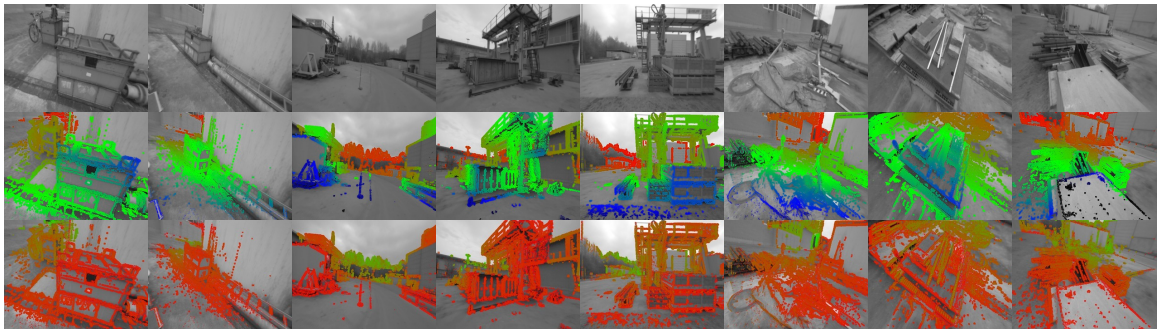


Figure 4.2: **1st row:** example frames from a video sequence. **2nd row:** corresponding estimated semi-dense inverse depth maps. **3rd row:** corresponding estimated inverse depth variance maps.

This is the way depth estimates are stored for (key)frames: for each pixel, there is an inverse depth hypothesis consisting of the *inverse* depth mean and variance estimates of a Gaussian distribution. The inverse representation is due to the variance of estimates determined by stereo comparisons, which can be far better represented in inverse depth space than in actual depth.

These depth maps only cover image regions with sufficient intensity gradient. This is for the reason that the remaining, homogeneous regions are unsuitable for stereo comparisons, as there is few or no image information there which can be used to find the same world location in another image. Furthermore, not processing depth estimates in these regions accelerates the system.

Visualization of depth maps is done by overlaying a grayscale version of the camera image with rainbow-colored depth estimates: from close to far away pixels the color gradient is blue-green-red. See Fig. 4.2 for some examples.

Keyframe graph

This is the graph with keyframes as vertices and relative keyframe poses as edges, used to do global optimization. It is also commonly referred to as pose graph.

4.1.2 Initialization

For tracking camera poses by direct image alignment, a frame with corresponding inverse depth and variance map is required. Thus these maps needs to be estimated for one initial video frame to bootstrap the system. There are various ways for doing this:

- **Keypoint-based initialization** uses a system similar to the well-known PTAM [29] based on detecting and matching **keypoints** in image pairs, i.e. distinctive, well localizable corner points or blobs (circular regions of the same color), to estimate the images' relative pose. If a transformation between the images was estimated which is suitable for stereo comparisons, for which enough translation must be present, the current frame is taken as start frame with its depth map estimated by stereo comparisons to the first frame. More details on keypoint tracking for this type of initialization are given in Sec. 4.2.1.
- **Depth or stereo camera initialization** is possible if the camera being used is a depth camera such as the Microsoft Kinect [54], or a stereo camera, by directly taking the depth map given by the camera or estimated by stereo comparisons. This is mainly interesting for benchmarks as a reliable and deterministic choice for the initial depth map, in addition providing the system with the absolute scene scale.
- **Direct initialization** works by setting the whole depth map of the first frame to a random depth value and setting the depth variances very high. In effect this specifies that the depth is unknown by making the probability distributions span the whole possible range. The running system subsequently converges into a consistent state. However, care must be taken to prevent convergence into an invalid state; see ideas for future work in Sec. 8.1.

4.1.3 Pose tracking

As soon as the first depth map for a camera frame is available, this frame is used as a tracking reference for estimating the pose of subsequent frames. Pose estimation is iterated by the tracking component for each new video frame as quickly as possible.

Tracking works by direct image alignment: the point cloud defined by the tracking reference's image and depth map is first projected to a virtual image from another viewpoint. The transformation used for this viewpoint is then optimized to minimize the difference between the virtual image and new camera image, giving the estimate for the real camera pose when converged.

In this thesis, images are represented as functions mapping pixel coordinates to the image value at this point. Stated mathematically, with I_{ref} being the reference image, D_{ref} the reference depth map, I the new image, ω the warp function and $\mathbf{p}_i$ pixels with valid depth map value, the basic energy term to minimize is depending on the pose $\boldsymbol{\xi}$:

$$E(\boldsymbol{\xi}) = \sum_i \left(I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \boldsymbol{\xi}))\right)^2, \tag{4.1}$$

Details on minimizing this on the $\mathrm{SE}(3)$ and $\mathrm{Sim}(3)$ groups are given in Sec. 4.2. An example residual image is shown in Fig. 4.3 on the right.

Figure 4.3: **Left:** semi-dense depth map of tracking reference image. **Middle:** new image to be tracked. **Right:** residual image in converged state in the reference image view; gray is the optimum. Occlusions are visible as darker or brighter spots, and the difference in field of view can be seen as there are no residuals for the upper and left parts of the image.

4.1.4 Depth mapping

Each new camera frame usually provides more information about the scene. As soon as the pose of a new frame is estimated, the frame is thus used to do stereo comparisons with the current tracking reference to improve and extend its depth map.

For various reasons discussed in Sec. 4.3, at some point it is decided to switch to a new tracking reference. The depth map of the old reference frame is then projected into the new reference frame to initialize its depth map. Parts which are not covered by this initialization are filled in later by subsequent iterations of depth map refinement. At the same time, the old tracking reference frame becomes a keyframe which stays in memory and is later used to build a keyframe graph. Details on the mapping component are given in Sec. 4.4.

4.1.5 Constraint search

When a new keyframe is created, its pose has only been estimated based on the previous tracking reference, which is the previous keyframe. This alone would make for a visual odometry system only, leading to drift. To counteract this, for each new keyframe additional pose estimations are done relative to other existing keyframes. The constraint search component finds suitable keyframe candidates for this which are likely to have sufficient overlap with the new keyframe, and tries to determine their relative pose. Details on this are given in Sec. 4.5.

The relative pose estimates together define a set of pose constraints in the form of a graph with vertices denoting keyframes and edges denoting relative pose estimations: the keyframe graph. For this reason, additional pose constraints as determined by the constraint search component are also called **loop closures**: they create loops in the keyframe graph.

4.1.6 Graph optimization

This component takes the keyframe graph built by the constraint search component and optimizes it to distribute the pose error in graph loops optimally on all involved constraints. A large accumulated error detected by returning to an old position after doing a loop will thus be distributed over the whole loop, correcting the trajectory. Fig. 4.4 gives an example for such a correction step. Details on the graph optimization part are given in Sec. 4.6.

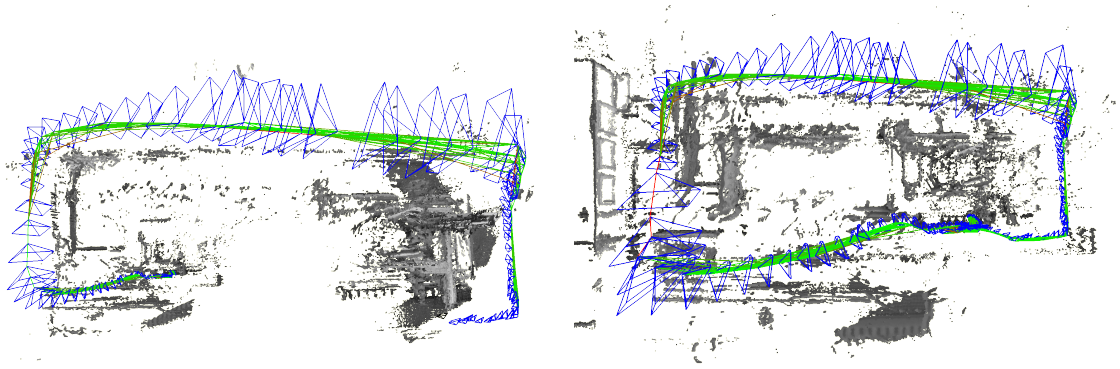


Figure 4.4: Example of a loop closure on returning to a known position. Blue camera frames visualize the pose of keyframes (including scale). Lines between them visualize constraints and their error (increasing from green to red). On the left, the previous state is shown in which some pose error has accumulated. On the right, visiting the known location was recognized and the trajectory corrected with loop closures.

4.2 Pose tracking

Tracking is the process of determining the relative transformation between two camera frames. Historically, in computer vision for a long time feature-based approaches were used for this task. A well-known example of this is the PTAM system [29]. The system presented in this thesis does not use this approach during normal operation, but it is useful as part of the initialization procedure. Thus it is briefly described in Sec. 4.2.1.

With the introduction of consumer-grade depth cameras such as the Kinect, dense per-pixel tracking methods became popular [27, 7] which do whole-image alignment using the Lucas-Kanade method, leading to more robust results. The tracking component of this thesis also uses this method; it is described in general in Sec. 4.2.2 and in particular,

tracking on the $\mathrm{SE}(3)$ and $\mathrm{Sim}(3)$ Lie groups is described in Sec. 4.2.3 and Sec. 4.2.4, respectively.

4.2.1 Feature-based tracking

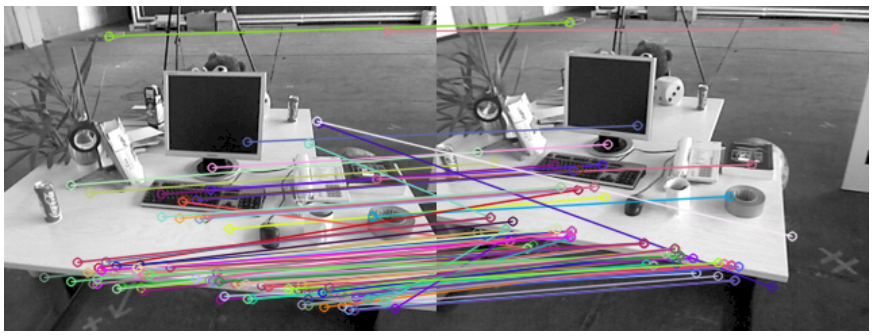


Figure 4.5: Result of keypoint matching between two images, thresholded by match quality. The images are displayed next to each other with the matches shown as colored lines. While most points are matched well, there are some outliers being matched to different locations. In some parts of the images, no good matches are found at all.

The first step in this method is to reduce the images to a set of distinctive, well localizable features. This eases the pose estimation problem and reduces the computational load, but discards information not contained in the features chosen. The most common feature type in use are keypoints. Other feature types have also been evaluated, but are only rarely used [30, 13, 8]. Two popular and fast methods for detecting keypoints are FAST [42] and SURF [4].

In a second step, the detected features are described by a feature vector derived from a local patch of the image around the feature. These vectors allow to evaluate the similarity of a pair of features by computing the difference. SURF [4], BRIEF [6] and ORB [43] are examples of descriptor methods which are suitable for real-time use.

Both the detector and descriptor stages are (with some exceptions, see BRIEF [6]) designed to be as invariant against various transformations as possible: the same set of keypoints and corresponding descriptors is supposed to be found for the same object points projected to the image, before and after camera or object movements and lighting changes. However, this comes at the price of reduced specificity.

The third step in the algorithm is the matcher stage which finds the most likely pairs of features in two images which belong to the same object point. This works by finding nearest neighbors between descriptor vectors in descriptor space, and can be accelerated by methods such as FLANN [38].

Finally, from the set of matches the relative geometry of the images is estimated by means of fundamental matrix estimation [2].

With two images as input only, the transformation can just be estimated up to scale. For constraining the scale, the depth of at least one keypoint being matched must be known before.

To increase the robustness of this method against outliers (which occur frequently due to un-modeled effects such as reflective lighting or moving objects), variations of the RANSAC (random sample consensus) [18] method are often used as part of the last step. Their underlying basic idea is to select a random minimal set of samples (matches) and estimate the model (transformation) from it. Then the numbers of inliers among the remaining samples fitting to the estimated model is counted. This is repeated a number of times, and the estimate with the most inliers chosen as result after refining it with the data of all inliers. Fig. 4.5 shows an example visualization for matched keypoints between two images.

4.2.2 Forwards-additive Lucas-Kanade tracking

Overview

The Lucas-Kanade image alignment algorithm [3] expects as input a template image T and an input image I. A warp function $\omega(\mathbf{x}, \mathbf{T})$ is defined to warp pixel $\mathbf{x}$ in the template coordinate system to the image I with transformation $\mathbf{T}$. The warp can be freely chosen as for example a translation, a homography, or a full 3D rigid body or similarity transformation as is covered in the following sections.

The goal of the algorithm is to minimize the sum of squared differences (SSD) photometric error between the template and the warped image with respect to the warp parameters $\mathbf{T}$ by iterative Least-Squares:

$$\sum_{\mathbf{x}} (I(\omega(\mathbf{x}, \mathbf{T})) - T(\mathbf{x}))^2 \tag{4.2}$$

Starting from an initial transformation estimate $\mathbf{T}_0$, this non-linear optimization is done by the *forwards-additive* algorithm variant by solving for increments $\Delta\boldsymbol{\xi}$ to minimize a first-order Taylor expansion of:

$$\sum_{\mathbf{x}} (I(\omega(\mathbf{x}, \mathbf{T}_i \boxplus \Delta\boldsymbol{\xi})) - T(\mathbf{x}))^2 \tag{4.3}$$

and then updating the current estimate $\mathbf{T}_i$ to $\mathbf{T}_{i+1}$ by composition of parameters denoted $\boxplus$, for example by component-wise addition (but not necessarily, despite the name *forwards-additive*):

$$\mathbf{T}_{i+1} := \mathbf{T}_i \boxplus \Delta\boldsymbol{\xi} \tag{4.4}$$

The parameter estimate and the update are denoted differently here as the update may be calculated in another space as the parameters. This becomes important when optimizing on Lie groups as described later.

Convergence is usually checked by thresholding a norm of the update $\Delta\boldsymbol{\xi}$ and aborting if it is too small, or by checking if the ratio of new and old residual is too large.

Parameter update

Computing update steps is done with a non-linear optimization algorithm as explained in Sec. 2.4. Here, the Gauss-Newton method is applied exemplarily: first order Taylor

expansion of the residual in Eq. 4.3 at the identity update $\Delta\boldsymbol{\xi} = 0$ with application of the chain rule leads to:

$$\sum_{\mathbf{x}} (I(\omega(\mathbf{x}, \mathbf{T}_i)) + \underbrace{\nabla I|_{\omega(\mathbf{x},\mathbf{T}_i)} \left.\frac{\partial \omega}{\partial \mathbf{T}}\right|_{\mathbf{T}_i} \left.\frac{\partial \boxplus(\mathbf{T}_i, \boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}}}_{:=\mathbf{J}_{\mathbf{x}}} \Delta\boldsymbol{\xi} - T(\mathbf{x}))^2 \tag{4.5}$$

in which ∇I denotes the row vector of the image gradient. The minimum can be found by deriving by the warp update $\Delta\boldsymbol{\xi}$ and setting the term to zero, giving:

$$\Delta\boldsymbol{\xi} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \mathbf{J}_{\mathbf{x}}^T (T(\mathbf{x}) - I(\omega(\mathbf{x}, \mathbf{T}_i))) \tag{4.6}$$

in which $\mathbf{H}$ is the Gauss-Newton approximation to the Hessian matrix:

$$\mathbf{H} = \sum_{\mathbf{x}} \mathbf{J}_{\mathbf{x}}^T \mathbf{J}_{\mathbf{x}} =: \mathbf{J}^T \mathbf{J} \tag{4.7}$$

Fig. 4.6 shows a sketch of the update iteration principle.

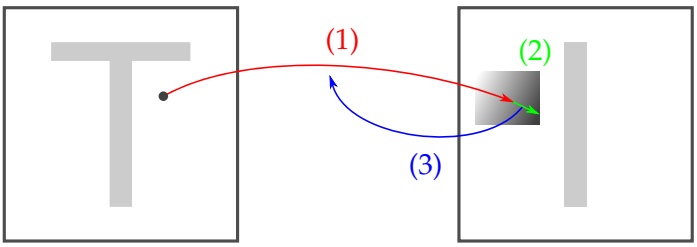


Figure 4.6: Overview of the general Lucas-Kanade method iteration: (1) Each template point is warped with the current transformation estimate to the image. (2) The residual and its jacobian are determined using the image gradient at this point. (3) The transformation estimate is updated with the jacobian accumulated from all sample points.

Covariance estimation

It is often useful to get an estimate of the covariance of the final transformation parameters $\mathbf{T}$. As the approximate Hessian used in the Gauss-Newton method is equivalent to the Fisher information matrix of a multivariate normal distribution [52], an estimate for the covariance can be obtained by inverting it. Note however that this assumes the residuals to be independent, which they are not in the case of the system described in this thesis. Thus the resulting covariance is only a lower bound.

With above definition of the warp composition, the covariance matrix applies to a right-applied error increment onto the real value, i.e. the error is modeled as:

$$\mathbf{T} = \mathbf{T}_{\text{true}} \boxplus \boldsymbol{\epsilon}, \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{\xi}}) \tag{4.8}$$

Note that this is not necessarily interchangeable with a left-application update rule if the operation is not commutative. This needs to be payed attention to if using the covariance matrix e.g. in a graph optimization framework like g2o (which assumes right-multiplication of transformations). As the covariance applies to an increment, it is also given in its space.

Robustification

The approach presented so far assumes that the template stays constant, fully visible in the image and the lighting respectively camera exposure time or gain is also constant. In practice, these assumptions are violated and robustification must be used in order not to let outlier pixels induce a large error resulting from the sum of squared differences norm which gives a big weight to them.

While there exist methods to cope with whole-scene lighting changes (see part 3 of [3]), they are not discussed here as they are not used in the system. This section focuses on robustification against a small number of outlier pixels, as caused by e.g. occlusions or reflective lighting, by means of robust weighting functions.

In order to achieve this, with the photometric residuals used so far named $r_{\mathbf{x}}(\mathbf{T})$, the error norm is extended by individual sample weights to:

$$\sum_{\mathbf{x}} w_{\mathbf{x}}(\mathbf{T}) r_{\mathbf{x}}(\mathbf{T})^2 \tag{4.9}$$

With $\mathbf{W}$ being the weight matrix with weights on its diagonal, the update calculation step then becomes:

$$\Delta\boldsymbol{\xi} = (\underbrace{\mathbf{J}^T\mathbf{W}\mathbf{J}}_{=\mathbf{H}})^{-1} \sum_{\mathbf{x}} \mathbf{J}_{\mathbf{x}}^T w_{\mathbf{x}}(\mathbf{T}_i) r_{\mathbf{x}}(\mathbf{T}_i) \tag{4.10}$$

Fig. 4.7 shows an example tracking weights image. Details on which weight functions are used are given later in the specific $\mathrm{SE}(3)$ and $\mathrm{Sim}(3)$ sections.

Figure 4.7: **Left:** semi-dense depth map of tracking reference image. **Middle:** new image to be tracked. **Right:** weight image in converged state in the reference image view; the brighter, the higher the weight is. The occlusion of the region left to the computer screen is visible as a black region (circled in red).

Multi-level tracking

As direct image alignment optimizes a highly non-convex energy, it may easily get trapped in a local optimum. An important way to increase the chance of convergence is to use a

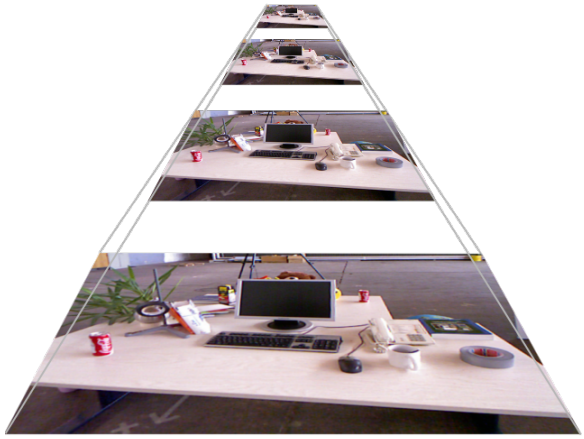


Figure 4.8: Sketch of an image "pyramid" with four levels.

multi-level approach: first an image pyramid is created for the template and new image in which the image resolution is halved in each step. This is sketched in Fig. 4.8. Similarly the semi-dense depth map is downsampled to pyramid levels. Tracking then starts on the highest pyramid level with lowest resolution. After converging on this level, it iteratively continues on the lower level using the pose determined so far as initial estimate. The final pose estimate is calculated on the lowest level using the original image resolution. As convergence is able to start on higher pyramid levels, local optima on lower levels are avoided, drastically increasing the convergence radius. Fig. 7.1 in the results section shows this by example.

Efficient Second-Order Minimization (ESM)

Efficient Second-Order Minimization [5] is another way to improve the tracking convergence radius which does however not affect the resulting accuracy.

So far, a first-order Taylor approximation of the pixel residuals $r_i(\Delta\boldsymbol{\xi})$ was used, repeated here in simplified notation (dropping the pixel index i):

$$r_{lin}(\Delta\boldsymbol{\xi}) = r(0) + \mathbf{J}_{\boldsymbol{\xi}=0}\Delta\boldsymbol{\xi} \tag{4.11}$$

This results in taking into account only the template frame gradient. Thus, if in an iteration a warped template image point lands on a target image area with negligible gradient, this point does not contribute to the correct solution as the best update direction for reducing its residual is unknown.

In order to get information even from such points, the Efficient Second-Order Minimization utilizes a second-order Taylor expansion:

$$r_{sec}(\Delta\boldsymbol{\xi}) = r(0) + \mathbf{J}_{\boldsymbol{\xi}=0}\Delta\boldsymbol{\xi} + \Delta\boldsymbol{\xi}^T\mathbf{H}_{\boldsymbol{\xi}=0}\Delta\boldsymbol{\xi} \tag{4.12}$$

Deriving this by $\Delta\boldsymbol{\xi}$ yields

$$\mathbf{J}_{\boldsymbol{\xi}=\Delta\boldsymbol{\xi}} = \mathbf{J}_{\boldsymbol{\xi}=0} + 2\Delta\boldsymbol{\xi}^T\mathbf{H}_{\boldsymbol{\xi}=0} \tag{4.13}$$

$$\Delta\boldsymbol{\xi}^T\mathbf{H}_{\boldsymbol{\xi}=0} = 0.5(\mathbf{J}_{\boldsymbol{\xi}=\Delta\boldsymbol{\xi}} - \mathbf{J}_{\boldsymbol{\xi}=0}). \tag{4.14}$$

Setting this in into the first (4.12) leads to:

$$r_{sec}(\Delta\boldsymbol{\xi}) = r(0) + 0.5(\mathbf{J}_{\boldsymbol{\xi}=\Delta\boldsymbol{\xi}} + \mathbf{J}_{\boldsymbol{\xi}=0})\Delta\boldsymbol{\xi} \tag{4.15}$$

With this, derivation continues like in the linear case. Thus, the essential difference is to take the mean of $\mathbf{J}_{\boldsymbol{\xi}=\Delta\boldsymbol{\xi}}$ and $\mathbf{J}_{\boldsymbol{\xi}=0}$ instead of just $\mathbf{J}_{\boldsymbol{\xi}=0}$. As the delta transformation is still unknown and to be determined, for $\mathbf{J}_{\boldsymbol{\xi}=\Delta\boldsymbol{\xi}}$ it is assumed to have the final correct $\Delta\boldsymbol{\xi}$ which will warp the source point to a target place which looks exactly as in the source image. Thus the source image gradient at the point's location, warped with the current warp estimate, is used for calculation of $\mathbf{J}_{\boldsymbol{\xi}=\Delta\boldsymbol{\xi}}$. Fig. 4.9 shows an example of how this technique increases the convergence radius.

For fast gradient warping, only the transformation which has most effect on it is considered in the final system: roll around the axis pointing along the camera view direction. It can be obtained by concatenating the current estimated rotation with the shortest rotation moving the forward-vector back to its original direction, i.e. neglecting the roll:

$$\mathbf{R}_{\text{roll}} = \mathbf{R}_{\text{back}}\mathbf{R}_{\text{estimated}} \tag{4.16}$$

As a result, $\mathbf{R}_{\text{roll}}$ consists of only the roll component of the rotation, which can be applied to the image gradients after dropping the third dimension (corresponding to the forward direction) from the matrix. For an evaluation of ESM's effect, see Fig. 7.1 in the results section.

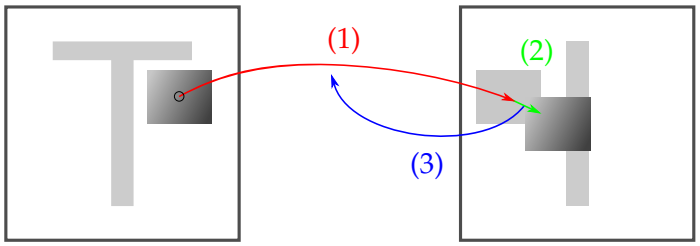


Figure 4.9: Example of how ESM increases the convergence radius: (1) The template point is warped to a homogeneous region. (2) By averaging template and image gradient, the correct update direction is still found if the estimated pose is close to the true pose. (3) The update is equal to the normal Lucas-Kanade method; see Fig. 4.6.

Optimization on Lie groups

The Gauss-Newton and Levenberg-Marquart optimization methods presented in Sec. 2.4 and used in the Lucas-Kanade method operate in Euclidean parameter space, but poses represented as $\mathrm{SE}(3)$ or $\mathrm{Sim}(3)$ group elements lie on a non-Euclidean manifold. If one would for example directly optimize on matrices, this over-parameterized representation would lead to problems. Optimization updates could lead out of the manifold and reprojection would need to be done.

In order to efficiently optimize poses in such spaces, the tangent space thus comes in handy: poses are stored as Lie group elements, but parameter updates are computed as

minimal Lie algebra elements. To apply a parameter update, it is first transformed to a group element with the hat and exp operations and then multiplied with the current state. The composition operator $\boxplus$ for the Lucas-Kanade method is thus defined as follows:

$$\mathbf{T}_i \boxplus \Delta\boldsymbol{\xi} := \exp(\widehat{\Delta\boldsymbol{\xi}})\mathbf{T}_i \tag{4.17}$$

Note that for this thesis, the multiplication order is switched in this operation. Alternatively, it could be left the same: both possibilities are equivalent and a pure implementation choice. Care has to be taken only to not mix them up. The above defines a left-multiplication convention and as mentioned in Sec. 4.2.2, the later estimated covariance matrix depends on this. g2o's default type implementations use a right-multiplication convention, so this must be paid attention to when using them with this library.

Defining parameter composition like this also requires to determine the Jacobian of the residuals for a multiplicative increment, which can be seen as a multiplicative instead of additive derivative.

4.2.3 Tracking on SE(3)

$\mathrm{SE}(3)$ tracking [28] operates on a reference (template) frame, consisting of image I_T, inverse depth map D_T and inverse depth variance map V_T, and an input frame image I_I for which no depth information is required, to determine the rigid body transformation as element of $\mathrm{SE}(3)$ between the two frames. This is done using image alignment as presented above, with the choices regarding the warp function and weighting described in the following.

The warp function is parametrized – in addition to the template pixel coordinates – with the inverse depth as further parameter. It transforms the un-projected pixel coordinates of the template into the currently estimated viewport of the input image, with $\mathbf{T}$ as a group element of $\mathrm{SE}(3)$:

$$\omega(\mathbf{x}, D_T(\mathbf{x}), \mathbf{T}) = \pi(\mathbf{T} * \pi^{-1}(\mathbf{x}, \frac{1}{D_T(\mathbf{x})})) \tag{4.18}$$

As the depth is necessary for the warp, only pixels having a valid depth estimate contribute to the residual; the others cannot be warped. The following is chosen as weighting function:

$$w_{\mathbf{x}}(\mathbf{T}) := \frac{\alpha_{\mathbf{x}}(\mathbf{T})}{\sigma^2_{r_{\mathbf{x}}(\mathbf{T})}} \tag{4.19}$$

$$\text{with} \quad \alpha_{\mathbf{x}}(\mathbf{T}) := \rho_\delta\left(\frac{r^2_{\mathbf{x}}(\mathbf{T})}{\sigma^2_{r_{\mathbf{x}}(\mathbf{T})}}\right) \tag{4.20}$$

$$\sigma^2_{r_{\mathbf{x}}(\mathbf{T})} := 2\sigma_I^2 + \left(\left.\frac{\partial r_{\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})}\right)^2 V_T(\mathbf{x}) \tag{4.21}$$

where ρ_δ is the Huber weighting function

$$\rho_\delta(r^2) := \begin{cases} 1 & \text{if } |r| \leq \delta \\ \frac{\delta}{|r|} & \text{otherwise.} \end{cases} \tag{4.22}$$

applied to the *normalized* residual. The residual's variance $\sigma^2_{r_{\mathbf{x}}(\mathbf{T})}$ is computed using uncertainty propagation as described in Sec. 2.5, and utilizing the available inverse depth variance V_T. Further, Gaussian intensity noise is assumed in both the reference and the input image.

Jacobians

For calculating pose updates, the Jacobian of the pixel residuals with respect to the 6 components of the pose update vector $\Delta\boldsymbol{\xi}$ is required, which is represented as a minimal Lie algebra element in $\mathfrak{se}(3)$. Setting in into Eq. 4.5 for SE(3) tracking, the Jacobian calculated by the chain rule is, with the un-projected point $\mathbf{p} = \pi^{-1}(\mathbf{x}, \frac{1}{D_T(\mathbf{x})})$:

$$\mathbf{J}_{\mathbf{x}} = \nabla I|_{\omega(\mathbf{x},\mathbf{T}_i)} \left.\frac{\partial \pi}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} \left.\frac{\partial \mathbf{T}\mathbf{p}}{\partial \mathbf{T}}\right|_{\mathbf{T}_i} \left.\frac{\partial \mathbf{T}\mathbf{T}_i}{\partial \mathbf{T}}\right|_{\mathbf{I}} \left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}} \tag{4.23}$$

These Jacobians are:

$\nabla I\|_{\omega(\mathbf{x},\mathbf{T}_i)}$	Derivative of the new image at the warped pixel position.
$\left.\frac{\partial \pi}{\partial \mathbf{p}}\right\|_{\mathbf{T}_i\mathbf{p}}$	Derivative of the projection function at the 3D point transformed with the current pose estimate $\mathbf{T}_i\mathbf{p}$.
$\left.\frac{\partial \mathbf{T}\mathbf{p}}{\partial \mathbf{T}}\right\|_{\mathbf{T}_i}$	Derivative of the matrix-vector multiplication of rigid body transformation at the current pose estimate with the un-projected pixel position.
$\left.\frac{\partial \mathbf{T}\mathbf{T}_i}{\partial \mathbf{T}}\right\|_{\mathbf{I}}$	Derivative of the rigid body transformation concatenation at the identity with the current pose estimate.
$\left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right\|_{\mathbf{0}}$	Derivative of the exp-hat function at the zero vector (corresponding to identity).

The order of elements in matrix derivatives is determined by virtually stacking their columns from left to right into a vector, from top to bottom. For derivatives of 3D rigid body (or similarity) transformation matrices, the last matrix row is neglected as it is always $(0, 0, 0, 1)$; thus these derivatives consist of the remaining 12 matrix elements only. The components of $\mathbf{p}$ are denoted x, y and z, while the components of $\mathbf{T}_i\mathbf{p}$ are denoted x', y' and z'. The current pose estimate $\mathbf{T}_i$ is represented in matrix form as:

$$\mathbf{T}_i = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.24}$$

The individual terms then evaluate to:

$$\nabla I|_{\omega(\mathbf{x},\mathbf{T}_i)} = \begin{pmatrix} \nabla I_x & \nabla I_y \end{pmatrix} \tag{4.25}$$

$$\left.\frac{\partial \pi}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} = \begin{pmatrix} f_x \frac{1}{z'} & 0 & -f_x \frac{x'}{z'^2} \\ 0 & f_y \frac{1}{z'} & -f_y \frac{y'}{z'^2} \end{pmatrix} \tag{4.26}$$

$$\left.\frac{\partial \mathbf{T}\mathbf{p}}{\partial \mathbf{T}}\right|_{\mathbf{T}_i} = \begin{pmatrix} x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 & 0 & 0 \\ 0 & x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 & 0 \\ 0 & 0 & x & 0 & 0 & y & 0 & 0 & z & 0 & 0 & 1 \end{pmatrix} \tag{4.27}$$

$$\left.\frac{\partial \mathbf{T}\mathbf{T}_i}{\partial \mathbf{T}}\right|_{\mathbf{I}} = \begin{pmatrix} r_{11} & 0 & 0 & r_{21} & 0 & 0 & r_{31} & 0 & 0 & 0 & 0 & 0 \\ 0 & r_{11} & 0 & 0 & r_{21} & 0 & 0 & r_{31} & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{11} & 0 & 0 & r_{21} & 0 & 0 & r_{31} & 0 & 0 & 0 \\ r_{12} & 0 & 0 & r_{22} & 0 & 0 & r_{32} & 0 & 0 & 0 & 0 & 0 \\ 0 & r_{12} & 0 & 0 & r_{22} & 0 & 0 & r_{32} & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{12} & 0 & 0 & r_{22} & 0 & 0 & r_{32} & 0 & 0 & 0 \\ r_{13} & 0 & 0 & r_{23} & 0 & 0 & r_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & r_{13} & 0 & 0 & r_{23} & 0 & 0 & r_{33} & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{13} & 0 & 0 & r_{23} & 0 & 0 & r_{33} & 0 & 0 & 0 \\ t_x & 0 & 0 & t_y & 0 & 0 & t_z & 0 & 0 & 1 & 0 & 0 \\ 0 & t_x & 0 & 0 & t_y & 0 & 0 & t_z & 0 & 0 & 1 & 0 \\ 0 & 0 & t_x & 0 & 0 & t_y & 0 & 0 & t_z & 0 & 0 & 1 \end{pmatrix} \tag{4.28}$$

$$\left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{4.29}$$

The final result obtained by matrix multiplication and simplification (e.g. by transformation $r_{11}x + r_{12}y + r_{13}z + t_x = x'$) is:

$$\mathbf{J}_{\mathbf{x}} = \frac{1}{z'} \begin{pmatrix} \nabla I_x f_x & \nabla I_y f_y \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{x'}{z'} & -\frac{x'y'}{z'} & (z' + \frac{x'^2}{z'}) & -y' \\ 0 & 1 & -\frac{y'}{z'} & -(z' + \frac{y'^2}{z'}) & \frac{x'y'}{z'} & x' \end{pmatrix} \tag{4.30}$$

For taking depth noise into account, the derivative of the residual with regard to the pixel's estimated inverse depth at its current estimate is required:

$$\left.\frac{\partial r_{\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} = \nabla I|_{\omega(\mathbf{x},\mathbf{T}_i)} \left.\frac{\partial \pi}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} \left.\frac{\partial \mathbf{T}_i\mathbf{p}}{\partial \mathbf{p}}\right|_{\mathbf{p}} \left.\frac{\partial \pi^{-1}}{\partial Z}\right|_{\frac{1}{D_T(\mathbf{x})}} \left.\frac{\partial \frac{1}{x}}{\partial x}\right|_{D_T(\mathbf{x})} \tag{4.31}$$

The new Jacobians are:

$\left.\frac{\partial \mathbf{T}_i \mathbf{p}}{\partial \mathbf{p}}\right|_{\mathbf{p}}$ Derivative of the matrix-vector multiplication of rigid body transformations at the un-projected point with the current pose estimate.

$\left.\frac{\partial \pi^{-1}}{\partial Z}\right|_{\frac{1}{D_T(\mathbf{x})}}$ Derivative of the un-projection function at the estimated depth with the current pixel position.

$\left.\frac{\partial \frac{1}{x}}{\partial x}\right|_{D_T(\mathbf{x})}$ Derivative of depth inversion at the current inverse depth estimate.

The pixel coordinates are denoted p_x and p_y. As above, x, y and $z = \frac{1}{D_T}$ respectively x', y' and z' denote the components of the un-projected pixel before and after transformation. The inverse intrinsic camera parameters matrix $\mathbf{K}^{-1}$ is represented in matrix form as:

$$\mathbf{K}^{-1} = \begin{pmatrix} k_{11} & 0 & k_{13} \\ 0 & k_{22} & k_{23} \\ 0 & 0 & 1 \end{pmatrix} \tag{4.32}$$

The Jacobians then evaluate to:

$$\left.\frac{\partial \mathbf{T}_i \mathbf{p}}{\partial \mathbf{p}}\right|_{\mathbf{p}} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{4.33}$$

$$\left.\frac{\partial \pi^{-1}}{\partial Z}\right|_{\frac{1}{D_T(\mathbf{x})}} = \begin{pmatrix} k_{11} p_x + k_{13} \\ k_{21} p_y + k_{23} \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} D_T(\mathbf{x}) \tag{4.34}$$

$$\left.\frac{\partial \frac{1}{x}}{\partial x}\right|_{D_T(\mathbf{x})} = \frac{-1}{D_T(\mathbf{x})^2} \tag{4.35}$$

The final result obtained by matrix multiplication and simplification (e.g. $r_{11}x + r_{12}y + r_{13}z = x' - t_x$) is:

$$\left.\frac{\partial r_{\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} = \frac{1}{D_T(\mathbf{x}) z'^2} \left(\nabla I_x f_x (t_x z' - t_z x') + \nabla I_y f_y (t_y z' - t_z y')\right) \tag{4.36}$$

4.2.4 Tracking on Sim(3)

Sim(3) tracking [15] operates on a pair of frames which must both contain a depth map, so scale estimation is possible. Components of the reference frame are denoted I_T, D_T and V_T, components of the input frame I_I, D_I and V_I. Further, the depth error must be included in the residual to be able to optimize for scaling.

The warp function is defined completely analogous to Eq. 4.18 from SE(3) tracking, just replacing the SE(3) element $\mathbf{T}$ by a Sim(3) equivalent. The residual is extended with the depth error as follows and now consists of the photometric residual $r_{p,\mathbf{x}}(\mathbf{T})$ and depth residual $r_{d,\mathbf{x}}(\mathbf{T})$ with associated weights. The notation $[\cdot]_i$ is used for access to the i-th vector component.

$$E(\mathbf{T}) := \sum_{\mathbf{x}\in\Omega_{D_T}} \left(\underbrace{\frac{\alpha_{\mathbf{x}}(\mathbf{T})}{\sigma^2_{r_{p,\mathbf{x}}(\mathbf{T})}}}_{=:w_{p,\mathbf{x}}(\mathbf{T})} r^2_{p,\mathbf{x}}(\mathbf{T}) + \underbrace{\frac{\alpha_{\mathbf{x}}(\mathbf{T})}{\sigma^2_{r_{d,\mathbf{x}}(\mathbf{T})}}}_{=:w_{d,\mathbf{x}}(\mathbf{T})} r^2_{d,\mathbf{x}}(\mathbf{T}) \right) \tag{4.37}$$

$$\text{with} \quad r_{p,\mathbf{x}}(\mathbf{T}) := I_T(\mathbf{x}) - I_I([\mathbf{x}']_{1,2}) \tag{4.38}$$

$$r_{d,\mathbf{x}}(\mathbf{T}) := [\mathbf{x}']_3 - D_I([\mathbf{x}']_{1,2}) \tag{4.39}$$

$$\alpha_{\mathbf{x}} := \rho_\delta \left(\frac{r^2_{p,\mathbf{x}}(\mathbf{T})}{\sigma^2_{r_{p,\mathbf{x}}(\mathbf{T})}} + \frac{r^2_{d,\mathbf{x}}(\mathbf{T})}{\sigma^2_{r_{d,\mathbf{x}}(\mathbf{T})}} \right) \tag{4.40}$$

Where $\mathbf{x}' := \omega_s(\mathbf{x}, D_T(\mathbf{x}), \mathbf{T})$ denotes the transformed and projected point, including its depth as 3rd component, and ρ_δ again denotes the Huber weighting function (see Eq. 4.22) applied to the sum of the normalized photometric and depth residual – which accounts for the fact that if one is an outlier, the other typically is as well. The variances required for normalization are computed using covariance propagation:

$$\sigma^2_{r_{p,\mathbf{x}}(\mathbf{T})} := 2\sigma_I^2 + \left(\left.\frac{\partial r_{p,\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} \right)^2 V_k(\mathbf{x}) \tag{4.41}$$

$$\sigma^2_{r_{d,\mathbf{x}}(\mathbf{T})} := V_I([\mathbf{x}']_{1,2}) \left(\left.\frac{\partial r_{d,\mathbf{x}}(\mathbf{T})}{\partial D_I}\right|_{D_I([\mathbf{x}']_{1,2})} \right)^2 + V_T(\mathbf{x}) \left(\left.\frac{\partial r_{d,\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} \right)^2 \tag{4.42}$$

Jacobians

The composition of the photometric residual $r_{p,\mathbf{x}}$ Jacobian with respect to the pose update vector is completely analogous to the $\mathrm{SE}(3)$ variant (Eq. 4.23):

$$\mathbf{J}_{p,\mathbf{x}} = \nabla I|_{\omega(\mathbf{x},\mathbf{T}_i)} \left.\frac{\partial \pi}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} \left.\frac{\partial \mathbf{T}\mathbf{p}}{\partial \mathbf{T}}\right|_{\mathbf{T}_i} \left.\frac{\partial \mathbf{T}\mathbf{T}_i}{\partial \mathbf{T}}\right|_{\mathbf{I}} \left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}} \tag{4.43}$$

The update vector now is a 7-component vector and $\mathrm{SE}(3)$ concepts are replaced by $\mathrm{Sim}(3)$ concepts. The terms which change their meaning are:

$\left.\frac{\partial \mathbf{T}\mathbf{p}}{\partial \mathbf{T}}\right|_{\mathbf{T}_i}$ This derivative is now of a similarity transformation instead of a rigid body transformation. However, the shape of the matrix is the same, and so is the resulting Jacobian. See Eq. 4.27.

$\left.\frac{\partial \mathbf{T}\mathbf{T}_i}{\partial \mathbf{T}}\right|_{\mathbf{I}}$ As above, the rigid body transformation changes to a similarity transformation here. This does not change the Jacobian either as the shape of the matrices is the same. See Eq. 4.28.

$\left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}}$ This derivative actually changes as now the exp-hat function of the $\mathrm{Sim}(3)$ group is used. This adds one column corresponding to the generator of the scale.

The derivative of the Sim(3) exp-hat function is extended by one column to:

$$\left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.44}$$

Calculating the resulting Jacobian by multiplication shows that it is extended by a column of zeros (because multiplying the whole transformation with a scale factor does not change the projected pixel's position in the camera image):

$$\mathbf{J}_{p,\mathbf{x}} = \frac{1}{z'} \begin{pmatrix} \nabla I_x f_x & \nabla I_y f_y \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{x'}{z'} & -\frac{x'y'}{z'} & (z' + \frac{x'^2}{z'}) & -y' & 0 \\ 0 & 1 & -\frac{y'}{z'} & -(z' + \frac{y'^2}{z'}) & \frac{x'y'}{z'} & x' & 0 \end{pmatrix} \tag{4.45}$$

Sim(3) tracking also requires the Jacobian of the depth residual with regard to the pose. With $\pi_z(\mathbf{x})$ being the function mapping a 3D point $\mathbf{x}$ to its inverse z component (depth), the full Jacobian is:

$$\mathbf{J}_{d,\mathbf{x}} = \left(\left.\frac{\partial \pi_z}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} - \nabla D|_{\omega(\mathbf{x},\mathbf{T}_i)} \left.\frac{\partial \pi}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} \right) \left.\frac{\partial \mathbf{T}\mathbf{p}}{\partial \mathbf{T}}\right|_{\mathbf{T}_i} \left.\frac{\partial \mathbf{T}\mathbf{T}_i}{\partial \mathbf{T}}\right|_{\mathbf{I}} \left.\frac{\partial \exp(\widehat{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}}\right|_{\mathbf{0}} \tag{4.46}$$

The depth gradient is neglected in the following for performance reasons, approximating it to be zero. The derivative of the inverse depth projection is:

$$\left.\frac{\partial \pi_z}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} = \begin{pmatrix} 0 & 0 & -\frac{1}{z'^2} \end{pmatrix} \tag{4.47}$$

Calculating the complete Jacobian results in:

$$\mathbf{J}_{d,\mathbf{x}} = \begin{pmatrix} 0 & 0 & -\frac{1}{z'^2} & -\frac{y'}{z'^2} & \frac{x'}{z'^2} & 0 & 0 & -\frac{1}{z'} \end{pmatrix} \tag{4.48}$$

Furthermore, derivatives of the residuals regarding the inverse pixel depths are required. For the photometric residual, the Jacobian is completely the same as for the SE(3) case (Eq. 4.36):

$$\left.\frac{\partial r_{p,\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} = \frac{1}{D_T(\mathbf{x}) z'^2} \left(\nabla I_x f_x (t_x z' - t_z x') + \nabla I_y f_y (t_y z' - t_z y')\right) \tag{4.49}$$

For the depth residual, the derivative by the new image inverse depth is simple:

$$\left.\frac{\partial r_{d,\mathbf{x}}(\mathbf{T})}{\partial D_I}\right|_{D_I([\mathbf{x}']_{1,2})} = -1 \tag{4.50}$$

The derivative by the template image's inverse depth is:

$$\left.\frac{\partial r_{d,\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} = \left.\frac{\partial \pi_z}{\partial \mathbf{p}}\right|_{\mathbf{T}_i\mathbf{p}} \left.\frac{\partial \mathbf{T}_i\mathbf{p}}{\partial \mathbf{p}}\right|_{\mathbf{p}} \left.\frac{\partial \pi^{-1}}{\partial Z}\right|_{\frac{1}{D_T(\mathbf{x})}} \left.\frac{\partial \frac{1}{x}}{\partial x}\right|_{D_T(\mathbf{x})} \tag{4.51}$$

All terms of this expression have been presented already. The multiplied result is:

$$\left.\frac{\partial r_{d,\mathbf{x}}(\mathbf{T})}{\partial D_T}\right|_{D_T(\mathbf{x})} = \frac{z' - t_z}{D_T(\mathbf{x}) z'^2} \tag{4.52}$$

4.3 Keyframe selection

Figure 4.10: Example of a continuous sequence of selected keyframes at the beginning of the fr2/desk sequence from [50], using the entropy ratio selection method and a ratio threshold of 0.7.

Each camera frame in a video in normally provides new information about the scene, and in an ideal system, all frames ever observed would be remembered to perform a global optimization over the whole video. As memory and processing power is limited in real systems, this is unfortunately unfeasible in a real-time setting and some kind of reduction has to be done. Because of this, selected camera frames are promoted to keyframes by the system. In contrast to normal frames, the system keeps keyframes in memory and thus these are the frames on which global optimization is performed. As a possible benefit, keyframe selection allows to prevent low-quality frames from being used as tracking reference, which may improve tracking quality.

To illustrate the need for data reduction, as a rough estimate, on a current high-end smartphone 2GiB of memory may be available to the application for camera frames. Assuming that camera frames are processed at 320×240 pixels with 30 Hz and stored as 32-bit per pixel (for example as single-precision floating point values), the memory would be full after 3.9 minutes of video. Furthermore, in practice frames may take up significantly more memory, for example if a depth map estimated for a frame is stored in addition to the image data.

This section describes the implications of a keyframe-based sparsification of camera frames. First, in Sec. 4.3.1 the data stored in keyframes and their different purposes are discussed. Then criteria for an ideal keyframe selection method are derived from that. Finally, different methods for keyframe selection are presented in Sec. 4.3.2 and evaluated in Sec. 4.3.3.

4.3.1 Theory

As keyframes are the only frames which are remembered by the system for a longer time, they store much information and are used for a multitude of different purposes. The main information stored is:

- Image data of the frame
- Semi-dense inverse depth map
- Inverse depth variance map
- Estimated pose
- Parent keyframe from which this was tracked
- Adjacent keyframes in keyframe graph

The purposes for which keyframes are used are:

- **Tracking reference:** new frames are tracked with the latest selected keyframe as tracking reference. This is because keyframe selection tries to select the best frames – with the least distortion or degradation – so they are best suited for tracking. Furthermore, depth map refinement is performed on the current tracking reference, which is useful in order that the keyframe later has a good depth map estimate.
- **Pose constraint acquisition:** keyframes are used as vertices of the keyframe graph. To perform global optimization of the graph structure, pose constraints between the vertices must be estimated using the data stored in keyframes.
- **Large-baseline stereo:** stereo comparisons are usually done between the current tracking reference and the current frame, which is close to the reference frame. Keyframes are the only possibility for large-baseline stereo comparisons.
- **Reconstruction:** if the point cloud generated by merging all semi-dense depth maps is used for 3D reconstruction purposes, keyframes are those frames contributing to the final reconstruction.

Deriving from this, the criteria for good keyframe selection are:

- A keyframe should not show un-modeled distortions such as motion blur, noise or rolling shutter distortion.
- Keyframes should not be generated redundantly at the same camera pose to prevent unnecessary processing.
- Enough keyframes for global optimization should be selected.
- The next keyframe should be selected while the current input frame can still be well tracked from the current keyframe, in order to prevent unnecessarily high pose errors.

- Enough keyframes for stereo comparisons should be selected.
- For reconstruction purposes, a new keyframe should be selected when the coverage of the current input frame by the existing keyframes becomes too low.
- As few keyframes as possible should be selected to save memory.

The following section presents different methods which were examined, which attempt to fulfill these requirements.

4.3.2 Selection methods

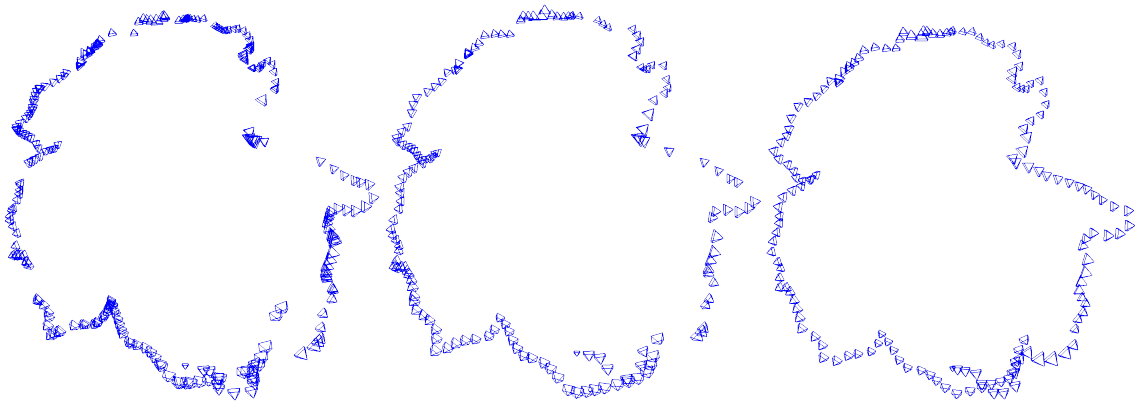


Figure 4.11: Optical comparison of selected keyframe poses by different keyframe selection methods for the fr2/desk benchmark sequence of [50], in which the camera moves around a central desk. From left to right: residual ratio, entropy ratio, pose-based measure. For the latter two, the selection threshold was chosen to result in approximately 150 keyframes for the sequence. For the residual ratio criterion, more frames were selected as the sequence is not tracked well otherwise.

Frame count measure

As the simplest method, this counts the frames received since creation of the current keyframe and selects a new keyframe at a specific frequency. This can be used to simulate frame-to-frame tracking with the extreme case of setting the keyframe interval to 1.

Residual ratio measure

This method is based on the tracking quality measure given by the root mean square residual

$$\sqrt{\frac{1}{n}\sum_{\mathbf{x}} r_{\mathbf{x}}^2}. \tag{4.53}$$

It selects a new keyframe if the ratio of this residual to the residual of the frame immediately following the current keyframe becomes too big. The new keyframe is then chosen as the newest frame whose residual ratio is still below the threshold.

Entropy ratio measure

This method, presented in [27], is based on a more sophisticated measure of tracking quality by means of the differential entropy of a multivariate normal distribution, which is a scalar measure of its uncertainty. The formula for the entropy calculated from the N-dimensional distribution's variance Σ is:

$$\frac{1}{2}\ln((2\pi e)^N \det(\Sigma)) \tag{4.54}$$

This is calculated for the estimated covariance of the current frame's pose estimate. It then applies ratio thresholding in the same way as the residual based method. While this works well for many scenes, the differential entropy may become negative. Taking ratios is problematic then, which makes it unsuitable for these cases.

Pose-based measure

This method thresholds a weighted combination of the translational and angular distance of the current frame to the current keyframe. With $\mathbf{t}$ being the translation between reference frame and current frame, and $\mathbf{q}_T$ and $\mathbf{q}_I$ the camera-to-world rotation quaternions of reference frame and current frame, the employed formula is:

$$||\mathbf{t}||_2 + \alpha 2\text{acos}(\mathbf{q}_T \cdot \mathbf{q}_I) \tag{4.55}$$

It selects a new keyframe if a certain threshold distance is reached. Note that the translation is taken relative to the average scene depth, as the real translation's scale is unknown because of scale ambiguity.

4.3.3 Conclusion

None of the measures presented so far fulfills the requirements stated above: the frame count measure is obviously too simple. The residual ratio measure does generally not result in a good selection of keyframes. It is furthermore dependent on the next frame following a keyframe, introducing an unwanted dependency. The entropy measure also suffers from this and in addition, taking ratios of differential entropies is unsuited and leads to a very high number of selected keyframes when the entropies approach zero. The pose-based measure neglects e.g. possible scale changes, so when going over from a large-distance scene to a close-distance scene, not enough keyframes may be taken. An evaluation of the two best measures is provided by Fig. 7.2 in the results section.

As best alternative, a combination of the pose-based and residual ratio measure is proposed, such that a new keyframe is taken if either of the measures suggests that it should be done. This allows to use the stable pose-based method together with an additional measure making sure that enough keyframes are taken in cases where this is necessary for different reasons than pose changes. An additional idea is discussed among future work in Sec. 8.1.

4.4 Depth mapping

The system uses an inverse depth map and inverse depth variance map estimated for the tracking reference frame for pose tracking. Mapping is the process of refining these estimates over time by integrating measurements from new frames and regularizing the depth map to enforce smoothness. In addition, transforming a depth map to a newly selected keyframe as initial estimate for its depth is also part of the mapping component.

The stereo comparison and regularization components were taken over from [16] and have not been changed in this thesis, so they are not described here. However, the frame processing order is different in a SLAM system with keyframe-to-frame tracking as opposed to a visual odometry system with frame-to-frame tracking, requiring some changes described Sec. 4.4.1. In addition, amending the initial depth estimate for new tracking references with estimates from old keyframes is discussed in Sec. 4.4.2 and Sec. 4.4.3 presents an approach to correct depth maps in keyframe graph optimization.

4.4.1 Stereo comparison scheme

New input frames are always tracked from the current keyframe. Thus its depth estimate is improved as long as it is in active use in order to determine the best possible depth map for it with the limited computing power available.

As described in [16], first small-baseline stereo comparisons should be done to get a rough estimate, followed by successively larger baselines and limited search range to get a more precise estimate while avoiding local optima.

Thus, when a new keyframe is created, stereo comparisons are first done with the immediately following frames. As mapping usually runs with a lower frequency than that of incoming new frames, intermediate frames may be skipped; however, it is important to process the frame immediately following a keyframe as it normally provides the smallest baseline, making it easiest to initialize new depth observations on it. For this reason, the system specifically ensures that these frames are processed by the mapping component.

If a certain accuracy has been reached for a pixel, large-baseline stereo comparisons may be done with old keyframes. For these, a pose optimization has been done already which makes them very accurate and well suited for stereo. The scheme is shown in Fig. 4.12.

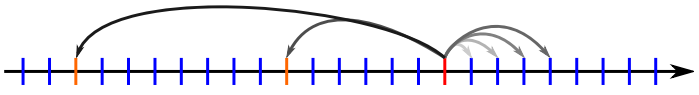


Figure 4.12: Visualization of the stereo comparison scheme, with frames shown as bars on the time axis. From the current keyframe (red), stereo comparisons are first done with immediately following normal frames (blue). On reaching a certain estimated accuracy, it is possible to continue the refinement steps with old keyframes (orange).

4.4.2 Depth map propagation

The system of [16] initializes depth estimates in a new frame by transforming the latest estimated depth map into it. A problem with this approach is that close-by objects leave a "shadow" in the depth map as the camera moves by. This is because depth observations belonging to objects behind them are deleted when occluded by close objects and need to be re-estimated as soon as they come into view again. To remedy this, depth maps from a number of old keyframes can be projected into the new frame in addition. Fig. 4.13 shows an example of how this helps to get a denser initial depth estimate for a frame. Conflict handling is required for pixels to which multiple depth estimates are projected to:

- If the conflicting depth estimates occlude each other, the estimate in front overwrites the other. Occlusion can be estimated with the inverse depths d_i and inverse depth variances v_i by checking if the estimates are within their 2-variance range as follows:

$$(d_1 - d_2)^2 > v_1 + v_2 \tag{4.56}$$

- If the conflicting estimates are close together so no occlusion is assumed, the one with lower variance is chosen. This is motivated by the fact that often, the conflicting estimates are not independent, but instead result from the same estimate propagated along. So no new information is added in many cases, instead the better (often newer) estimate should be chosen.

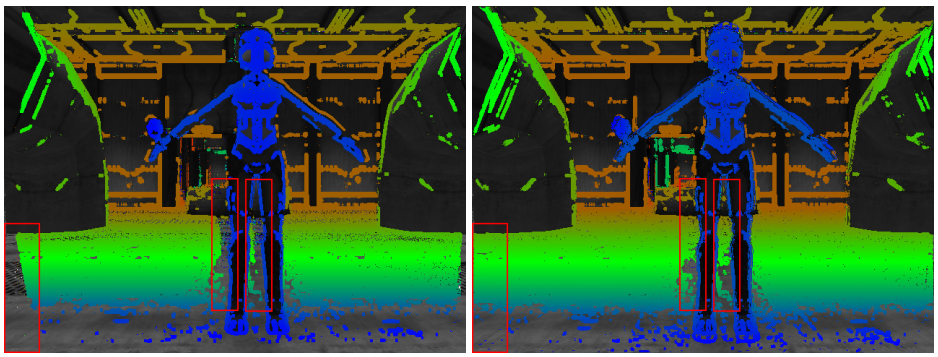


Figure 4.13: Comparison of depth maps without and with keyframe depth propagation in a simulated scene. The camera was moved first to the right, then to the left again. **Left:** no keyframe depth propagation. At the view border and at areas uncovered from occlusion, depth estimates are missing and need to be estimated again. **Right:** with keyframe depth propagation. The missing depth estimates were inserted again from an old keyframe.

4.4.3 Depth independency and retransformation

When a keyframe is no longer actively used as tracking reference, loop closures are found for it and the global keyframe graph optimized. This corrects the keyframe poses contained in the graph. However, depth maps contained within the keyframes are not corrected by this except for the scaling estimated by Sim(3) optimization. Because of this,

a different method should be used to make sure they are corrected – otherwise, errors introduced once will stay.

For keypoint-based SLAM systems, this problem is solved by **bundle adjustment**: the keypoints' positions are estimated in all frames independently and in the optimization step, camera poses are optimized jointly with 3D position estimates of keypoints.

For a direct per-pixel SLAM system, this type of optimization would be difficult as the number of depth estimates is too high to be handled in optimization. Furthermore, depth estimates between frames are not independent and the matching of depth estimates from different frames unknown. The following subsection details an approach to address this problem.

Depth map retransformation

This approach tries to correct for depth map errors after an optimization operation finishes which changed the keyframe's pose. For performance reasons, this operation is applied only for the first time a keyframe is optimized within a new loop in the graph – this is the optimization iteration in which the largest change is to be expected.

The underlying assumption is that a pose change of a keyframe relative to the keyframe it was tracked on (called the parent keyframe) results from a correction of a tracking error. Thus, the point cloud defined by the depth map should stay at the same relative position to the parent keyframe. To achieve this, in the coordinate frame of the new keyframe, the depth map must be transformed with a correction offset. This is calculated by first taking the transform to the coordinate system of the anchor frame, which is the first frame along the path of parent keyframes of the current frame which is not corrected in this retransformation iteration. Then all new transformation corrections of frames along the path from the anchor to the current frame are composed with this to compute the final retransformation.

With $\mathbf{T}$ denoting old camera-to-world poses, $\mathbf{T}'$ new camera-to-world poses, numbering them from 1 (the frame to be corrected) to $n+1$ (the anchor frame), and $\mathbf{S}_i$ scaling matrices with scaling factors defined below from the old and new frame scales s and s', the correction transformation to be applied to the point cloud is computed as:

$$\text{with} \quad \mathbf{S}_i := \text{scaling}(\frac{s'_i}{s_i} / \frac{s'_{i+1}}{s_{i+1}}) : \tag{4.57}$$

$$\left(\prod_{i=1}^{n} \left(\mathbf{S}_i \mathbf{T}'^{-1}_i \mathbf{T}'_{i+1} \right) \right) \cdot \text{scaling}(\frac{s'_{n+1}}{s_{n+1}}) \mathbf{T}^{-1}_{n+1} \mathbf{T}_1 \tag{4.58}$$

In effect this first transforms points back into the anchor frame using the old frame poses. Then it iteratively transforms them along the path to the current frame using new poses and applies a differential scale correction after each step. Switching from old to new poses makes the points move along with movements of the anchor frame which stem from non-local pose corrections, e.g. caused by large loop closures. Note that in addition to the scaling done by the above transform, points are implicitly scaled by changes to the current frame's scale estimate. The retransformation process is illustrated by Fig. 4.14 for $n = 2$.

The disadvantages of this approach are that it involves additional map transformations, and it only corrects offsets of the whole depth map resulting from wrong tracking. How-

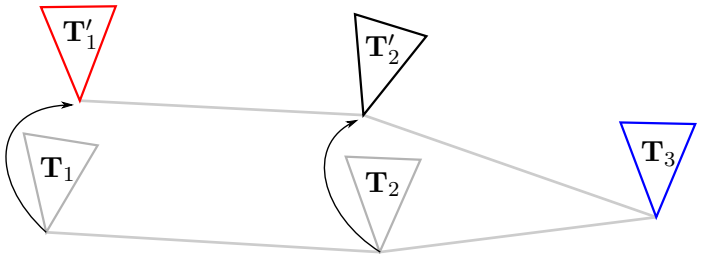


Figure 4.14: Example keyframe graph for depth map retransformation of the red frame. The previous frame poses are shown in gray. The $\boldsymbol{\xi}$ and $\boldsymbol{\xi}'$ represent the old and new camera-to-world poses of the frames. $\boldsymbol{\xi}_3$ is the camera-to-world pose of the (blue) anchor frame. Its absolute pose might also change, but is kept fixed relative to the points in the current frame.

ever, in experiments with synthetically added tracking errors it has been shown to significantly improve the result.

4.5 Constraint search

Constraint search consists of finding candidates from the set of existing keyframes which are likely to have an overlap with a given new keyframe, and determining their relative pose to the new keyframe. The pose constraints obtained this way are later inserted into the keyframe graph in order to do global pose optimization.

This chapter first presents in Sec. 4.5.1 different criteria and means to find suitable constraint candidates for a new keyframe. In Sec. 4.5.2, two alternative ways of determining a $\mathrm{Sim}(3)$ constraint between the new keyframe and a candidate keyframe are shown. Finally, Sec. 4.5.3 shows how constraints determined in this way are validated in order to reject estimates which come from a local pose optimization optimum instead of the true optimum.

4.5.1 Candidate search

The subtask of finding candidates has as input a new keyframe and as output a subset of candidate frames from the set of existing keyframes for which a relative pose estimation to the new keyframe is likely to be possible.

Criteria

For performance reasons, usually only a small number of relative poses can be determined, so the number of selected candidates should be low. The criteria for selecting a suitable candidate are:

- **Trackability:** the candidate must have sufficient overlap with the new keyframe, so the relative pose can be determined.

- **Information gain:** the constraint resulting from tracking the candidate pose should contribute as much as possible to optimizing the keyframe graph to be closer to the true keyframe graph.

These conditions are in some way opposing:

- Candidates for which the probability of being trackable is high, for example recently visited keyframes, usually do not add much information as their relative pose is well known already.
- On the other hand, keyframes which were not visited for a long time have a lower probability of being trackable from a given pose, as due to pose drift the relative pose estimate is very uncertain. If tracking such a frame succeeds however, the relative pose uncertainty can be greatly reduced, adding much information.

This leads to the notion of "small" and "large" loop closures, even if there is no objective distinction between them. Both are useful to the system: small loop closures correct local errors and add reliability; large loop closures correct a longer trajectory when visiting a known place again after some pose uncertainty has accumulated.

In the system developed in this thesis, two different methods for candidate finding are used which can be associated with these categories: pose based search finds small loop closures, while appearance based search finds large loop closures.

Pose based search

This candidate finding method works with the keyframe poses determined by the system. It is based on the assumption that keyframes which are estimated to be close to the new keyframe are likely to be trackable. Thus, a simple measure is to select the n closest keyframes based on a pose distance measure including positional and rotational difference.

Note that one could assume that instead of closeness of keyframes, closeness of the estimated geometry should be tested for. However, trackability often depends on keyframe closeness, as unmodeled lighting effects often prevent tracking between more distant viewports.

A disadvantage of this approach is that it often selects the latest created keyframes only which have been visited recently. Furthermore, for more distant keyframes in terms of vertex distance in the keyframe graph, the high relative uncertainty to the new keyframe makes direct pose comparisons useless.

Appearance based search

This candidate finding method is based on visual similarity between images. It selects keyframes with visually similar images to that of the new keyframe, which are likely to stem from the same place. A basic building block in algorithms for doing this is the **bag-of-words** representation of images: in a pre-processing step, first a set of **codewords** is learned by processing a set of training images. For each image, with a feature detector and descriptor, feature vectors of all determined features in the image are calculated. Then

clustering in feature space is used to find representative feature clusters. The center of each cluster becomes a codeword.

For processing an input image, the first step is feature detection and description, as for the training images. Then for each found feature vector the closest codeword is found. The images is then represented as a histogram vector of codeword counts. This is used to compare similarity between images: in similar scenes, the distribution of observed codewords is also similar.

The system of this thesis uses the existing openFabMap library [20] for appearance based loop-closure search, which implements different variants of the FabMap algorithm published earlier. The system uses the implementation of FabMap version 2 [10].

A disadvantage of this approach is that it obviously cannot cope with repetitive structures which lead to identical images taken from different positions. FabMap recognizes indistinct place observations and assigns them a low probability of coming from the same place.

4.5.2 Pose tracking

For every candidate keyframe determined by the means described above, the system tries to estimate its relative pose to the new keyframe. In addition, the direct predecessor keyframe of the new keyframe is also tracked to obtain a $\mathrm{Sim}(3)$ pose estimate between those frames. Previously this relative pose was only tracked on $\mathrm{SE}(3)$ because no depth map was available for the new frame yet. This changes when it is selected as a keyframe and its depth map refined successively. The following subsections describe two alternative approaches for constraint tracking.

Sim(3) **tracking**

Direct $\mathrm{Sim}(3)$ tracking (see Sec. 4.2.4) estimates relative position, rotation and scale simultaneously. The result, together with its associated 7×7 information matrix determined as described in Sec. 4.2.2 can be directly used as a 7-DOF constraint in the keyframe graph.

SE(3) tracking and scale estimation

Alternatively, based on $\mathrm{SE}(3)$ tracking (which is a vital system component for tracking new frames anyway), a separate scale estimation step can be attempted as an additional step after pose tracking.

This is done by projecting the depth map of one keyframe into the other to obtain a set of depth sample pairs (y_i, x_i) from all positions where a projected depth pixel is transformed to the same position as a depth pixel of the other keyframe. From these, a scaling factor λ is to be determined such that ideally for all samples it holds $x_i = \lambda y_i$.

While this might sound like a simple problem at first, a naive approach such as the arithmetical mean of scaling factors is unsuitable, as the following overview of approaches shows.

Arithmetical mean This is calculated by:

$$\frac{1}{n}\sum_{i=0}^{n}\frac{x_i}{y_i} \tag{4.59}$$

The result is heavily influenced by single erroneous summands which are close to zero or infinity. As an example, for 100 samples with expected result 1, having 99 correct samples and 1 outlier with a ratio of 100 results in an average of 1.99, which is far off.

Geometrical mean This is calculated by:

$$\sqrt[n]{\prod_{i=0}^{n}\frac{x_i}{y_i}} \tag{4.60}$$

For the example given in the above paragraph, this results in an estimated value of 1.0471. As Fig. 4.15 shows, the geometrical mean in general performs far better than the arithmetical mean for this application, but still has a bias, i.e. it does not converge to the correct value with the number of samples approaching infinity.

Note that for stability reasons, the geometrical mean is usually calculated by applying $\exp(\log(\cdot))$ to above term, resulting in:

$$\exp\left(\frac{1}{n}\sum_{i=0}^{n}\log(\frac{x_i}{y_i})\right) \tag{4.61}$$

This prevents having to multiply all samples with each other. Instead, the arithmetical mean of logarithms is computed and exponentiated at the end.

Maximum-likelihood estimator This estimator is based on probabilistically modeling the problem, taking into account the different variance estimates $\sigma_{y,i}^2$ and $\sigma_{x,i}^2$ for each depth sample. It is assumed that the depth samples are distributed as follows, with μ_i being the unknown true depth of the sample pair:

$$x_i \sim \mathcal{N}(\lambda\mu_i, \sigma_{x,i}^2) \tag{4.62}$$

$$y_i \sim \mathcal{N}(\mu_i, \sigma_{y,i}^2) \tag{4.63}$$

From this, analogously to Sec. 4.3 of [17] the log-likelihood $\mathcal{L}(\lambda)$ to minimize and its first and second derivatives are determined:

$$\mathcal{L}(\lambda) \propto \mathcal{L}_p(\lambda) := \frac{1}{2}\sum_{i=1}^{n}\frac{(x_i-\lambda y_i)^2}{\lambda^2\sigma_{y,i}^2+\sigma_{x,i}^2} \tag{4.64}$$

$$\mathcal{L}_p(\lambda)' = \sum_{i=1}^{n}\frac{(\lambda y_i - x_i)(\sigma_{x,i}^2 y_i + \lambda\sigma_{y,i}^2 x_i)}{(\lambda^2\sigma_{y,i}^2+\sigma_{x,i}^2)^2} \tag{4.65}$$

$$\mathcal{L}_p(\lambda)'' = \sum_{i=1}^{n}\frac{y_i^2}{\lambda^2\sigma_{y,i}^2+\sigma_{x,i}^2} + \frac{4\lambda^2\sigma_{y,i}^4(x_i-\lambda y_i)^2}{(\lambda^2\sigma_{y,i}^2+\sigma_{x,i}^2)^3}$$

$$- \frac{\sigma_{y,i}^2(x_i - \lambda y_i)^2 + 4\lambda\sigma_{y,i}^2 y_i(x_i - \lambda y_i)}{(\lambda^2\sigma_{y,i}^2 + \sigma_{x,i}^2)^2} \tag{4.66}$$

As there is no analytical solution for λ, this is used by first calculating the geometric mean of the samples as initial estimate, as this was experimentally shown to yield the best direct results. Then this estimate is updated with one Gauss-Newton step utilizing above derivatives. Results are shown in Fig. 4.15. A disadvantage of this approach is that it does not determine covariance between pose and scale, thus direct Sim(3) tracking should be preferred. Furthermore, e.g. occlusions are not taken into account, except if corresponding per-pixel weighting is added.

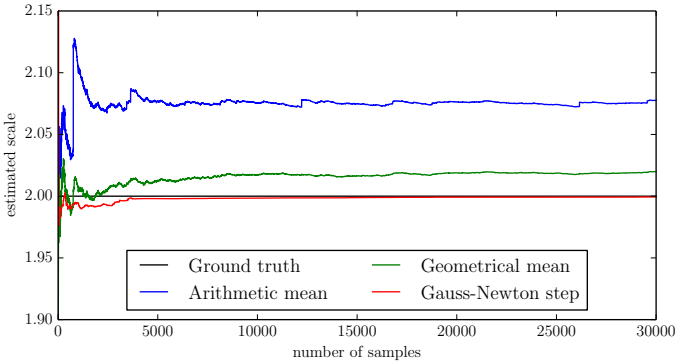


Figure 4.15: Scale estimation results using different approaches, with a growing number of samples. Only the approach involving one Gauss-Newton step converges to the right value. The parameters used in this experiment are: $\lambda = 2$, μ_i is drawn from $\mathcal{N}(10, 1^2)$, y is drawn from $\mathcal{N}(\mu_i, \mathcal{N}(0.3, 1.5^2)^2)$, x is drawn from $\mathcal{N}(\lambda\mu_i, \mathcal{N}(1, 1.7^2)^2)$.

4.5.3 Constraint validation

For tracking new camera frames, with the pose of the previous frame a very good initial pose estimate is available because usually the movement from one frame to the next is small relative to the observed scene geometry. For tracking loop closures the initial estimate may however be much worse if drift has accumulated. Thus the danger of tracking converging to a local optimum instead of the correct solution is high and a means to validate the result must be used to prevent using wrong pose estimates.

A technique for doing this is **reciprocal tracking**: after successfully tracking frame B with frame A as reference, the roles are switched and frame A is tracked with frame B as reference, using the same initial estimate (in one case inverted) for both tracking attempts. The resulting constraint is added only if both pose estimates obtained this way are statistically similar, as suggested by [15]. As an additional benefit, the two estimates can be both used instead of having only one constraint to obtain a possibly more accurate pose estimate.

To check if the estimate $\mathbf{T}_{AB}$ with covariance $\mathbf{\Sigma}_{AB}$ is consistent with its reciprocal estimate $(\mathbf{T}_{BA}, \mathbf{\Sigma}_{BA})$, first an error vector (which should be zero in the ideal case) is defined

as:

$$\mathbf{e} := \log(\mathbf{T}_{BA}\mathbf{T}_{AB})^\vee \tag{4.67}$$

Its deviation from zero, relative to its covariance $\mathbf{\Sigma}$, can be quantified using the Mahalanobis norm:

$$\sqrt{\mathbf{e}^T\mathbf{\Sigma}^{-1}\mathbf{e}} \tag{4.68}$$

which is thresholded to accept or reject the constraint. For this however, the covariance $\mathbf{\Sigma}$ of $\mathbf{e}$ must be estimated. This is done using uncertainty propagation of all involved variables, as introduced in Sec. 2.5. First, $\mathbf{T}_{BA}$ is examined. As tracking errors in the system are modeled as left-multiplied increments $\boldsymbol{\epsilon}$ onto the true pose, looking at $\mathbf{T}_{BA}$ the error term can be written as:

$$\mathbf{e} = \log(\exp(\widehat{\boldsymbol{\epsilon}_{BA}})\mathbf{T}_{BA,\text{true}}\mathbf{T}_{AB})^\vee \tag{4.69}$$

The Jacobian for uncertainty propagation is then simply:

$$\mathbf{J}_{\boldsymbol{\epsilon}_{BA}} = \frac{\partial}{\partial \boldsymbol{\epsilon}_{BA}}\boldsymbol{\epsilon}_{BA} = \mathbf{I} \tag{4.70}$$

Next, $\mathbf{T}_{AB}$ is examined. First the adjoint property given in Eq. 2.24 is used to bring the $\boldsymbol{\epsilon}$ to the left:

$$\mathbf{e} = \log(\exp(\widehat{\mathbf{Ad}_{BA}\boldsymbol{\epsilon}_{AB}})\mathbf{T}_{BA}\mathbf{T}_{AB,\text{true}})^\vee \tag{4.71}$$

Then the Jacobian is calculated:

$$\mathbf{J}_{\boldsymbol{\epsilon}_{AB}} = \frac{\partial}{\partial \boldsymbol{\epsilon}_{AB}}\mathbf{Ad}_{BA}\boldsymbol{\epsilon}_{AB} = \mathbf{Ad}_{BA} \tag{4.72}$$

So the final combined covariance is:

$$\underbrace{\mathbf{I}\mathbf{\Sigma}_{BA}\mathbf{I}^T}_{\mathbf{\Sigma}_{BA}} + \mathbf{Ad}_{BA}\mathbf{\Sigma}_{AB}\mathbf{Ad}_{BA}^T \tag{4.73}$$

4.6 Graph optimization

Graph optimization takes as input the keyframe graph, consisting of

- A **vertex** with an absolute frame-to-world pose estimate $\mathbf{T}_i$ for each keyframe i, and
- An **edge** between vertices i and k with a relative i-to-k transformation estimate $\mathbf{T}_{ki}$ and associated covariance matrix $\mathbf{\Sigma}_{ki}$ for each constraint determined between two keyframes i and k.

The process optimizes the absolute vertex poses $\mathbf{T}_i$ to best fulfill the constraints given by the edges. Mathematically, graph optimization therefore minimizes the following error norm, consisting of a sum of squared Mahalanobis norms (see Eq. 4.68) with one term for each constraint in the constraint set $\mathcal{E}$:

$$E(\mathbf{T}_1 \dots \mathbf{T}_n) := \sum_{(\mathbf{T}_{ki},\mathbf{\Sigma}_{ki})\in\mathcal{E}} \log(\mathbf{T}_{ki}\mathbf{T}_i^{-1}\mathbf{T}_k)^{\vee T}\mathbf{\Sigma}_{ij}^{-1}\log(\mathbf{T}_{ki}\mathbf{T}_i^{-1}\mathbf{T}_k)^\vee \tag{4.74}$$

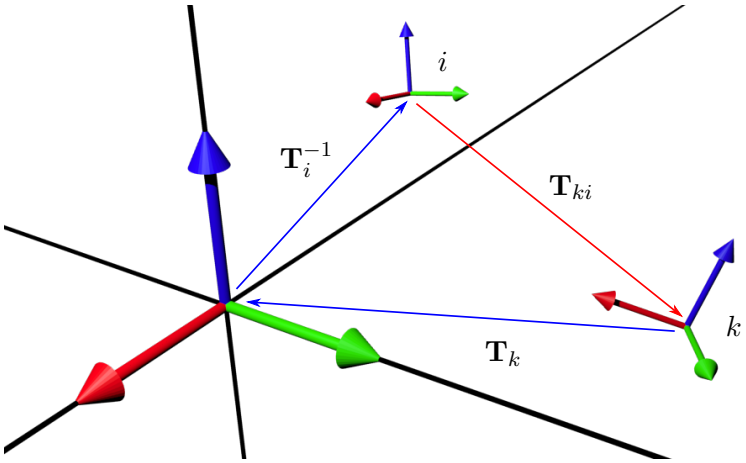


Figure 4.16: The error term $\mathbf{T}_{ki}\mathbf{T}_i^{-1}\mathbf{T}_k$ is composed of three transformations visualized here. The colored coordinate systems show the world coordinate frame (left) and two frame poses, i and k. Arrows between the frames show the coordinate system switch done by the respective transformation. The red arrow corresponding to $\mathbf{T}_{ki}$ is the measurement, while the blue arrows depict the poses to be optimized. If the constraint is fulfilled exactly, the transformations form a circle.

Here, $\log(\mathbf{T}_{ki}\mathbf{T}_i^{-1}\mathbf{T}_k)^\vee$ is the error vector which is $\mathbf{0}$ if the constraint is fulfilled exactly. Note that the order of transformations in this term matters, as the covariance estimate of the whole term is assumed to be $\mathbf{\Sigma}_{ij}$. As only relative constraints are given, there are 7 free degrees of freedom corresponding to the absolute placement of the keyframe graph. One keyframe pose must be fixed to be able to do the optimization.

As for pose tracking, weights may be assigned to individual constraints, which can be calculated by robust error functions such as the Huber norm (Eq. 4.22). The error is then minimized with the Gauss-Newton or Levenberg-Marquart methods as described in Sec. 2.4. The system developed in this thesis uses the g2o framework [32] for graph optimization.

5 Implementation on mobile devices

This chapter discusses the system's implementation with a focus on mobile devices, in particular modern smartphones with the Android operating system. The program has been tested on a Sony Xperia Z1 and a Samsung Galaxy Note 3 smartphone.

5.1 Specifics of mobile devices

5.1.1 ARM processors

While the processing power of mobile devices has increased rapidly in the last years, mobile processors based on the ARM architecture are generally still much slower than their desktop counterparts (as shown in Sec. 7.3). They especially differ in the performance of floating point operations: while on desktop CPUs operations with 32-bit or even 64-bit floating point numbers are not significantly slower than operations with integer operands, on current mobile CPUs there is still a difference.

For this reason, optimization plays a bigger role here. One possibility is to use fixed-point arithmetic, i.e. using integers for simulating floating-point operations with a fixed decimal point. Another possibility is use of the "NEON" [1] SIMD instruction set available on modern ARM processors which is discussed in section 5.4.

5.1.2 Rolling shutter cameras

Current smartphone cameras have a rolling shutter, which means that image rows or columns are read out sequentially, for example from top to bottom. This introduces distortions which are hard to correct as they depend on the readout order and duration of the image, the camera motion during the readout time, and the observed geometry. Fig. 5.1 shows an example of rolling shutter distortions.

During quick motion, this can have strong effects on the accuracy of stereo observations. While there exist methods to correctly model this in an off-line reconstruction setting [44], or to approximate it in real-time [26, 35], ignoring the rolling shutter still gives very good results in practice, and significantly saves computational time.

5.1.3 Android operating system

Currently, the two mobile operating systems with the largest user bases are Apple's iOS and Google's Android, followed by Microsoft's Windows Phone. The Android system, based on Linux, is generally the most open systems of these and was chosen as target platform for this thesis.

Android is Java-based, which is unsuitable for demanding applications such as computer vision. Therefore, Google introduced the Android native development kit (NDK)

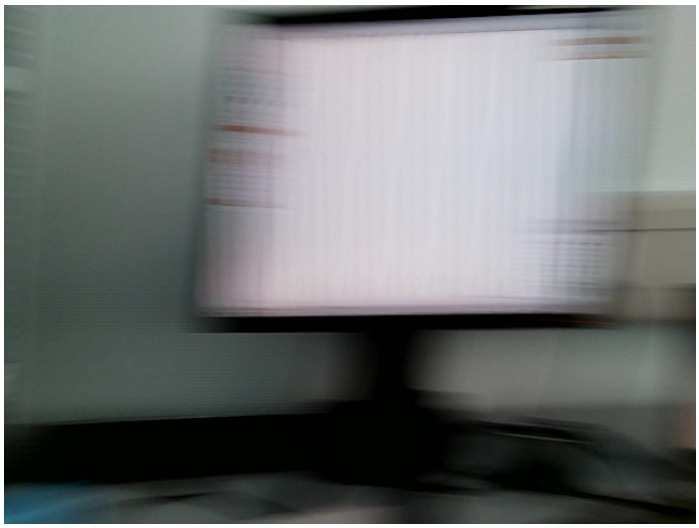


Figure 5.1: Example of rolling shutter artifacts: the image is taken by a Samsung Galaxy Note 3 smartphone in landscape orientation while quickly rotating the device to the right. As the lower rows of the image are read out later than the upper rows, rolling shutter artifacts are visible as the vertical borders of the computer monitor are skewed. In addition, strong motion blur is present due to the quick rotation.

[21] which allows to compile native C++ libraries which can be loaded and interacted with from Java. This is briefly described in Sec. 5.3.

5.2 Camera calibration

Camera calibration is the process of determining the intrinsic parameters of the camera model (described in Sec. 2.2), which are assumed to stay fixed, before running the actual system. For this thesis, the two existing calibration routines from PTAM [29] and OpenCV [40] have been evaluated.

5.2.1 PTAM camera calibrator

The calibration program from PTAM requires to take a number of photos of a black-and-white checkerboard pattern from different angles. It detects the checkerboard in the image with the following procedure:

1. Find all potential checkerboard corners in the image with a corner detector. Abort if not enough corners are found.
2. Pick the corner point closest to the image center as starting point.
3. With the dominant image derivatives around the current corner point, find adjacent corner points by following these directions and iterate this until the whole grid has been detected.

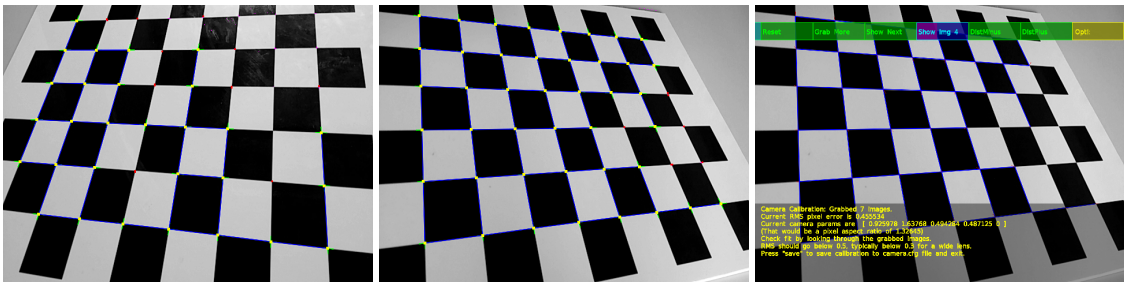


Figure 5.2: Example images from the camera calibration process with PTAM's camera calibrator. The left and middle image show detected corner points (yellow and red), derivative directions (green) and grid parts (blue) in individual images. The right image shows the projected grid (blue) in one image after optimization on an image batch.

Assuming a regular checkerboard, equations for projecting the checkerboard corners into the images depending on the camera parameters are stated. Setting in the observed corner positions leads to a system of projection constraints. From this, an error norm is determined and optimized to find the camera parameters.

PTAM uses a radial distortion model with one parameter, w, expressing the ratio f of the distorted normalized image coordinates radius to the radius of the un-distorted coordinates $\mathbf{x}$:

$$f = \begin{cases} 1 & \text{if } ||\mathbf{x}||_2 = 0 \vee w = 0 \\ \frac{\arctan(2||\mathbf{x}||_2 \tan(\frac{w}{2}))}{||\mathbf{x}||_2 w} & \text{else} \end{cases} \tag{5.1}$$

In practice, for the two tested smartphone cameras the optimal distortion parameter for this model was 0, effectively disabling distortion correction, while the estimated reprojection error was low. This may be a sign that smartphone manufacturers already correct or build the cameras in a way to minimize distortions, supposedly for aesthetic reasons. The model cannot express the remaining distortion. Fig. 5.2 shows example images from the calibration process.

5.2.2 OpenCV camera calibrator

The OpenCV library contains a calibration function which uses the same type of checkerboard marker as the PTAM calibrator, but may use a more sophisticated radial distortion model and in addition a model for tangential distortion, caused by lenses which are not aligned in parallel to the image plane. It is defined by 5 parameters k_1, k_2, k_3, p_1, p_2 and the following formula for radial correction of distorted normalized image coordinates $\mathbf{x}'$ to un-distorted normalized image coordinates $\mathbf{x}$:[1]

$$\mathbf{x} = \mathbf{x}' * (1 + k_1|\mathbf{x}'|_2^2 + k_2|\mathbf{x}'|_2^4 + k_3|\mathbf{x}'|_2^6) \tag{5.2}$$

[1] source: `http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html`

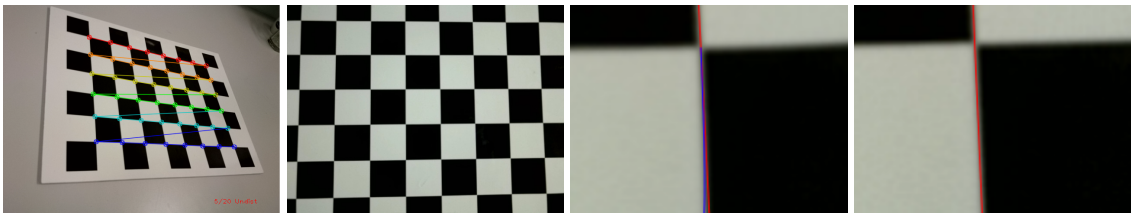


Figure 5.3: From left to right: checkerboard corners found by the OpenCV `findChessboardCorners` function, example of an undistorted image, close-up of the bottom right corner with straight line (red) and actual edge (blue), same close-up from the original image used for calibration without distortion artifacts.

For tangential correction, the following formula is used:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} + \begin{pmatrix} 2p_1 x'_1 x'_2 + p_2(|\mathbf{x}'|_2^2 + 2{x'_1}^2) \\ p_1(|\mathbf{x}'|_2^2 + 2{x'_2}^2) + 2p_2 x'_1 x'_2 \end{pmatrix} \tag{5.3}$$

However, in experiments with the two tested smartphone cameras, while correcting the central image regions well this lead to distortions in the image corners. The corner detection routine does not detect corners there, so sample points are missing in these regions. Fig. 5.3 shows an example of the distortion.

5.3 Software architecture

Each of the 4 program components which is shown as a column in Fig. 4.1 runs in a separate program thread. The application thus greatly benefits from quad-core processors as commonly used in modern high-end mobile devices. All threads are designed as event-based loops: when started, they first wait for a new item to process. This can be for example a new incoming video frame or a newly calculated frame pose. After finishing processing of this item, the thread goes back into its original state, waiting for the next item.

There is a single `Frame` class for handling all data corresponding to a video frame. Accessor methods to derived data such as image derivatives or downsampled image pyramid levels internally check whether this has been computed yet and if not, calculate it on demand. This way, other program components are able to use the data conveniently without explicitly computing it, while ensuring that the minimum amount of derived data is calculated to save processing time and memory.

The system is developed in `C++`. In order to use `C++` code in an Android app, there are several possibilities:

- Using a native "activity" to write the whole app in `C++`.
- Using a Java app with JNI [53] to interact with a `C++` library bundled with the app.
- Using a framework such as OpenCV or Qt, which come with default Java app frameworks which call the user-defined `C++` code with JNI.

As it is (at the time of writing) not possible in an officially supported way to access cameras from C++ code on Android, the second possibility was chosen so camera handling can easily be done with Java. The last possibility was not used because of the restrictions of a fixed Java framework.

The Java part thus consists of a small program frame which sets up display, does the camera handling and apart from that mainly just calls a method of the C++ library containing the actual program to start the app.

5.4 SIMD optimization with NEON

Many steps of the algorithms used in the system consist of loops with relatively simple iterations, lending themselves ideally to parallelization by Single-Instruction-Multiple-Data (SIMD) instruction sets. Support for such instruction sets differs between processor types: while for desktop CPUs different versions of the SSE instruction set have become a standard, for ARM processors as commonly used in Android devices there exists a similar, but not identical instruction set called NEON [1].

It operates on a separate set of 16 additional 128-bit processor registers. These registers can be addressed in various different ways, for example as 4-vectors of 32-bit `float`s or `int`s, or as two separate 2-vectors of these types. The instruction set contains basic operations such as vector addition, subtraction, element-wise multiplication as well as more specific operations such as element-wise multiplication followed by addition. Combining such operations which are common for linear algebra calculations into one instruction greatly accelerates them.

From C++ there are different ways of utilizing these special instructions:

- **Auto-vectorization** is a compiler feature which tries to determine suitable places and do vectorization automatically. However, the compiler's abilities are usually limited.
- **Intrinsics** are special methods which are translated into a single, specific instruction. They allow to mix C++ code and vectorized code directly.
- **Assembler** code allows to insert special instructions directly.

For this thesis, it was chosen to implement the parts to optimize directly in assembler, as compilers sometimes fail to produce optimal code. In addition, auto-vectorization was activated. Table 5.1 shows an overview of all operations which have been optimized with NEON and the speedups achieved. Table 7.2 in the results section shows an overall performance comparison.

Table 5.1: Speedups (ratio of time of C++ version with auto-vectorization to time of NEON-assembler version) for all methods which have been NEON-optimized. For measurement, the operations were performed many times in a loop with artificial data. All runs were done twice and the first discarded to be independent from caching effects.

Operation	Speedup
6×6 matrix-matrix addition $\mathbf{M}_{6\times6} + \mathbf{M}_{6\times6}$	2.23
6×6 matrix-vector-product addition $\mathbf{M}_{6\times6} + \mathbf{V}_6\mathbf{V}_6^T$	6.08
$\mathrm{SE}(3)$ tracking residual and buffer calculation	1.01
$\mathrm{SE}(3)$ tracking weight and residual calculation	5.02
$\mathrm{SE}(3)$ tracking Jacobian and parameter update calculation	1.51
Image downsampling to half size	1.12

6 Demo applications

To show the capabilities of the visual odometry system running on Android devices, two demo applications have been developed. The **virtual reality demo** (Sec. 6.1) shows a completely artificial 3D scene in which the virtual camera is moved by moving the mobile device in reality. The **augmented reality demo** (Sec. 6.2) shows 3D objects rendered into the device's camera image which are able to interact with the environment. The following sections present these demos and describe the most important steps in their implementation.

6.1 Virtual reality demo

Figure 6.1: **Left:** screenshot of VR demo as displayed on a smartphone. **Middle:** screenshot of stereo mode, to be used with virtual reality goggles such as the Durovis Dive [12], shown on the **right**. The smartphone is inserted as display at the front of the goggles.[1]

The virtual reality (VR) demo shows a 3D Sci-Fi scene in which the virtual camera is controlled by real motions of the Android device. Example screenshots are shown in Fig. 6.1.

This alone may already provide a base for e.g. computer games. An exciting application is the use together with 3D goggles for full-view virtual reality. For this, pose tracking is an important requirement to allow the user to look around, and to prevent motion sickness. At the time of writing, the most important device designed for this is the Oculus Rift [55], which must be connected to a PC and has only limited motion tracking via infrared markers. However there are 3D goggle frames for smartphones, in which an Android device is inserted as screen. While this has many drawbacks – e.g. the screen of those devices is not designed for extremely fast reactions and low-persistence, and the whole construction is usually shaky because of being compatible with devices of somewhat different sizes – it shows the potential of mobile VR and where the development could lead to.

[1] The models used in the 3D scene, "Sci-Fi level for FPS (Base Z2)" and "Girl Soldier & Blaster (Rigged)", are created by David Radford and licensed under the Attribution-ShareAlike 3.0 Unported license (`http://creativecommons.org/licenses/by-sa/3.0/`). The Durovis Dive photo is from [12].

VR goggles for smartphones work by splitting the screen into two parts such that the left half is only visible to the left eye and analogously the right half is only visible to the right eye. As the screen is only a few centimeters away from the eyes, there is a lens in front of each eye which allows it to focus on the screen. From the application side, the only requirement to make the app compatible with such devices is to render two images of the scene on the left and right half of the screen, corresponding to the views of the two eyes. The positional offset between the views should be adjustable by the user. Not only the eye distance differs between people, but the eyes may also be on slightly different heights. As a default, the average eye distance of adult humans is used which is 6.1cm [56], and zero height offset.

As monocular visual odometry cannot provide information about the scene's scale, the scaling of real to virtual movements must be chosen separately. For this thesis, no implementation of such a method was done – it uses a fixed factor based on the average scene distance at the time when the VR demo is started. Possible methods would be to estimate the scale based on the device's IMU [37], or to use a visual marker with known size, and an estimator as described in Sec. 4.5.2 for all scale samples.

6.2 Augmented reality demo

6.2.1 Overview

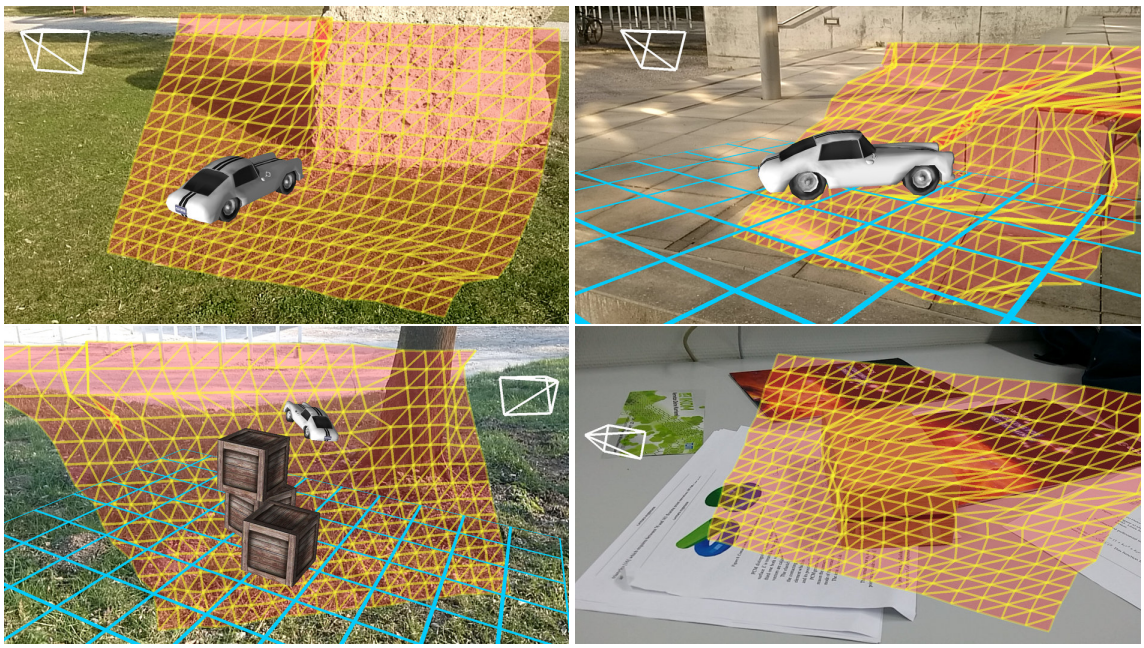


Figure 6.2: Example AR demo screenshots as displayed on a smartphone. The augmentations shown are: a steerable car and wood crates, a grid showing the ground plane (blue), the collision mesh used for physics simulation (yellow-red), and the camera pose at which the collision mesh was fixed (white).

The augmented reality (AR) demo [45] shows artificial 3D objects rendered into the live camera image of an Android device. In addition, it estimates a world model of the environment from the semi-dense depth maps to allow for collisions of simulated objects with real objects using a physics engine. Fig. 6.2 shows some example screenshots.

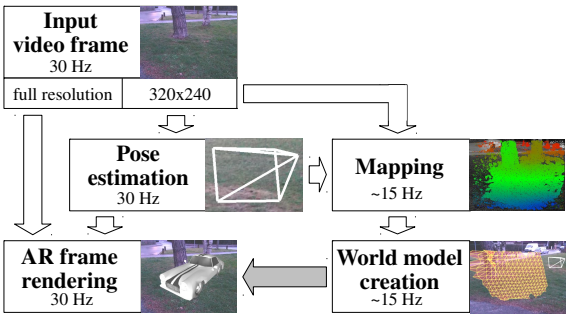


Figure 6.3: AR Processing Pipeline: Camera images are retrieved and displayed directly at full resolution. Simultaneously, a down-sampled (320×240) version is computed and processed in the visual odometry pipeline, to allow real-time tracking and mapping. The estimated pose of that frame, as well as the generated world model are then used to render artificial objects into the scene. The world model is updated asynchronously at a lower frequency. All components run in parallel.

The processing pipeline for this demo is shown in Fig. 6.3. The first special part for this demo is acquiring the camera image. While image processing must be done at a pixel resolution of 320×240 in order to run at 30 Hz, for display the highest resolution possible should be chosen. Another consideration is that for display the camera image should be available as OpenGL texture, while for processing it is required to have it in CPU memory.

Android's camera API provides a way to automatically deal with the latter requirement by both setting a preview callback with a buffer and a preview texture for the `android.hardware.Camera` object. This way the image is returned both in the CPU memory buffer in an easy-to-process format and also available on the GPU as a special texture which is bound to the `GL_TEXTURE_EXTERNAL_OES` target. It however remains to manually scale down the image on the CPU side to the desired resolution. The image should be blurred accordingly to prevent subsampling artifacts which degrade tracking quality.

The second addition of the AR demo to the processing pipeline is world model creation. The demo uses the free Bullet physics library [9] to simulate physical interactions between objects. For making the semi-dense depth map coming from the odometry component usable as collision mesh in the physics engine, it is converted to a triangle mesh. In addition, to cover up holes in the semi-dense depth map, this component estimates a ground plane. Sec. 6.2.2 describes this in detail.

Finally, the camera image and augmented objects are displayed by the rendering component. An important detail of this is proper camera-rendering synchronization which must ensure that objects rendered into a frame are drawn with the camera pose determined from the same frame. For Android, this can be done by calling `updateTexImage`

of `android.graphics.SurfaceTexture`, drawing the camera image and then waiting for the pose of the frame which was retrieved by the update before continuing with rendering the objects.

6.2.2 World model creation

Ground plane estimation

First, the plane normal is determined by low-pass filtering accelerometer measurements. This filters out quick movements to get the acceleration caused by gravity only, pointing downwards. Android provides a virtual "gravity sensor" which does this internally. To determine the plane height, the algorithm searches for the lowest height which is supported by a certain minimum number of depth map samples in a small height interval; the interval's size should be set to the size of the expected height uncertainty. The maximum height of all supporting samples is then taken as ground plane. This assures that small bumps, caused by inaccurate individual height estimates, are covered up with a smooth ground surface.

Collision mesh generation

The base for the collision mesh is a downsampled version of the current keyframe's semi-dense depth map at 20×15 pixels. Downsampling has several benefits: the map becomes significantly more dense, as a pixel at a lower resolution gets a valid depth hypothesis if at least one of the contained higher-resolution pixels has a depth hypothesis. Small bumps are smoothed out, and the map becomes much faster to process – both for processing it into a triangle mesh, and for using the mesh in a physics engine afterwards.

Depth maps are downsampled by factors of two, using a weighted average of the *inverse* depth. To account for the strong correlation between neighboring pixels, the (inverse) variances are averaged. With l being the index of the pyramid level, and $\Omega_{\mathbf{x}}$ denoting the set of valid pixels (i.e. with depth value) contained in pixel $\mathbf{x}$ at the next higher resolution, the downsampling formulas for inverse depth map D and inverse depth variances V are:

$$D_{l+1}(\mathbf{x}) := \frac{\sum_{\mathbf{x}' \in \Omega_{\mathbf{x}}} \frac{D_l(\mathbf{x}')}{V_l(\mathbf{x}')}}{\sum_{\mathbf{x}' \in \Omega_{\mathbf{x}}} \frac{1}{V_l(\mathbf{x}')}} \tag{6.1}$$

$$V_{l+1}(\mathbf{x}) := \frac{|\Omega_{\mathbf{x}}|}{\sum_{\mathbf{x}' \in \Omega_{\mathbf{x}}} \frac{1}{V_l(\mathbf{x}')}} \tag{6.2}$$

On the low-resolution depth map, a variational approach is applied to compute a fully dense, regularized depth map. As data term for a valid pixel the depth hypothesis at this pixel is used; for invalid pixels, the depth of the estimated ground plane π is used if it is visible at this pixel. The Huber norm of the inverse depth gradient is used as regularizer:

$$||\mathbf{x}||_\epsilon := \begin{cases} \frac{||\mathbf{x}||_2^2}{2\epsilon} & \text{if } ||\mathbf{x}||_2 < \epsilon \\ ||\mathbf{x}||_1 - \frac{\epsilon}{2} & \text{otherwise} \end{cases} \tag{6.3}$$

The Huber norm is a combination of a quadratic regularizer favouring smooth surfaces, and the total variation (TV), which allows sharp transitions at occluding edges. The combined energy to be minimized is hence given by

$$E(u) := \int_{\Omega_D} \frac{(u(\mathbf{x}) - D(\mathbf{x}))^2}{V(\mathbf{x})} d\mathbf{x} \tag{6.4}$$

$$+ \int_{\Omega \setminus \Omega_D} \frac{(u(\mathbf{x}) - \pi(\mathbf{x}))^2}{V_\pi} d\mathbf{x} \tag{6.5}$$

$$+ \alpha \int_{\Omega} ||\nabla u||_\epsilon d\mathbf{x} \tag{6.6}$$

This is a convex energy, and on the used resolution can be minimized globally and quickly using gradient descent. Afterwards, a triangle mesh is generated from the resulting depth map by interpreting the depth pixels as corners of a regular triangle grid. Some examples in different scenes are shown in Fig. 6.2. Fig. 6.4 shows examples of depth map inpainting.

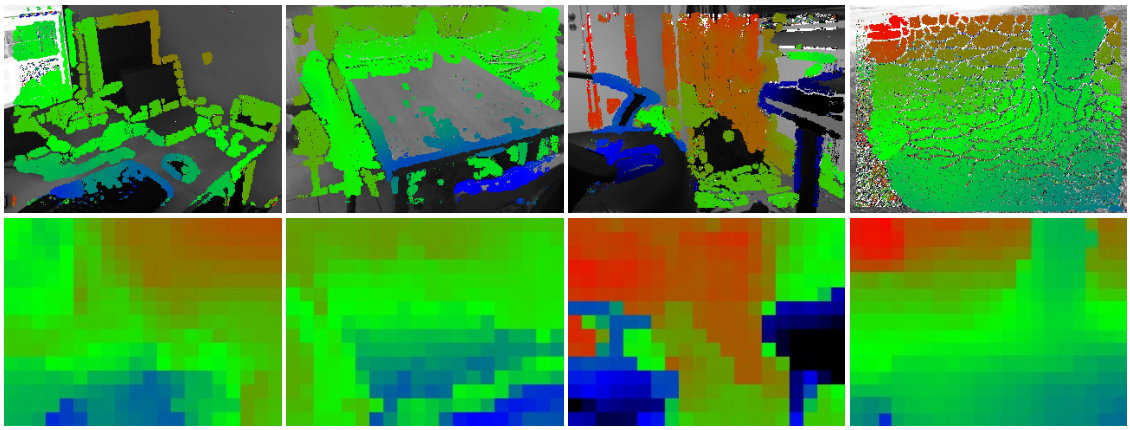


Figure 6.4: Examples of dense, regularized depth maps (bottom row). The top row shows corresponding full-resolution semi-dense depth maps.

A desirable effect of this approach is that the collision meshes naturally have a higher resolution in close-by than in far-away regions, where the un-projected mesh vertices are more tightly spaced and at the same time the depth map is more accurate.

As this approach creates a fully dense collision mesh, occlusions of simulated objects are made impossible as they collide instead of being occluded. While this is an artificial restriction, correct rendering of occlusions would require a dense and high resolution depth map. As this is not available, prevention of occlusions is done as alternative.

7 Results

This chapter presents an extensive evaluation of the system divided up into quantitative results (Sec. 7.1), qualitative results (Sec. 7.2) and performance measurements (Sec. 7.3).

7.1 Quantitative

7.1.1 TUM RGB-D benchmark

This benchmark [50] consists of a series of video sequences taken with a first-generation Microsoft Kinect camera [54]. Ground truth data for camera movement has been recorded with a motion capture system and is available for all sequences except a separate set of evaluation sequences.

The depth map of the first frame of a sequence is used for deterministic initialization of the SLAM system while also establishing the absolute scene scale. Depth maps for the rest of the frames are not used. The low quality color camera of the Kinect makes this a difficult benchmark for systems not using the depth data. Motion blur and rolling shutter degrade the image quality. Tab. 7.1 shows the absolute trajectory error reached by the system in comparison to similar approaches.

7.1.2 Sim(3) tracking convergence radius

The tracking convergence radius is hard to evaluate quantitatively, as it highly depends on the observed scene geometry and quality of the estimated depth map. For this reason it has been tested on two specific example scenes, shown in Fig. 7.1.

Table 7.1: Benchmark results on the TUM RGB-D dataset [50], and two simulated sequences from [22], measured as absolute trajectory RMSE (cm). For the system presented here, the number of keyframes created is also given. 'x' denotes tracking failure, '-' no available data. [27] and [14] use depth information provided by the Kinect, the others estimate depth themselves.

	Semi-dense SLAM (#KF)	Semi-dense VO [16]	PTAM [29]	DVO-SLAM [27]	Keypoint-based RGB-D SLAM [14]
fr2/desk	4.5 (116)	13.5	x	1.8	9.5
fr2/xyz	1.58 (38)	3.79	24.28	1.18	2.6
sim/freiburg	0.05 (39)	1.53	-	0.27	-
sim/slowmo	0.47 (12)	2.21	-	0.13	-

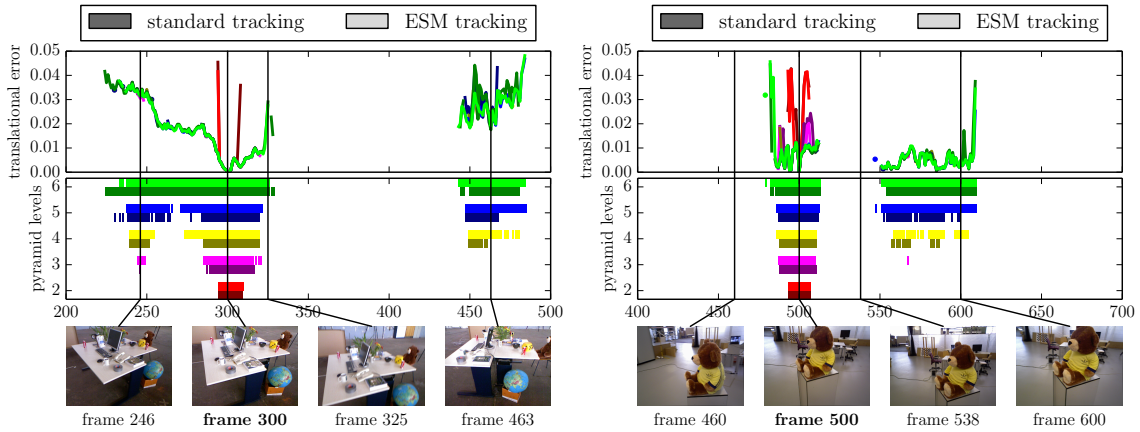


Figure 7.1: Sim(3) tracking evaluation for the fr2/desk and fr3/teddy scenes of [50], using ground-truth depth maps from the dataset, depending on use of Efficient-Second-Order-Minimization (ESM) tracking (brightness) and number of pyramid levels (color). Frames 300 respectively 500 were used to track all other frames using identity as initial pose estimate. **Top:** resulting tracking accuracy. **Middle:** convergence indicator (for tracking error below 0.05 m). **Bottom:** example images. The best convergence radius is achieved with the maximum number of pyramid images and ESM tracking. Tracking accuracy is however unaffected by this. Note that, for successive frames in videos, much better initial estimates are available.

7.1.3 Keyframe selection

The keyframe selection process has high influence on the resulting accuracy and requires a trade-off between accuracy and memory use. Fig. 7.2 shows an extensive evaluation of the two best keyframe selection methods presented in Sec. 4.3.2 regarding these aspects.

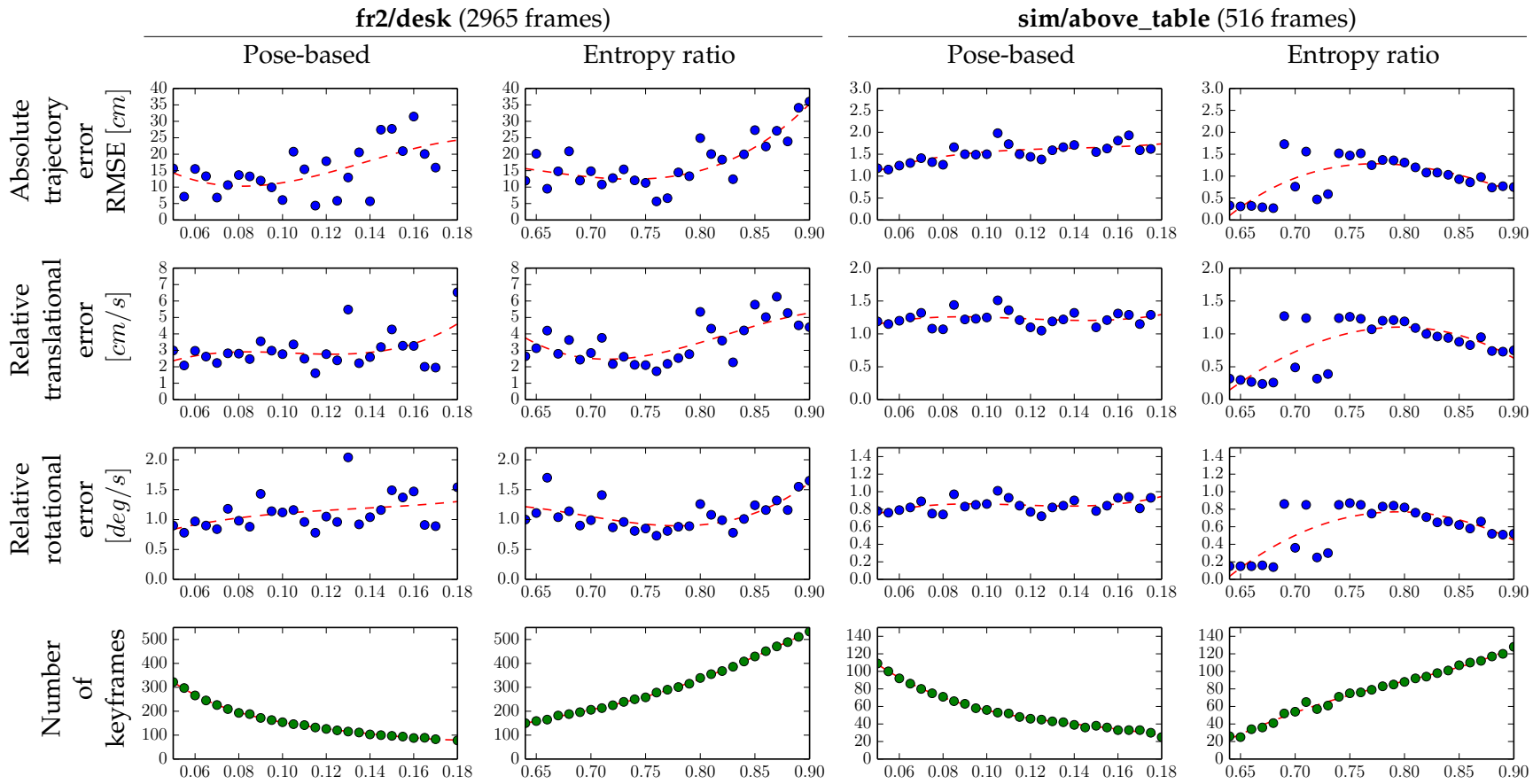


Figure 7.2: **Keyframe selection criterion evaluation:** accuracy results on the fr2/desk and sim/above_table sequences depending on keyframe selection criterion and its settings. While the individual runs (blue dots) exhibit strong randomness, trends can be seen as illustrated by the fitted polynomials (red lines). In addition to the accuracy, the memory requirements must be kept in mind: the last row shows the number of created keyframes. The parameters on the plots' x axes are the thresholds specific to the keyframe selection criteria, i.e. the distance or ratio threshold. Note that the sim/above_table scene contains an ill-conditioned subsequence which makes it challenging.

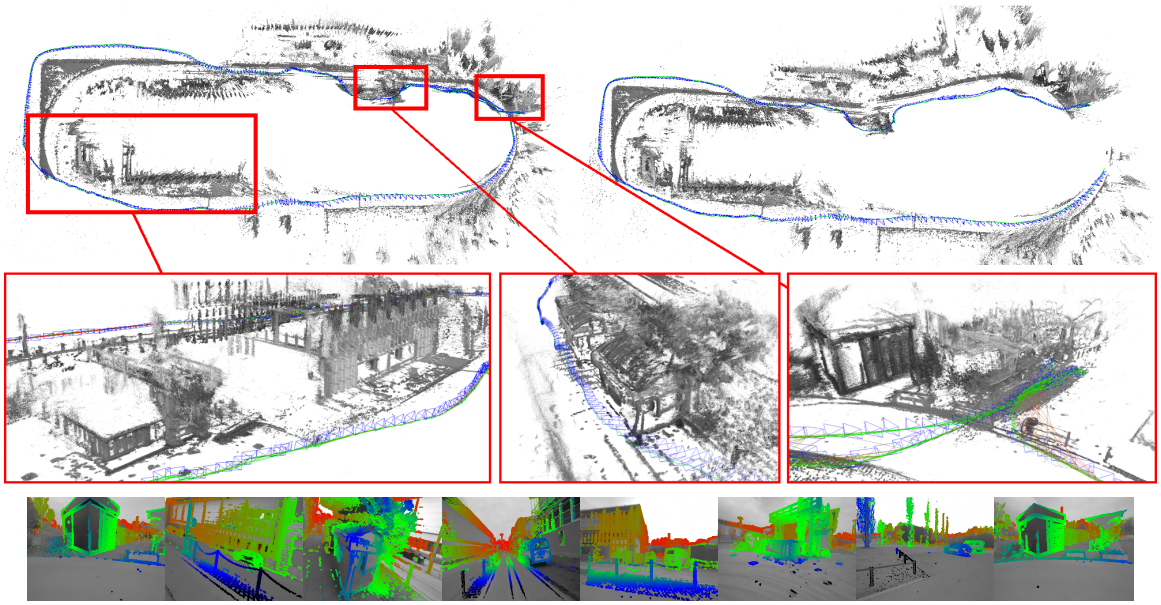


Figure 7.3: Reconstructed point cloud from a large trajectory, thresholded by estimated depth and inverse depth variance. The reconstruction state is shown before the big loop closure (top right) and after it (top left), together with some close-ups and semi-dense depth maps for selected keyframes.

7.2 Qualitative

The system has been tested qualitatively on multiple long sequences without available ground-truth recorded with an iDS uEye camera [23]. Images are from [15]. Fig. 7.3 shows a large trajectory around the building at WGS84 latitude 48.264073 longitude 11.669577, which is roughly 500 m long and takes 7 minutes. Fig. 7.5 shows reconstruction details of a sequence with large variations in average observed depth, recorded next to the above mentioned building; the loop closure in this sequence is shown in Fig. 7.4. Fig. 7.6 shows the graph of the average estimated scene depth for this sequence, and Fig. 7.7 shows another example of a reconstructed scene.

7.3 Performance

The hardware used for performance evaluations is a Sony Xperia Z1 smartphone with the quad-core Krait 400 CPU at 2.2 GHz. For comparison with a PC, a laptop with a dual-core Intel i7-3667U CPU with 2.0 GHz is used (with "Turbo Boost" up to 3.2 GHz).

The implementations on phone and PC differ in some aspects: the PC version is SSE-optimized, while the phone version is NEON-optimized. $\mathrm{Sim}(3)$ tracking used in constraint search only has a SSE-optimized code path. On the phone, appearance-based constraint search is not implemented and thus left out.

The evaluations have been done on the first 600 frames of the fr2/desk sequence, which are replayed at 30 Hz. All measured times are given in ms and specify the mean time of

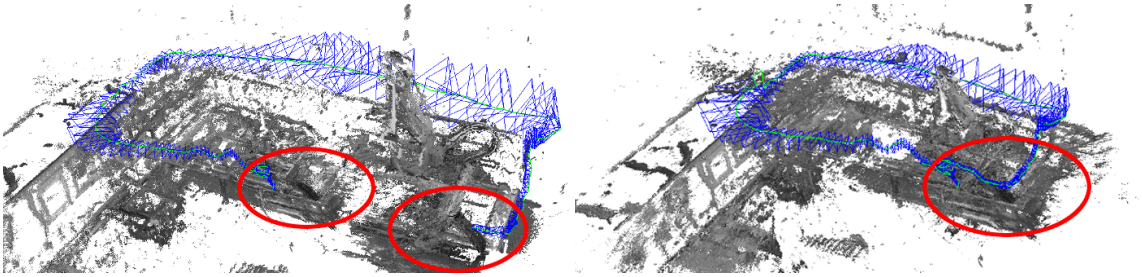


Figure 7.4: Loop closure in a trajectory with large scale variation. Before (left), geometry is duplicated in a different pose (including scale difference) due to the accumulated tracking error. Afterwards, it is aligned correctly.

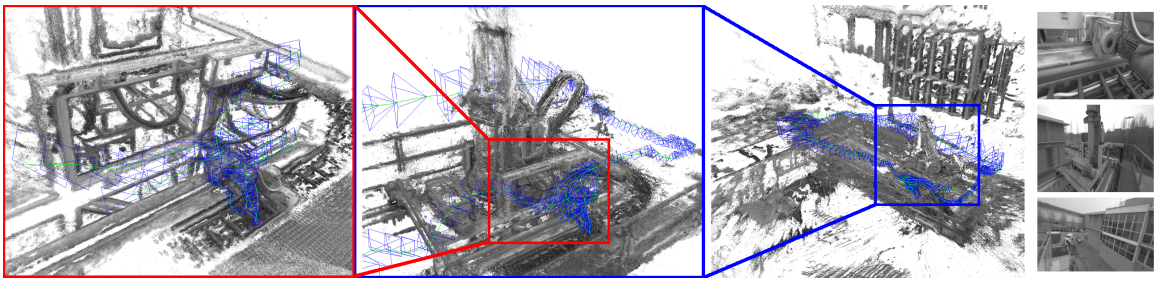


Figure 7.5: Reconstruction details of the same trajectory as in Fig. 7.4, showing that details at vastly different scales are reconstructed.

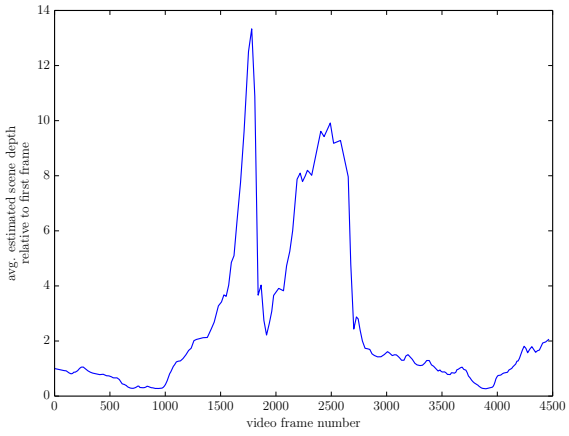


Figure 7.6: Graph of the average estimated scene depth for the trajectory in Fig. 7.4, showing that the approach can handle large variations in scale.

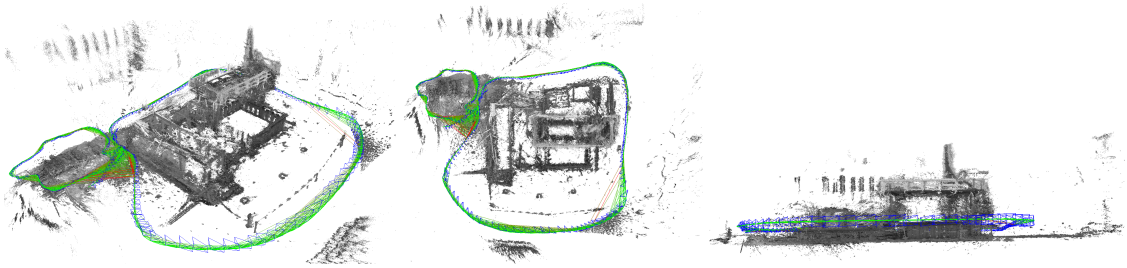


Figure 7.7: Another example of a reconstructed scene, seen from different views.

Table 7.2: Visual odometry performance comparison on Sony Xperia Z1. The mapping implementation is not NEON-assembler-optimized. Only the NEON version at 320×240 consistently runs with more than 30 Hz.

	C++ 320×240	ASM / NEON 640×480	ASM / NEON 320×240
Tracking	30.7 ($\pm$11.2)	39.2 ($\pm$15.9)	14.7 ($\pm$6.8)
Mapping	46.6 ($\pm$35.4)	184.4 ($\pm$123.3)	52.6 ($\pm$37.7)

one component iteration. Additionally, their standard deviation is given in brackets. The resolutions given are the mapping resolutions; tracking stops one pyramid level higher.

Tab. 7.2 (originally in [45]) evaluates the performance of the visual odometry component on the phone with regard to NEON optimizations and different resolutions. The visual odometry running mode is interesting for VR.

Tab. 7.3 evaluates the SLAM component with different resolutions on the phone and on the PC.

Table 7.3: SLAM performance comparison on phone and PC. Although the x86 CPU is just a dual-core processor, it runs the system much faster than the quad-core ARM CPU with similar frequency.

		Tracking	Mapping	Constraint S.	Optimization
Phone	320×240	12.0 ($\pm$5.0)	42.0 ($\pm$12.1)	619.4 ($\pm$217.9)	302.3 ($\pm$242.3)
	640×480	35.8 ($\pm$15.6)	148.6 ($\pm$52.5)	1733.6 ($\pm$779.9)	218.7 ($\pm$189.6)
PC	320×240	7.4 ($\pm$3.8)	35.8 ($\pm$9.8)	224.2 ($\pm$74.7)	208.7 ($\pm$170.2)
	640×480	22.0 ($\pm$10.7)	111.8 ($\pm$34.0)	872.7 ($\pm$301.3)	283.1 ($\pm$235.9)

8 Conclusion

In this thesis, a visual SLAM system was developed based on a visual odometry system which directly operates on images and estimated semi-dense depth maps without an intermediate representation such as keypoints. The system works in real-time not only on consumer laptops, but also on current-generation Android smartphones.

As the system's base the underlying odometry system tracks image poses and estimates semi-dense depth maps for camera frames. The SLAM system selects keyframes and adds them to a keyframe graph, which is then optimized based on $\mathrm{Sim}(3)$ constraints between keyframe pairs. Constraint candidates are searched for with complementary pose-based and appearance-based methods. Relative poses are determined with direct $\mathrm{Sim}(3)$ tracking or by estimating the scale in addition to $\mathrm{SE}(3)$ tracking. Constraint validation by reciprocal tracking discards false loop closures.

The system has been optimized for ARM processors which are common for Android devices by writing parts in assembler, utilizing NEON parallelization. Two demo applications have been developed for the Android platform. The virtual reality demo shows how the system can be used for pose tracking, which is a vital component for virtual reality goggles. The augmented reality demo demonstrates that it can be used for augmented reality devices, including use of the estimated depth maps for collisions of simulated objects with real world objects. This shows that the system is a good base for the currently emerging fields of virtual and augmented reality, among other applications.

8.1 Future work

Improvements

- **SLAM memory use:** Operation with the SLAM component enabled is currently limited by the CPU memory available, as all keyframes ever created are kept in memory until the program terminates. To mitigate this, if desired keyframes should be swapped out to disk when a memory limit is reached and they have not been accessed for some time. As an alternative or addition to this, the oldest keyframes should be marginalized out and deleted if there is no more space available at all.

- **Keyframe density and re-visit:** To allow location-based loop closure search to work well even if the camera stays in one region for a longer time, a maximum keyframe density (relative to the average scene depth) should be enforced. This also requires to be able to switch back to old keyframes for tracking. Note that both of these points are complicated by pose uncertainty: pose corrections due to loop closures may create a high keyframe density in an area without actively creating keyframes that are estimated to be close together, and keyframes which are thought to be close but are

not in reality must be avoided when evaluating old keyframes to use as tracking reference again.

- **Initialization:** The success of system initialization is currently dependent on luck, as the randomly initialized depth map may converge into an invalid state which represents a local optimum. It should be further investigated how to prevent this, e.g. by checking for a false optimum [25].
- **Tracking:** According to [3], the compositional Lucas-Kanade variants perform better than the additive variants. It could be investigated whether tracking could be changed to use a compositional variant.
- **Keyframe selection:** selection of new keyframes currently happens after a frame's pose is tracked. This does not give enough information to fulfill all criteria in Sec. 4.3.1. As another approach, the point in time at which keyframes are selected could be deferred to a later stage. Then additional information, most importantly the amount of successful stereo comparisons, would be available to determine whether a frame is suited or not. However, this would require decoupling from tracking: for fast movements, new keyframes currently need to be selected quickly to keep up a minimum amount of depth hypotheses. This cannot be deferred to a later stage.
- **Mapping self-similar regions:** Those are currently a problem for the stereo component. To counteract this, the stereo support region could be adaptively enlarged in case there is no clear optimum in stereo search. By taking a more descriptive image patch, false optimums would be reduced. On the other hand, this decreases the sharpness of the depth map at these points and requires more processing time. Good results also have been obtained by using a mixture model for the depth distribution which explicitly models outliers [19], instead of a normal distribution only.

General additions

- **World model:** Tracking of new frames is currently only done on the current keyframe. Parts of the scene which were visible in the keyframes before, but not in the latest are thus not used to track the position of new frames. This could be improved by tracking on a more complete world model instead of only the point cloud seen from the last keyframe, which might e.g. include depth pixels from multiple prior keyframes. As a result, this would not only reduce the need for small-scale loop closures, but also improve robustness: the current system may completely fail if e.g. the view is completely covered for a short time. With a world model, it may be possible to track the following frames again if the visible geometry is still present in the model, without using a separate re-localizer.
- **Depth observation redundancy:** Depth observations in a new keyframe are initialized with those of the previous keyframe. However, when observations are improved in the new frame, the old frame having a copy of the old observation is not updated. It could be investigated whether depth observations could be linked between frames instead of copied to reduce the amount of redundant and outdated depth observations in memory.

- **Re-localizer:** There are cases where even the best visual SLAM system imaginable would inevitably get lost, for example if the view is covered while the camera is moved to a different place. For these cases, a re-localizer is required. It may find out the camera position if the currently visible scene has been observed before, or build a completely new map first if the scene has not been observed yet, which may be merged with previous maps as soon as a common place is visited.
- **GPGPU programming:** The algorithm currently runs exclusively on the CPU. With GPGPU programming, certain parts could be moved to the GPU, accelerating it even more to be able to work with higher resolutions or frame rates.

Hardware additions

- **Use of smartphone sensors:** Smartphones provide a sensor platform with e.g. an accelerometer, gyroscope and GPS receiver being standard on today's devices. Use of these sensors to aid the visual SLAM could improve the system in different aspects:
  - Accelerometer and gyroscope could be used to estimate the absolute scale of the world.
  - The accelerometer could be used to remove pitch and roll drift.
  - The gyroscope could be used to determine good initial estimates for device rotation, and the accelerometer could be used to determine (weak) estimates for device translation.
  - GPS positions could be used outdoors to prevent long-term drift.

  However, delays and offsets between the different sensors must be modeled well for accurate processing.
- **Depth observations from heterogeneous sources:** With the addition of a second camera or depth camera, additional depth observations determined in a different way could be merged into the semi-dense depth maps. This would also allow to determine the absolute scene scale.

VR and AR demos

- **Minimum-latency VR:** For the VR demo being used with 3D goggles, it is very important to reduce the reaction time between head movements of the user and adaption of the rendered scene to an absolute minimum. Because of this, fusing the odometry pose with rotation obtained from the gyroscope (which works at a much higher frequency) as described in the hardware additions section would be particularly important for this demo.
- **VR and the real world:** A way to keep track of the real world while wearing the goggles would be an important addition. For this, the camera images could be shown directly to the user in some way; however, as most current smartphones contain only one back camera, only mono vision would be supported. Alternatively, the estimated depth information could be used to find out when an object comes too close to the user.

- **SLAM in the AR demo:** In the AR demo, the world model is always calculated from one keyframe only. It would greatly benefit the stability to use the full SLAM system for this demo and merge collision meshes determined from different keyframes. One caveat is that using the SLAM system instead of just the odometry system means that there is the possibility of sudden jumps in the estimated trajectory because of the effect of new loop closures. This could be mitigated by smoothly incorporating such pose offsets into the pose used for AR. If necessary, it could be coupled to the camera movement: while the camera stands (mostly) still, pose corrections would be very visible, so they should be avoided during this time; while the camera is moved, pose corrections could be done faster as they are not as obvious at this point.

Bibliography

[1] ARM. ARM NEON technology, `http://www.arm.com/products/processors/technologies/neon.php`.

[2] X. Armangué and J. Salvi. Overall view regarding fundamental matrix estimation. *Image and vision computing*, 21(2):205–220, 2003.

[3] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.

[5] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 943–948. IEEE, 2004.

[6] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. Brief: Computing a local binary descriptor very fast. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1281–1298, 2012.

[7] A. Comport, E. Malis, and P. Rives. Accurate quadri-focal tracking for robust 3d visual odometry. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2007.

[8] A. Concha and J. Civera. Using superpixels in monocular SLAM. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[9] E. Coumans et al. Bullet physics library, `http://bulletphysics.org`.

[10] M. Cummins and P. Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011.

[11] A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 29, 2007.

[12] Durovis. Durovis Dive, `https://www.durovis.com/dive.html`.

[13] E. Eade and T. Drummond. Edge landmarks in monocular SLAM. In *In Proc. British Machine Vision Conf*, 2006.

[14] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D slam system. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2012.

[15] J. Engel, T. Schöps, and D. Cremers. Large-scale direct monocular SLAM. Submitted to European Conference on Computer Vision (ECCV) 2014.

[16] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Intl. Conf. on Computer Vision (ICCV)*, 2013.

[17] J. Engel, J. Sturm, and D. Cremers. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems (RAS)*, 2014.

[18] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[19] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[20] A. Glover, W. Maddern, M. Warren, R. Stephanie, M. Milford, and G. Wyeth. OpenFABMAP: an open source toolbox for appearance-based loop closure detection. In *Intl. Conf. on Robotics and Automation (ICRA)*, pages 4730–4735, 2012.

[21] Google. Android NDK, `https://developer.android.com/tools/sdk/ndk`.

[22] A. Handa, R. Newcombe, A. Angeli, and A. Davison. Real-time camera tracking: When is high frame-rate best? In *European Conference on Computer Vision (ECCV)*, 2012.

[23] iDS imaging. `http://en.ids-imaging.com`.

[24] Junaio. `http://www.junaio.com`.

[25] O. Kahler and J. Denzler. Tracking and reconstruction in a combined optimization approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):387–401, 2012.

[26] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. *CSTR*, 1:2, 2011.

[27] C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. In *Intl. Conf. on Intelligent Robot Systems (IROS)*, 2013.

[28] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[29] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Intl. Symp. on Mixed and Augmented Reality (ISMAR)*, 2007.

[30] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *European Conference on Computer Vision (ECCV)*, 2008.

[31] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Intl. Symp. on Mixed and Augmented Reality (ISMAR)*, 2009.

[32] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[33] Layar. `https://www.layar.com`.

[34] J. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2003.

[35] M. Li, B. Kim, and A. Mourikis. Real-time motion estimation on a cellphone using inertial sensing and a rolling-shutter camera. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[36] M. Li and A. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *International Journal of Robotics Research*, 32:690–711, 2013.

[37] A. Martinelli. Vision and IMU data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. *Robotics, IEEE Transactions on*, 28(1):44–60, 2012.

[38] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Intl. Conf. on Computer Vision Theory and Application (VISSAPP)*, 2009.

[39] R. Newcombe, S. Lovegrove, and A. Davison. DTAM: Dense tracking and mapping in real-time. In *Intl. Conf. on Computer Vision (ICCV)*, 2011.

[40] OpenCV. `http://opencv.org`.

[41] M. Pizzoli, C. Forster, and D. Scaramuzza. REMODE: Probabilistic, monocular dense reconstruction in real time. In *Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[42] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*, 2006.

[43] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *Intl. Conf. on Computer Vision (ICCV)*, pages 2564–2571. IEEE, 2011.

[44] O. Saurer, K. Köser, J.-Y. Bouguet, and M. Pollefeys. Rolling shutter stereo. In *Intl. Conf. on Computer Vision (ICCV)*, 2013.

[45] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. Submitted to Intl. Symp. on Mixed and Augmented Reality (ISMAR) 2014.

[46] G. Silveira, E. Malis, and P. Rives. An efficient direct approach to visual SLAM. *Robotics, IEEE Transactions on*, 24(5):969–979, 2008.

[47] H. Strasdat. Local accuracy and global consistency for efficient visual SLAM, 2012.

[48] String. `http://www.poweredbystring.com`.

[49] J. Stuehmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Pattern Recognition (DAGM)*, 2010.

[50] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intl. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[51] P. Tanskanen, K. Kolev, L. Meier, F. C. Paulsen, O. Saurer, and M. Pollefeys. Live metric 3d reconstruction on mobile phones. In *Intl. Conf. on Computer Vision (ICCV)*, 2013.

[52] Wikipedia. Fisher information, `http://en.wikipedia.org/wiki/Fisher_information`.

[53] Wikipedia. Java Native Interface, `http://en.wikipedia.org/wiki/Java_Native_Interface`.

[54] Wikipedia. Kinect, `http://en.wikipedia.org/wiki/Kinect`.

[55] Wikipedia. Oculus Rift, `http://en.wikipedia.org/wiki/Oculus_Rift`.

[56] Wikipedia. Pupillary Distance, `http://en.wikipedia.org/wiki/Pupillary_distance`.

[57] Wikipedia. Unscented Transform, `http://en.wikipedia.org/wiki/Unscented_transform`.

[58] Wikitude. `http://www.wikitude.com`.