An Integrative Approach to Object Recognition in VSLAM

Diploma Thesis

submitted at
Dresden University of Technology
Department of Computer Science
Institute of Artificial Intelligence

by
Jan Funke

on
September 15th, 2010

Written under the supervision of
Prof. Dr. rer. nat. habil. Steffen Hölldobler

Task

Since the seminal work by Davison [11], the research area of VSLAM (*Visual Simultaneous Localisation and Mapping*), evolved rapidly. A lot of the fundamental tasks have been solved satisfactorily, e.g., data association, loop closing, hierarchical map maintenance, and efficient update algorithms providing statistical consistent map estimates [16, 27, 44].

Recently, attempts have been made to incorporate object recognition in VSLAM [5]. The idea is both to provide the user suitable information about the environment as well as improving the map and localisation estimates by using geometric constraints of the tracked objects.

However, object recognition is currently limited to the detection of planar and rectangular surfaces. The following achievements should be made in this thesis:

- Current techniques of planar surface recognition should be extended to handle objects of any kind.
- The object representation should be integrative, i.e., found objects should easily be integrated into the current map estimate and further improve localisation and mapping.
- Learning of the object's appearance should be performed automatically by providing a 3D model of the object to recognize.
- Multiple objects should be recognizable in one scene concurrently.
- The found objects should be made visible in the live video stream in an augmented reality fashion.

The object recognition system should be implemented on the basis of the VSLAM system by Tobias Pietzsch [30]. The implementation should be evaluated on real and synthetic images. For that, the evaluation framework *Slamdunk* [18] should be used.

Acknowledgements

I would like to thank Prof. Steffen Hölldobler for his agreement to supervise this thesis and the valuable recommendations concerning the formalisations used in this work. I am especially grateful for the many fruitful discussions and coffees I had with my supervisor Tobias Pietzsch. His work was the ideal starting point for my thesis and allowed me to focus on the implementation of my approach. Also, I would like to thank my proofreaders Christoph, Sharon, and Mr. Wiese for carefully reading large parts of this work. Last but not least, I would like to thank the KRR group and their associates for providing me a nice working place.

Notation

$\mathbb{Z}$	whole numbers (likewise: $\mathbb{R}$ for real numbers)
$\mathbb{Z}_{[a,b]}$	whole numbers between a and b: $\{a, a+1, ..., b\}$
$\mathbb{Z} \times \mathbb{Z}$	whole 2D coordinates
$\mathbb{Z}_{[a \times b]}$	shorthand for $\mathbb{Z}_{[0,a]} \times \mathbb{Z}_{[0,n]}$
$\mathbf{v}$	vector, position
$\tilde{\mathbf{v}}$	vector in homogeneous representation
$\boldsymbol{\rho}$	rotation; application to a vector: $\boldsymbol{\rho}(\mathbf{v})$
$\mathbf{M}$	matrix; special matrices: $\mathbf{0}_{[n \times m]}$ (zero filled matrix) and $\mathbf{1}_{[n \times m]}$ (identity matrix)
$\mathcal{F}$	frame, pose ; application to a vector: $\mathcal{F}(\mathbf{p})$
$\mathcal{F}^{\mathcal{G}}$	frame represented with respect to another frame $\mathcal{G}$
$\mathbf{v}^{\mathcal{F}}$	vector $\mathbf{v}$ represented with respect to frame $\mathcal{F}$
$\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$	derivation of h by $\mathbf{x}$
$\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\Big\vert_{\mathbf{z}}$	value of derivation of h by $\mathbf{x}$ at $\mathbf{z}$
$\boldsymbol{t}$, $\boldsymbol{T}$	feature template, set of feature templates
$\boldsymbol{s}$, $\boldsymbol{S}$	SIFT descriptor, set of SIFT descriptors
$\boldsymbol{o}$	object descriptor

Contents

1. Introduction

Since the first mention of the term "robot" by Karel Čapek in his science fiction play *Rossum's Universial Robots* in 1920, robots became an important part of our everyday life. Whenever tasks have to be performed that are dull, dirty, dangerous, or require precision beyond human abilities, robots proved to be very useful. However, most of the robots used today are neither autonomous nor intelligent.

An important step to actually build autonomous systems is the ability to explore an unknown environment. This task, which seems very easy for humans, poses some serious problems to robots. One of these problems is the question of how to build a map of the environment and localise the robot in it without any prior knowledge. This problem is widely referred to as SLAM, i.e., *simultaneous localisation and mapping*.

With a wide variety of hardware and different approaches the SLAM problem has been addressed since the eighties of the last century, starting with a seminal work by Smith et al. [35].

As soon as digital cameras became popular and their quality increased, one saw their potential to be a cheap sensor that delivers rich information about the environment. Of course, in contrast to other sensors like laser range finders, camera images could not be used as they are. A lot of effort has been put recently in the processing of the image data with the aim to robustly extract meaningful and distinctive measurements. In the late nineties, Andrew J. Davison investigated the use of active search to solve the SLAM problem for systems that have a camera as the only sensor. Some years later, the results have been published in and laid the foundation for today's research on *Visual SLAM* [11].

Since then, the research area of Visual SLAM evolved rapidly. A lot of the fundamental tasks have been solved satisfactorily, e.g., data association, loop closing, hierarchical map maintenance, and efficient update algorithms providing statistical consistent map estimates [16, 27, 44].

1.1. Motivation

In the recent past, the question of mapping was not only concerned about the *where*, i.e., where is the camera and where are the landmarks used for measurements; but also about the *what*. In general, a system is not that useful for further applications if its map estimate is sparse, containing only a few landmarks and no semantic information. It is desirable to know where certain entities of the world are: Rescue robots should be able to detect victims of earthquakes, first aid boxes, fire hoses, and so on. A household robot should be able to find the coffee machine and the refrigerator containing cool beer to be of any use at all. A robotic museum guide should be able to know where a certain painting is located.

On the other hand, an incorporation of semantic information in a Visual SLAM system does not just help autonomous systems. Wearable cameras could allow for augmented reality applications. A smartphone with an interactive city guide could imprint

the name of sights that are currently visible. Visually disabled persons could use a wearable camera system to navigate in an unknown building and find useful information like the location of exit doors via auditive feedback from the system. Of course, one could also think of pure entertainment applications: Imagine a real world jump-and-run game on your mobile, in which your protagonist has to find vending machines to refresh himself.

1.2. Related Work

The achievements made with this work touch three partly overlapping research areas in the context of Visual SLAM: map maintenance, relocation, and object recognition.

1.2.1. Map Maintenance

The size of the map remains to be a problem even with state of the art SLAM systems. A popular technique for Visual SLAM is to use the EKF (see section 2.1.3) for the incorporation of measurements. However, the complexity of the map update grows at least quadratically with the size of the state and therefore the map.

Reducing the map size while preserving meaningful information is one of the research problems in SLAM. For a mobile robot equipped with a 2D laser range finder Thrun et al. proposed to fit geometric primitives, in particular planar rectangular surfaces, to the map [39]. This helps to reduce the number of landmarks and provides a dense map. In experiments the authors found by visual inspection that the accuracy of the tracking increased with their method. The method for fitting the planar surfaces is a modified expectation maximisation algorithm that runs on top of the SLAM system and alters the map. The modifications have been made to ensure the real time performance of the system. However, no semantic information is achieved with this approach.

Rottmann et al. decided to learn a classifier for certain areas of a map to gain some semantic information [33]. Based on a laser range finder and a camera, they use boosting to decide whether the mobile robot is in a certain type of room of a previously unknown indoor environment. This classification is independent from the SLAM process and just annotates the map. However, once an annotated map has been established the robot can use the knowledge about the type of room to perform a more robust relocation.

1.2.2. Relocation

Due to unexpected movement, noise, violated world assumptions, or some other external disturbances it might get necessary for a SLAM system to relocate itself in a known environment. This is especially a problem for Visual SLAM systems tracking point features since these systems need to have a good estimate of the current state to perform measurements at all. Therefore, good relocation strategies for Visual SLAM are crucial for robust systems. Furthermore, a relocation strategy enables the system to detect already visited areas and allows for *loop closing*.

Eade et al. propose to tackle both problems by partitioning the landmarks in slightly overlapping sets [13]. These sets are arranged as nodes in a graph structure with edges representing the transformations between neighboring nodes. Once the system is considered lost, a new and not yet connected node is built that contains newly initialised landmarks. Via fitting of a bag-of-words appearance model on the current and some stored camera images, the most similar nodes to the current one are sought in the graph.

Using MLESAC and a simplified version of SIFT descriptors, the camera is localised relative to the candidate nodes, thus introducing candidate edges in the graph. On successful measurements of the expected landmarks, at most one of the candidate edges becomes a normal edge. During normal operation, the same technique serves to identify already visited nodes and therefore helps to close loops.

Williams et al. follow a different strategy to perform relocation [45]. For each point feature that is initialised during normal SLAM operation a randomised tree classifier is learnt online. The classifier is particularly robust against rotation, scale, and illumination changes [24]. Based on this classifier, potential point correspondences are sought once the system is considered lost. Using RANSAC the pose of the camera is estimated and checked via measurement attempts of other point features. On success, i.e., if enough point features have been measured at their predicted position, the camera is getting relocated and the measurements are incorporated immediately to refine the pose estimate.

1.2.3. Object Recognition

When it comes to real time requirements, the idea to track objects on video sequences instead of recognising them on each individual frame is quite prominent in the computer vision community.

Although not integrated in a SLAM process, Gordon et al. proposed to track stationary objects with a hand-held camera in real time based on an exhaustive object descriptor obtained from sample shots only [19]. For that, SIFT descriptors are extracted from the training images, matches are made up, and their 3D pose is recovered using bundle adjustment. The 3D object descriptor obtained this way is compared to SIFT descriptors extracted from the current camera image. Possible matches are filtered using RANSAC to provide a camera pose estimate. At this, a regularisation term is introduced which penalises large camera movement between consecutive frames.

An approach for online object learning and recognition in Visual SLAM is given by Lee et al. [23]. For their EKF-based system they detect objects as agglomerations of SIFT key points that are additionally enclosed by a contour. The center point of all SIFT descriptors belonging to one object is considered a measurement of that object. For that, affine transformations are taken into account to compensate viewpoint changes. However, it remains questionable whether the found regions actually constitute individual objects. Furthermore, the approach only works for planar objects.

In the work of Ramos et al. object categories are learnt in an off-line process and recognised during the SLAM run [31]. For that, a combination of a 2D laser range finder and a camera is used to recognise landmarks by their appearance and position. An object class descriptor is learnt as a generative Gaussian mixture model of landmark appearances. During the SLAM run the presence of an object is recognised by discontinuities in the laser range data. From the camera image patches are extracted accordingly. Via the estimate of the global patch positions given by the EKF and the appearance model learnt off-line, the system builds a probability density function for the observed object appearance and position. This probability density function is compared by means of the Kullback-Leibler divergence against the probability density functions of already mapped objects. If the distance is small enough, the closest object is considered remeasured and its probability density function is refined. Otherwise, a new object is added to the map. However, this approach does not allow to recognise and track individual objects.

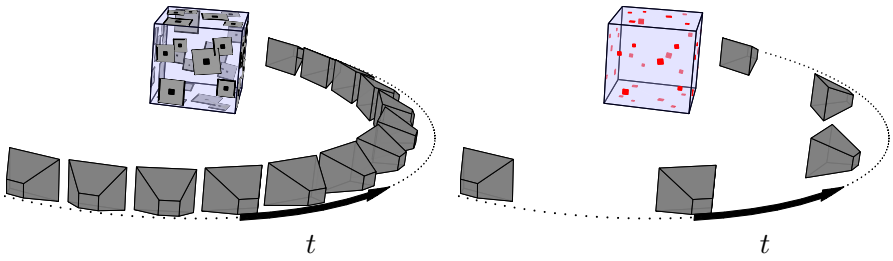
Recently, Wangsiripitak et al. extended a monoSLAM system with a RAPiD object tracker [43]. Both processes are largely independent from each other. The only benefit for the SLAM system is that found objects indicate point features that are currently occluded or that lie on the object and are considered movable. These point features are not getting measured, thus stabilising the SLAM process.

The approach that is closest to ours has been published by Castle et al. while this work was in progress [6]. The authors use a database of object descriptors obtained via SIFT descriptor extraction as well. During the SLAM run, object pose measurements are performed based on SIFT feature correspondences and RANSAC. In contrast to our approach, however, the system is currently limited to planar rectangular objects. The pose measurements are integrated in the EKF update equations to refine the camera pose. For that, the authors use a similar method of delayed measurement integration as we do: Since the object recognition takes a longer time to deliver a measurement than usual active vision measurements, the EKF performed some updates before a pose measurement of a selected camera image is available. In both works, the state is augmented with the camera pose at the time of selection of the camera image for object recognition. Through correlation of this pose to other entities in the state its pose gets updated and refined until the pose measurement is available. Castle et al. consider the refined camera pose for the actual measurement to retrieve the world coordinates of the object. In contrast to that, our object pose measurements are given relative to the observing camera and are completely correlated with its uncertainty. The objects in the work of Castle et al. are represented in the state via three points, which uniquely determine the object pose. We use a more compact description by inserting the position and rotation of the object into the state. Another difference in our approach is that we use regular point feature templates besides SIFT descriptors for our object descriptors. As we shall show in our experiments, this makes a big difference to the tracking accuracy compared to just using SIFT descriptors.

1.3. Approach

With this work we will show how object recognition can be integrated in Visual SLAM systems to achieve the following goals:

- **Add semantic information to the map.** We extend the point feature map of the used Visual SLAM system to contain objects. This way, the system does not just track random point features but knows about the relative poses of objects to the camera.
- **Reduce the map size and therefore the filter update speed.** We argue that the tracking of a single object is already sufficient to get a good camera pose estimate. In contrast to using several point features for the same purpose, an object consumes less state space. This is especially beneficial during the update step of the EKF filter.
- **Reduce the error in the camera pose estimate.** Whenever an object was recognised and added to the map it provides a lot of valuable information about the current camera pose.



(a) object tracking (b) object recognition

Figure 1.1.: Visualisation of our approach. Objects are processed in two ways: The object tracking (left image) runs at the camera frame rate and provides fine granularity measurements of the object pose. For that, feature templates (gray squares) are sought in the camera images. The object recognition (left image) delivers measurements less often but allows to keep global consistency. Based on SIFT descriptors (red points), the pose of the object is estimated and incorporated in the map update. In contrast to the tracking, the recognition does not rely on a good state estimate and thus helps to increase the robustness of the whole system.

- **Relocate the camera whenever the system lost track.** The object recognition does not rely on a good state estimate. Whenever the system is lost, a single object pose measurement can be used to relocate the camera.

To achieve this goals we propose to separate between object *tracking* and *recognition*. The object tracking is about the repeated measurement of an object that was already added to the map. This measurement is very similar to the common measurement of point features via feature templates and active search. Thus, the object tracking can be performed at the rate in which the camera delivers images. The measurements obtained this way are integrated in the standard measurement process of the Visual SLAM system we used for our implementation. The model behind this measurements is explained in detail in chapter 4.

The object *recognition* runs separately but not independently of the tracking in an own thread. The task of the recognition is to initialise objects in the map that have been observed for the first time, to refine the pose estimates of already initialised objects, and to relocate the camera in case the system lost track. The pose estimates obtained from the recognition thread cannot be provided at the same rate in which the camera delivers images. Regardless of that, we consider the pose estimates as measurements in an EKF framework and integrate them as soon as they become available with a technique called *delayed measurements*. For that, we extended the object model as described in chapter 5.

For both the tracking and recognition we need exhaustive object descriptors. In our approach, these descriptors are learnt from handmade 3D models of the objects.

For the implementation of our object model we used a point feature based Visual SLAM system by Tobias Pietzsch which we will briefly introduce in section 3.4.

1.4. Outline

This work is structured as follows: In chapter 2 we give an introduction to the general concepts that are necessary to follow our approach. In the section about temporal inference we will derive and discuss the iterated extended Kalman filter (IEKF) as a special case of the Bayes filter. In the second part of this chapter we will briefly introduce the reader to multiple view geometry.

In chapter 3 we will explain the SLAM problem. In particular, we will provide a formal framework for Visual SLAM using point features and the IEKF. In this chapter we will clarify the notation used throughout this thesis. Further, we will discuss how measurements can be retrieved from camera images and used to update the knowledge about the map.

The main part of this thesis can be found in the chapters 4 and 5, which are about the object model and the extension for object recognition, respectively. In these chapters we extend and refine the formal framework provided earlier for our specific model. We will explain how object descriptors are learnt, what measurements in our model are, how they are related to the state, and how to integrate them in the IEKF update. In chapter 5 we will also see how the object recognition can be used to relocate the camera.

Experimental results obtained with our approach on real and artificial sequences can be found in chapter 6. There we will describe the setup for our experiments and discuss the outcome as well. With a short discussion of our approach we conclude in chapter 7.

In the appendices A-D we will give an introduction to vector calculus, probability theory, derive the Jacobians used in the main part, and provide the plots of our experiments.

2. Basics

2.1. Temporal Inference

In this thesis we are considering agents that interact with their environment (as opposed to agents that try to solve Sudokus or perform similar static tasks). This interaction is carried out in two ways: *Measurements* are made to get information about the current state of the world, and *actions* are performed to change the current state of the world. Whenever an agent interacts with its environment, there are two major points to notice:

- Each action performed by the agent takes some *time*. The environment encounters some changes as effect of these actions. Usually, the time needed to manipulate the environment is not expressed explicitly. Instead, we think of the world as going from one state to another.
- Each measurement as well as the outcome of an action underlies *uncertainty*. Measurements are imprecise due to noise or unexpected behavior. The result of an action may not always be the desired outcome.

The first point basically describes a quantization of time. Assuming the *state* of the world (or at least the part we are interested in) is expressed by the vector $\mathbf{x}$, we call $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ two subsequent states between which an action $\mathbf{u}_t$ was performed.

In a very simple world, $\mathbf{x}$ might be an agent's position and orientation, represented by a state vector $\mathbf{x}_t = (p_{x,t}, p_{y,t}, \theta_t)^T$. An action $\mathbf{u}_t$ in this world could be "move forward by one meter". However, the state of the world might not change exactly in the intentioned way. In fact, it is a very hard task for an agent to move exactly by a specified amount due to wheel slipping and other disturbances.

This leads to the second mentioned point. An agent's representation of the current state of the world cannot be exact - it needs to incorporate uncertainty. This is achieved by not just considering a single state $\mathbf{x}$ as the agent's representation of the world, but a *pdf* [1] $p(\mathbf{x})$ over all possible states. This pdf is commonly called the agent's *belief*.

Obviously, the performance of an action $\mathbf{u}_t$ changes the belief. However, with just actions performed the uncertainty would increase, making the agent's belief more and more uniform. To compensate this effect the agent needs to make measurements $\mathbf{z}_t$ to regain some certainty about its belief. Even though measurements may underly uncertainty as well they help improving an agent's belief by making some hypotheses less likely than others.

A measurement could be, for example, the position of a known landmark in a camera image. Taking into account this information gives some evidence about the agent's position and hopefully compensates for the uncertainty introduced by the egomotion of the agent.

[1] pdf; *probability density function*, see definition B.2 in appendix B, page 85

2.1.1. Bayes Filter

As stated above, the pdf of the state of the world is called the agent's belief. After an initialisation the belief changes accordingly to the performed actions and encountered measurements. Usually, this change is modelled in two steps: a *prediction* step based on the performed actions, and a *correction* step taking into account the measurements. Both updates are modelled as conditional probabilities:

$$\overline{\text{bel}}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}) \qquad \text{prediction} \tag{2.1}$$

$$\text{bel}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t}) \qquad \text{correction} \tag{2.2}$$

Ideally, the definition of the pdf for a certain state $\mathbf{x}_t$ depends on all previously performed actions and taken measurements. However, a straightforward implementation of these functions would be very hard or even impossible.

Firstly, this approach requires keeping track of all past events. This can be a problem in environments in which memory is limited but agents are expected to keep on working for a long time. Even more important than the memory limitation is the increase in computational complexity.

Secondly, modelling the conditional pdfs might get very hard. Intuitively, it is possible that there is a dependence between very early measurements (e.g., someone left the room leaving a cake on the table) and the present state (e.g., the room is crowded). However, explicitly describing these dependencies might be intractable.

These difficulties can be avoided if the tracked state is considered a *complete state*. A complete state describes every aspect of the world that is necessary to predict the next state. Additional information like past measurements would not gain any benefit on a complete state, since this information is already present implicitly. The complete state assumption generalizes to the *Markov assumption* which is very common in graphical probabilistic models .

Definition 2.1: (Complete State Assumption) Assuming a complete state, it is possible to give a *state evolution pdf* that does only depend on the previous state and the most recent actions and measurements:

$$p(\mathbf{x}_t|\mathbf{x}_{0:t-1}, \mathbf{z}_{1:t-1}, \mathbf{u}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{u}_t) \tag{2.3}$$

Note that the state evolution probability is not equal to the belief because it is additionally conditioned on the previous states, which we don't know exactly. We will see how these two pdfs relate to each other soon.

Of course, having a complete state representation is impossible because of the wide range of factors that influence future states. Making the assumption to have a complete state is therefore always a violation that has to be handled with care. Putting more suitable information in the state representation eases this violation and improves the prediction performance but increases computational complexity. What suitable information exactly is depends highly on the agent's domain. In a mobile robot environment, for example, it might be useful to incorporate the current speed and acceleration of the vehicle in the state representation as well.

In the remainder of this thesis we will assume to have a complete state representation. We will see how this can be used to update the agent's belief of a state by considering the previous belief and most recent actions and measurements only.

The most general framework to accomplish this task is the *Bayes filter*. This filter (and all it's concrete implementations mentioned later) requires three pdfs: An initial belief $\mathrm{bel}(\mathbf{x}_0) = p(\mathbf{x}_0)$, a measurement probability $p(\mathbf{z}_t|\mathbf{x}_t)$, and a state transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$. Using these functions it is possible to give a recursive update scheme for the current belief:

Definition 2.2: (Bayes Filter) Given an initial state estimate $p(\mathbf{x}_0)$, a measurement probability $p(\mathbf{z}_t|\mathbf{x}_t)$, and a state transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$, the Bayes filter provides update equations for the current belief $\mathrm{bel}(\mathbf{x}_t)$ with respect to an action $\mathbf{u}_t$ and measurements $\mathbf{z}_t$:

$$\mathrm{bel}(\mathbf{x}_0) = p(\mathbf{x}_0) \qquad \text{initialisation} \tag{2.4}$$

$$\overline{\mathrm{bel}}(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)\mathrm{bel}(\mathbf{x}_{t-1})d\mathbf{x}_{t-1} \qquad \text{prediction} \tag{2.5}$$

$$\mathrm{bel}(\mathbf{x}_t) = \eta p(\mathbf{z}_t|\mathbf{x}_t)\overline{\mathrm{bel}}(\mathbf{x}_t) \qquad \text{correction} \tag{2.6}$$

Note that the previous belief $\mathrm{bel}(\mathbf{x}_{t-1})$ is required to compute the current belief, making this update scheme a recursive one.

Theorem 2.1: (Correctness of the Bayes Filter) Assuming a complete state representation, the Bayes filter is correct, i.e.,

$$\overline{\mathrm{bel}}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}) \tag{2.7}$$

$$\mathrm{bel}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t}) \tag{2.8}$$

Proof. We proof the correctness of this theorem by showing that the equations 2.5 and 2.6 of the Bayes filter can be derived from the definition of the belief functions by using the complete state assumption only.

Equation 2.5 follows directly by application of the marginalisation rule (see B.7) and the complete state assumption:

$$\begin{aligned}
\overline{\mathrm{bel}}(\mathbf{x}_t) &= p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t-1}) \\
&= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1})p(\mathbf{x}_{t-1}|\mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1})d\mathbf{x}_{t-1} && \text{marginalisation} \\
&= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)p(\mathbf{x}_{t-1}|\mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1})d\mathbf{x}_{t-1} && \text{complete state} \\
&= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)\mathrm{bel}(\mathbf{x}_{t-1})d\mathbf{x}_{t-1} && \text{definition of } \mathrm{bel}(\mathbf{x}_t)
\end{aligned}$$

Equation 2.6 follows from rewriting the definition of the correction step according to the Bayes rule (see B.8) and applying the complete state assumption as well:

$$\begin{aligned}
\mathrm{bel}(\mathbf{x}_t) &= p(\mathbf{x}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t}) \\
&= \frac{p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1})p(\mathbf{x}_t|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t})} && \text{Bayes rule} \\
&= \eta p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{u}_{1:t}, \mathbf{z}_{1:t-1})p(\mathbf{x}_t|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) && \text{normalization constant} \\
&= \eta p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t}) && \text{complete state assumption} \\
&= \eta p(\mathbf{z}_t|\mathbf{x}_t)\overline{\mathrm{bel}}(\mathbf{x}_{t-1}) && \text{definition of } \overline{\mathrm{bel}}(\mathbf{x}_t)
\end{aligned}$$

□

Thus, the Bayes filter provides a very basic recursive update scheme for beliefs. However, in most cases it is not applicable directly due to the integration in the prediction step. The following sections will introduce concrete implementations of the Bayes filter that allow for closed form solutions to the prediction and correction steps.

2.1.2. Kalman Filter

The Kalman filter (invented by Swerling in 1958 and Kalman in 1960 [22, 37]) is an attempt to formulate the update equations of the Bayes filter in a closed fashion. The key concept is to assume that the belief can be represented by a *multivariate normal distribution*, or *Gaussian distribution*.

Definition 2.3: (Multivariate Normal Distribution) A random variable X is said to be *multivariate normal distributed* or *Gaussian distributed* with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ if:

$$p(\mathbf{x}) = \det(2\pi\boldsymbol{\Sigma})^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right] \tag{2.9}$$

$$\stackrel{\text{def}}{=} \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.10}$$

Hence, for each time step t we can represent our belief as $\mathrm{bel}(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. This model assumption fits well if the world's state is known with a region of uncertainty around the true state. On the other hand, this model is not capable to express more than one different hypotheses with equal probability because it has just one global optimum which is at the mean μ of the distribution.

Consider for example a room which is point-symmetric by 180° and an agent who is trying to locate itself in it. Until the very end, the agent would not know on which side of the room it is, although it might know very precisely its location with respect to whichever side it is in. Therefore it is necessary for the agent to express both hypotheses with the same probability. This can not be done using a single Gaussian distribution.

Linear Gaussian System

The assumption that the belief at every time-step t is represented by a Gaussian distribution, i.e., parameters $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$, poses some requirements on the update equations of the Bayes filter: The state transition probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$ and the measurement

probability $p(\mathbf{z}_t|\mathbf{x}_t)$ need to ensure that the updated belief $\mathrm{bel}(\mathbf{x}_t)$ (given $\mathrm{bel}(\mathbf{x}_{t-1})$, $\mathbf{u}_t$ and $\mathbf{z}_t$) is also Gaussian distributed. This is the case if the system under consideration is a *linear Gaussian system*, i.e., the mentioned probability density functions are linear functions of their arguments with added Gaussian noise.

Definition 2.4: (Linear Gaussian System) A system is called a *linear Gaussian system* if its state transition function and its measurement function can be written as:

$$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\epsilon}_t \qquad \text{state transition} \tag{2.11}$$

$$\mathbf{z}_t = \mathbf{C}_t\mathbf{x}_t + \boldsymbol{\delta}_t \qquad \text{measurement} \tag{2.12}$$

where $\boldsymbol{\epsilon}$ and $\boldsymbol{\delta}$ are zero mean Gaussian distributed random variables, i.e.,

$$p(\boldsymbol{\epsilon}_t) = \mathcal{N}(0, \mathbf{R}_t) \tag{2.13}$$

$$p(\boldsymbol{\delta}_t) = \mathcal{N}(0, \mathbf{Q}_t) \tag{2.14}$$

and $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are $n \times n$, $n \times m$, and $k \times n$ matrices, respectively (with n being the size of the state vector, m the size of the action vector, and k the size of the measurement vector).

According to the definitions above we can easily convince ourselves that the state transition pdf and the measurement pdf are also Gaussian distributed now:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) = \mathcal{N}(\mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t, \mathbf{R}_t) \tag{2.15}$$

$$p(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{C}_t\mathbf{x}_t, \mathbf{Q}_t) \tag{2.16}$$

Given these pdfs we can now derive the update equations of the Kalman filter as a special case of the Bayes filter.

Update Equations

If all of the above mentioned assumptions are made the update equations of the Kalman filter can be derived. Let's recapitulate these assumptions to see how they are related to each other. So far we assumed that:

- the state is complete.
- the belief is Gaussian distributed.
- the system is linear with added Gaussian noise.

The complete state assumption made it possible to formulate recursive update equations for the agent's belief which resulted in the very general Bayes filter (see definition 2.2). To carry out the update equations of the Bayes filter in a closed fashion we made the assumption that our belief can be represented by a multivariate normal distribution, i.e., $\mathrm{bel}(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. To ensure that an updated belief $\mathrm{bel}(\mathbf{x}_t)$ from a previous belief $\mathrm{bel}(\mathbf{x}_{t-1})$, an action $\mathbf{u}_t$ and a measurement $\mathbf{z}_t$ is still Gaussian distributed we further posed some requirements on the involved pdfs for the state transition and the measurements, namely, the linear Gaussian system requirements.

Putting all this together (i.e., plugging the Gaussian distributions in the update equations of the Bayes filter) results in the following update equations for the parameters of the belief:

$$\overline{\boldsymbol{\mu}}_t = \mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t \mathbf{u}_t \tag{2.17}$$

$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{A}_t \boldsymbol{\Sigma}_{t-1} \mathbf{A}_t^\mathsf{T} + \mathbf{R}_t \tag{2.18}$$

$$\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t \mathbf{C}_t^\mathsf{T} (\mathbf{C}_t \overline{\boldsymbol{\Sigma}}_t \mathbf{C}_t^\mathsf{T} + \mathbf{Q}_t)^{-1} \tag{2.19}$$

$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t (\mathbf{z}_t - \mathbf{C}_t \overline{\boldsymbol{\mu}}_t) \tag{2.20}$$

$$\boldsymbol{\Sigma}_t = (I - \mathbf{K}_t \mathbf{C}_t) \overline{\boldsymbol{\Sigma}}_t \tag{2.21}$$

The parameters of the intermediate belief $\overline{\mathrm{bel}}(\mathbf{x}_t)$ are denoted by $\overline{\boldsymbol{\mu}}_t$ and $\overline{\boldsymbol{\Sigma}}_t$. The matrices $\mathbf{A}_t$, $\mathbf{B}_t$, $\mathbf{C}_t$, $\mathbf{R}_t$, and $\mathbf{Q}_t$ originate from the linear Gaussian system definition. The matrix $\mathbf{K}_t$ is called the *Kalman gain* and can be seen as the correcting part of the Kalman filter taking into account the measurements made. I is just an identity matrix.

The derivation of these update equations is not trivial. However, a lot of different proofs exist throughout the literature. An excellent step by step derivation can be found in the book by Thrun et al. [40].

Practical Considerations

The Kalman filter in its simplest form stated above became quite popular due to its simplicity and efficiency. Its complexity is in the worst case determined by the matrix inversion in the computation of the Kalman gain which can be performed in approximately $O(m^{2.4})$ (m being the dimension of the measurement vector). However, in practice the dimensionality of the measurement space is often much smaller than the state space. Thus, the matrix multiplication in the calculation of the updated covariance matrix $\boldsymbol{\Sigma}_t$ dominates the calculation. The multiplication can be carried out with a complexity of $O(n^2)$ (n being the dimension of the state space).

For systems that have the Gaussian linear properties the Kalman filter produces exact results. However, a lot of problems in probabilistic reasoning over time do not underly these linearities. The following section deals with the application of the Kalman filter in such environments.

2.1.3. Extended Kalman Filter

A major drawback of the use of the Kalman filter is the made assumption about the system to be linear Gaussian. This is not the case in most real world problems. The *extended Kalman filter*, or short *EKF*, tries to overcome these difficulties in the most straightforward way: The nonlinear system under consideration is linearized by approximation at each time step and fed to the original Kalman filter.

Linearisation

The original Kalman filter requires the system to be linear Gaussian. In particular, the following equations need to hold:

$$\mathbf{x}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \boldsymbol{\epsilon}_t \tag{2.22}$$

$$\mathbf{z}_t = \mathbf{C}_t \mathbf{x}_t + \boldsymbol{\delta}_t \tag{2.23}$$

Usually, however, we encounter nonlinear systems which follow the more general state transition and measurement equations:

$$\mathbf{x}_t = g(\mathbf{x}_{t-1}; \mathbf{u}_t) + \boldsymbol{\epsilon}_t \tag{2.24}$$

$$\mathbf{z}_t = h(\mathbf{x}_t) + \boldsymbol{\delta}_t \tag{2.25}$$

In this case, $g(.)$ and $h(.)$ are arbitrary functions that are differentiable with respect to $\mathbf{x}_{t-1}$ or $\mathbf{x}_t$, respectively. Even though these functions can be arbitrary, we still make the assumption of Gaussian distributed additive noise. Note that we consider $\mathbf{u}_t$ to be a fixed parameter of $g(.)$ and not an argument. Thus we have two functions on the state space.

The method for linearisation used in the EKF is the *first order Taylor expansion*. Instead of using $g(.)$ or $h(.)$ in the update equations, we use functions which describe a hyperplane that is tangent to these function at a certain point. The choice of this point is crucial for the quality of the approximation. Therefore, in each attempt to update the belief, the point of the currently most likely state is taken, which is $\boldsymbol{\mu}_{t-1}$ for $g(.)$ and $\overline{\boldsymbol{\mu}}_t$ for $h(.)$. The resulting linear approximations of $g(.)$ and $h(.)$ at this points read as follows:

$$g(\mathbf{x}_{t-1}; \mathbf{u}_t) \approx g(\boldsymbol{\mu}_{t-1}; \mathbf{u}_t) + \underbrace{\left.\frac{\partial g(\mathbf{x}; \mathbf{u}_t)}{\partial \mathbf{x}^\mathsf{T}}\right|_{\boldsymbol{\mu}_{t-1}}}_{\mathbf{G}_t} (\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \tag{2.26}$$

$$h(\mathbf{x}_t) \approx h(\overline{\boldsymbol{\mu}}_t) + \underbrace{\left.\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}^\mathsf{T}}\right|_{\overline{\boldsymbol{\mu}}_t}}_{\mathbf{H}_t} (\mathbf{x}_t - \overline{\boldsymbol{\mu}}_t) \tag{2.27}$$

In these equations $\mathbf{G}_t$ and $\mathbf{H}_t$ are the Jacobians[2] of the functions $g(.)$ and $h(.)$ at $\boldsymbol{\mu}_{t-1}$ and $\overline{\boldsymbol{\mu}}_t$, respectively.

Update Equations

Replacing the nonlinear functions $g(.)$ and $h(.)$ in the state transition and measurement equations 2.24 and 2.25 with their linear approximations, we obtain the following probability density functions for state transitions and measurements:

$$p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t) = \mathcal{N}(g(\boldsymbol{\mu}_{t-1}; \mathbf{u}_t) + \mathbf{G}_t(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}), \mathbf{R}_t) \tag{2.28}$$

$$p(\mathbf{z}_t | \mathbf{x}_t) = \mathcal{N}(h(\overline{\boldsymbol{\mu}}_t) + \mathbf{H}_t(\mathbf{x}_t - \overline{\boldsymbol{\mu}}_t), \mathbf{Q}_t) \tag{2.29}$$

These can be used in the same way as with the original Kalman filter to derive the following update equations for the EKF:

$$\overline{\boldsymbol{\mu}}_t = g(\boldsymbol{\mu}_{t-1}; \mathbf{u}_t) \tag{2.30}$$

$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^\mathsf{T} + \mathbf{R}_t \tag{2.31}$$

$$\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\mathsf{T} (\mathbf{H}_t \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\mathsf{T} + \mathbf{Q}_t)^{-1} \tag{2.32}$$

$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t(\mathbf{z}_t - h(\overline{\boldsymbol{\mu}}_t)) \tag{2.33}$$

$$\boldsymbol{\Sigma}_t = (I - \mathbf{K}_t \mathbf{H}_t) \overline{\boldsymbol{\Sigma}}_t \tag{2.34}$$

[2]For an introduction of the concept of Jacobians see appendix A, page 80.

Note that the matrices $\mathbf{A}_t$, $\mathbf{B}_t$, and $\mathbf{C}_t$ have been completely replaced by the Jacobians $\mathbf{G}_t$ and $\mathbf{H}_t$. The derivation of these equations is very similar to that of the original Kalman filter and can also be found in the book by Thrun et al. [40].

Practical Considerations

The EKF inherits its simplicity and efficiency from the original Kalman filter. Beside that, it allows for nonlinear systems. Both these points made the EKF the most popular choice for tracking problems in probabilistic robotics.

However, it is important to notice that the EKF just provides a very rough linear approximation of state transition and measurement functions. In general, the EKF does not estimate the true mean and covariance of an agent's belief. The degree of the violation made by the linearisation does not just depend on the "local linearity" of the function being approximated. The resulting estimation also suffers from a high uncertainty in the belief. Intuitively, if uncertainty is high, hypotheses far from the mean need to be considered. But the linear approximation of the state transition and measurement functions is just precise in a small area around the mean.

2.1.4. Iterated Extended Kalman Filter

The poor estimates of the extended Kalman filter in the case of a highly nonlinear measurement function $h(.)$ can be compensated by iterating the update step: The assumption is that each update provides a better estimate of the true state than the previous one. With such a better estimate, in particular with the mean of the state estimation, the measurement function gets linearised again and the Kalman gain is recomputed. This procedure is continued until the change to the state estimate is not significant anymore.

Update Equations

The update equations of the so-called *iterated extended Kalman filter* (IEKF [21]) are given as:

$$\overline{\boldsymbol{\mu}}_t = g(\boldsymbol{\mu}_{t-1}; \mathbf{u}_t) \tag{2.35}$$

$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^\mathsf{T} + \mathbf{R}_t \tag{2.36}$$

$$\boldsymbol{\mu}_t^0 = \overline{\boldsymbol{\mu}}_t \tag{2.37}$$

$$\mathbf{K}_t^i = \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\mathsf{T} (\mathbf{H}_t \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^\mathsf{T} + \mathbf{Q}_t)^{-1} \tag{2.38}$$

$$\boldsymbol{\mu}_t^i = \boldsymbol{\mu}_t^{i-1} + \mathbf{K}_t^i (\mathbf{z}_t - h(\boldsymbol{\mu}_t^{i-1})) \qquad i = 1, \ldots, k \tag{2.39}$$

$$\boldsymbol{\Sigma}_t = (I - \mathbf{K}_t^k \mathbf{H}_t) \overline{\boldsymbol{\Sigma}}_t \tag{2.40}$$

Practical Considerations

The IEKF turned out to give good approximations even in the case of nonlinear measurement functions. Surprisingly, in the case of very accurate sensors with few noise the extended Kalman filter shows a poor behavior if the measurement function is not linear [12]. The iteration of the update step compensates the overshooting that is an effect of the linearisation.

Input : $\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, g_t, \mathbf{R}_t$
Output: $\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t$

$\overline{\boldsymbol{\mu}}_t \leftarrow g(\boldsymbol{\mu}_{t-1}; \mathbf{u}_t)$;
$\mathbf{G}_t \leftarrow \frac{\partial g(\mathbf{x}; \mathbf{u}_t)}{\partial \mathbf{x}}\Big|_{\boldsymbol{\mu}_t}$;
$\overline{\boldsymbol{\Sigma}}_t \leftarrow \mathbf{G}_t \boldsymbol{\Sigma}_{t-1} \mathbf{G}_t^{\mathsf{T}} + \mathbf{R}_t$;
return $\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t$

Listing 1: function `Predict`

Input : $\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t, \mathbf{z}_t, h_t, \mathbf{Q}_t$
Output: $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

$\boldsymbol{\mu}_t^0 \leftarrow \overline{\boldsymbol{\mu}}_t$;
$i \leftarrow 0$;
repeat
  $i \leftarrow i + 1$;
  $\mathbf{H}_t^i \leftarrow \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\boldsymbol{\mu}_t^{i-1}}$; // linearisation
  $\mathbf{K}_t^i \leftarrow \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^{i\mathsf{T}} (\mathbf{H}_t^i \overline{\boldsymbol{\Sigma}}_t \mathbf{H}_t^{i\mathsf{T}} + \mathbf{Q}_t)^{-1}$; // Kalman gain
  $\boldsymbol{\mu}_t^i \leftarrow \boldsymbol{\mu}_t^{i-1} + \mathbf{K}_t^i (\mathbf{z}_t - h(\boldsymbol{\mu}_t^{i-1}))$; // state update
until $|\boldsymbol{\mu}_t^i - \boldsymbol{\mu}_t^{i-1}| \leq \epsilon$;
$\boldsymbol{\Sigma}_t \leftarrow (I - \mathbf{K}_t^k \mathbf{H}_t^i) \overline{\boldsymbol{\Sigma}}_t$;
return $\boldsymbol{\mu}_t^i, \boldsymbol{\Sigma}_t^i$

Listing 2: function `IterateUpdate`

This advantage comes at a cost: Each iteration of the IEKF requires an update of the Kalman gain, which involves a matrix inversion of the size of the measurement vector. As mentioned earlier, this inversion can at best be carried out with a complexity of approximately $O(m^{2.4})$ (with m being the size of the measurement vector).

Regardless of this performance bottleneck, the IEKF became a popular choice for VSLAM and the reference implementation we are using throughout this thesis is using it as well. In listing 1 and 2 we provide the predict and update step of the IEKF as pseudo code. We will refer to them later in section 3.2 when we show how the VSLAM process can be interfaced to the IEKF .

2.2. Multiple View Geometry

In this section we will give a brief introduction to multiple view geometry, basically introducing the concepts and notations that are used throughout this thesis. For a detailed view on the whole theory behind multiple view geometry we would like to refer the reader to the book by Hartley and Zisserman [20].

2.2.1. Homogeneous Image Coordinates

Consider an image I with image points $\Omega \subset \mathbb{R}^2$. Apparently, each image point $\mathbf{p}$ is determined by two coordinates $(u, v)^{\mathsf{T}}$. Now, lets think of the image as being a

projection plane of a simple pinhole camera in a 3D coordinate system. An image point represents a ray of sight in terms of the outer 3D coordinate system. Any 3D point on this ray would project to this image point. Thus, each of the points on this ray is an equally appropriate representation of the 2D image point $\mathbf{p}$. By assuming that each possible ray that made up the image intersects the origin of the outer coordinate system $(0, 0, 0)^\mathsf{T}$, we find that the ray representing an image point is given by all points $\lambda(x, y, z)^\mathsf{T}$. By further assuming that the image plane is located at $z = 1$and is parallel to the x-y-plane, we see that the image point $(u, v)^\mathsf{T}$ is located at the 3D point $(u, v, 1)$ and represented by the ray $\lambda(u, v, 1)^\mathsf{T}$. We call any of the points on that ray with $z \neq 0$ a *homogeneous representation* of the image point $\mathbf{p}$ and denote them by $\tilde{\mathbf{p}} = (x, y, z)^\mathsf{T}$. The according image point $\mathbf{p}$ can be restored as the projection of $\tilde{\mathbf{p}}$ to the image plane:

$$\mathbf{p} = \pi(\tilde{\mathbf{p}}) \stackrel{\text{def}}{=} \begin{pmatrix} x/z \\ y/z \end{pmatrix}; \quad z \neq 0 \tag{2.41}$$

It can be seen from this definition, that the projection is invariant to scaling of the homogeneous representation, i.e.:

$$\pi(\tilde{\mathbf{p}}) = \pi(\lambda\tilde{\mathbf{p}}); \quad \lambda \neq 0 \tag{2.42}$$

Not every possible ray represents a finite point on the image plane: All points $(x, y, 0)$ define rays $\lambda(x, y, 0)$ that are parallel to the image plane at $z = 1$. These rays represent infinite points on the image plane in a certain direction, which is given by x and y. Theses representations constitute the actual extension that projective geometry makes to the Euclidean geometry. However, for our purposes it is sufficient to care about finite representations only.

2.2.2. Projective Camera

In the previous section, we implicitly assumed a very simple pinhole camera model, namely a camera with a projection plane at $z = 1$ and a *principal point* at the origin of Ω.

The principal point of a camera is the image point of the intersection of the z-axis of the outer coordinate system. This point describes an offset or a planar translation of the image points with respect to the outer coordinate system. We will denote the principal point by $(\Delta u, \Delta v)$.

The location of the projection plane on the z-axis determines the angle of view of the camera. This corresponds to a scaling of the image content: The closer the image plane is to the origin, the wider is the angle of view. The distance to the origin on the z-axis is called the *focal length* and denoted by f.

To account for these camera parameters, the 3D points that have to be projected to the image plane need to be transformed. This transformation is carried out by a matrix $\mathbf{K}$, which is commonly known as the *intrinsic camera matrix*:

$$\mathbf{K} = \begin{bmatrix} f & 0 & \Delta u \\ 0 & f & \Delta v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.43}$$

The projection of a 3D point $\tilde{\mathbf{p}}$ under consideration of the intrinsic camera matrix is given as:

$$\mathbf{p} = \pi_\mathbf{K}(\tilde{\mathbf{p}}) = \pi(\mathbf{K}\tilde{\mathbf{p}}) = \begin{pmatrix} \Delta u + fx/z \\ \Delta v + fy/z \end{pmatrix}; \quad z \neq 0 \tag{2.44}$$

Due to the properties of homogeneous coordinates, this projection is invertible. In general, π can not be inverted because it is not bijective. However, homogeneous coordinates are invariant to scale. Thus it is sufficient to give *any* of the possible $\tilde{\mathbf{p}}$. Given an image point $\mathbf{p}$ we obtain the ray representation in the outer coordinate system as:

$$\tilde{\mathbf{p}} = \pi_{\mathbf{K}}^{-1}(\mathbf{p}) \stackrel{\text{def}}{=} \begin{pmatrix} (u - \Delta u)/f \\ (v - \Delta v)/f \\ 1 \end{pmatrix} \tag{2.45}$$

We call this special representation of the ray representing $\mathbf{p}$ the *normalized image coordinates* of $\mathbf{p}$.

2.2.3. Rigid Transformations

Besides the projection covered in the previous section, we have to deal with two more transformations: The translation and the rotation of 3D points. We will use these transformations to express different camera and object poses.

A translation will be denoted as a vector $\mathbf{t} = (\mathbf{t}_x, \mathbf{t}_y, \mathbf{t}_z)^\mathsf{T}$. The translated point $\mathbf{q}$ is just given as $\mathbf{p} + \mathbf{t}$.

A rotation will be denoted by a vector $\boldsymbol{\rho} = (\boldsymbol{\rho}_x, \boldsymbol{\rho}_y, \boldsymbol{\rho}_z)^\mathsf{T}$. The direction of $\boldsymbol{\rho}$ determines the rotation axis and the length of $\boldsymbol{\rho}$ the rotation angle.We prefer this representation of a rotation, which is referred to as *exponential representation*, because it is very compact: It just covers the three degrees of freedom of a rotation in 3D space. In contrast to that, a quaternion representation would need four scalar values and a rotation matrix even nine scalar values.

The application of a rotation $\boldsymbol{\rho}$ to a point $\mathbf{p}$ will be written as

$$\mathbf{q} = \boldsymbol{\rho}(\mathbf{p}) \tag{2.46}$$

and gives

$$\mathbf{q} = \cos\theta\mathbf{p} + (\mathbf{n}^\mathsf{T}\mathbf{p})(1 - \cos\theta)\mathbf{n} + \sin\theta(\mathbf{p} \times \mathbf{n}) \tag{2.47}$$

where

$$\mathbf{n} = \begin{cases} \frac{\boldsymbol{\rho}}{|\boldsymbol{\rho}|} & \text{if } |\boldsymbol{\rho}| > 0 \\ (1,0,0)^\mathsf{T} & \text{else} \end{cases} \tag{2.48}$$

$$\theta = |\boldsymbol{\rho}|. \tag{2.49}$$

Rotations can be concatenated. The resulting rotation $\boldsymbol{\theta} = \boldsymbol{\rho} \circ \boldsymbol{\phi}$ is the transformation obtained by first rotating about $\boldsymbol{\phi}$ and then about $\boldsymbol{\rho}$. Unfortunately, the computation of θ is not straight forward. In appendix C.2 we give a solution that involves the conversion to quaternions.

2.2.4. Frames and Poses

The combination of a translation $\mathbf{t}$ and a rotation $\boldsymbol{\rho}$ is an *Euclidean transformation* and is denoted by $\mathcal{F} = (\mathbf{t}, \boldsymbol{\rho})$. The application of this transformation $\mathcal{F}$ to a point $\mathbf{p}$ will be written as

$$\mathbf{q} = \mathcal{F}(\mathbf{p}) \tag{2.50}$$

and gives

$$\mathbf{q} = \boldsymbol{\rho}(\mathbf{p}) + \mathbf{t} \tag{2.51}$$

We will refer to this kind of a transformation as a *frame* or a *pose*: Any Euclidean transformation of a point can reasonably be thought of as the rotation and translation of

the base of a reference coordinate system, while the point itself stays fixed. In this case, the transformed point coordinates are just the projections of the point to the base of the transformed coordinate system. The transformed coordinate system is what we call a frame. In the same manner, one can think of the transformation as the pose of an entity in the 3D world: Any Euclidean transformation unambiguously determines a 3D pose.

A special frame is the world frame $\mathcal{W} = ((0,0,0)^\mathsf{T}, (0,0,0)^\mathsf{T})$, which does not perform any transformation at all. In the following, we will give any point with respect to some reference frame, which will be written as a superscript. Unless otherwise stated, points or frames without a frame superscript will implicitly be given with respect to the world frame.

In general, points that are given with respect to a frame $\mathcal{F}$ will therefore be denoted by $\mathbf{p}^\mathcal{F}$. Their world coordinates $\mathbf{p}$ can be recovered by applying the respective transformation:

$$\mathbf{p} = \mathcal{F}(\mathbf{p}^\mathcal{F}) \tag{2.52}$$

Euclidean transformations, and therefore frames or poses, can also be given with respect to another frame and will be denoted with a frame superscript as well. Consider a frame $\mathcal{G}^\mathcal{F} = (\mathbf{t}^\mathcal{F}, \boldsymbol{\rho}^\mathcal{F})$ that is given with respect to frame $\mathcal{F}$. The application of this frame to a point $\mathbf{p}^\mathcal{G}$ yields the representation of this point with respect to frame $\mathcal{F}$:

$$\mathbf{p}^\mathcal{F} = \mathcal{G}^\mathcal{F}(\mathbf{p}^\mathcal{G}) \tag{2.53}$$

Thus, a frame can be thought of as transforming points from one coordinate system to another. This transformation can be concatenated and yields:

$$\mathcal{G} = \mathcal{F} \circ \mathcal{G}^\mathcal{F} = (\boldsymbol{\rho}_\mathcal{F} \circ \boldsymbol{\rho}_{\mathcal{G}^\mathcal{F}},\ \boldsymbol{\rho}_\mathcal{F}(\mathbf{t}_{\mathcal{G}^\mathcal{F}}) + \mathbf{t}_\mathcal{F}) \tag{2.54}$$

2.2.5. Homography

Imagine two pinhole cameras at different positions observing the same 3D plane. Each point on the plane will be projected to both camera images. Thus, there is a functional dependency between the images of both cameras: It can be expressed by a function that gives for each coordinate of one camera image the coordinate of the corresponding pixel in the other camera image. The corresponding pixel in that sense is the pixel that was projected from the same 3D point on the plane.

A *homography* describes this functional dependency by means of homogeneous image coordinates. Using homogeneous image coordinates the homography can be expressed as a 3×3-matrix $\mathbf{H}$:

$$\tilde{\mathbf{p}} = \mathbf{H}\tilde{\mathbf{q}} \tag{2.55}$$

Here, $\tilde{\mathbf{q}}$ is the homogeneous representation of a point in one camera image and $\tilde{\mathbf{p}}$ the homogeneous representation of the corresponding point in the other camera image.

Due to the nature of homogeneous coordinates a homography is defined up to scale, thus having eight degrees of freedom. Usually, a homography is estimated given a set of known point correspondences. In our work, however, we will synthesise a homography to explain the appearance of planes from different viewpoints (see section 4.3.3).

3. Visual Slam Process

In this chapter we will give an introduction to simultaneous localisation and mapping (SLAM) and in particular Visual SLAM (VSLAM). We will discuss the problem of localisation and mapping in general in the first section before we focus on the state of the art of current VSLAM systems.

In section 3.2 we will see how VSLAM can be done using active search and the iterated extended Kalman filter (IEKF). Finally, as an example of such a system, we will introduce the reference implementation we used for the experiments of this thesis.

3.1. Goal of the SLAM Process

Generally speaking, SLAM is a process carried out by intelligent agents to build a map of an unknown environment without any prior. In particular, the agent is not aware of its own position in this environment and therefore has to keep track of it simultaneously.

Once a map of the environment has been established, the SLAM reduces to the problem of localisation. Depending on the concrete technique used, this may involve the consideration of map uncertainties.

There is no compulsory way to perform SLAM. In fact, every technique that is able to produce a map and localise the agent given some *sensor readings* and *actions* can be called a SLAM process. As a historic example, even the mapping of the coast of New Zealand done by Captain James Cook in 1769 can be considered as a SLAM process: Cook selected some distinctive landmarks on the coast that could easily be identified from the ship. Via a technique called triangulation he was able to measure the relative distances of more and more landmarks as he orbited the island. By measuring the speed of his ship he was also able to incorporate his ego-motion into the whole process. Thus, without prior knowledge about the ship's position Cook was able to provide a map that was precise enough that it was used for 200 years.

However, when speaking about SLAM today one usually implies a *real time constraint*: It states that the sensor readings have to be processed and integrated in the localisation and mapping within a certain period of time. Usually, this time is required to be the inverse frequency of the sensor readings. The reason for that is that SLAM systems are designed to give map estimates as the agent explores an unknown environment. The maps obtained this way may help the agent to draw decisions, e.g., finding a path. Localisation and mapping tasks that are not subject to a real time constraint fall within the category of *structure from motion* problems.

Therefore, one goal of the SLAM process is to handle the performance issues that naturally arise in the pre-processing of sensor readings, the integration of sensor readings and the maintenance of maps of arbitrary sizes.

Another goal that is common to all SLAM systems is the compensation for local errors in the map estimate. Sensor readings are never precise enough to give perfect estimates of the environment and the location. Now if an agent is moving in an unknown environment without revisiting an already seen place for a long time, the sensor

errors accumulate. The result is a map that might be locally acceptable but is globally deformed. The deformation can be corrected as soon as the agent visits an area it knows already - a technique called *loop closing*. Intuitively speaking, the difference between the expected position of the agent and the known position of the revisited area tells the agent what is wrong with the current map estimate. However, how to recognise that the agent revisits a known part of the map is still a problem.

For a map to be of any use for an intelligent agent, it has to provide sensible information. What exactly sensible information is depends highly on the scenario: In the case of a housekeeping robot it might be useful to incorporate walls into the map, whereas in the case of satellite navigation a map of points might be enough.

3.2. Visual Slam with the IEKF

Visual SLAM (VSLAM) aims to perform SLAM with a moving camera as the only sensor. The Bayes filter framework would allow for easy integration of other measurements like odometry data or the readings of positioning systems. Given that such measurements exist their integration would clearly improve the performance of the system. However, VSLAM focuses on the reconstruction and localisation of just a camera, which is usually hand-held.

The *extended Kalman filter* (EKF) as an implementation of the more general Bayes filter was made popular for the use in VSLAM systems by Andrew J. Davison [11]. However, to compensate for the linearisation errors that occur especially in the case of high uncertainties, we use the *iterated extended Kalman filter* (see section 2.1.4 for a discussion on the IEKF). In the following we will describe how a VSLAM system interfaces to this framework by first introducing some general concepts that we belief are common to all VSLAM systems.

In the subsequent parts of this section we will narrow our focus on VSLAM systems using point features and concretise the concepts introduced in the first part.

3.2.1. General Concepts

The Bayes filter framework and its implementations require the system to carry out measurements. However, the camera images can not be taken directly as measurements in that sense: This would require a measurement function that predicts every single image - and therefore describe every aspect of the world that leaves a trace in the camera images. Instead, in VSLAM we try to measure the poses of selected *features* of the real world.

Definition 3.1: (Feature) A feature is a real world entity like a point, line, plane, or object that we would like to measure. A feature has a pose in the world.

Definition 3.2: (Feature Pose) A feature pose is a vector that unambiguously describes the pose of a feature in the world with respect to a reference coordinate frame.

Features alone cannot be measured directly in the camera images, although it is evident that each visible feature is projected to the camera images in a way that depends

on its pose and on the camera pose. To be able to measure the image pose of a feature we need a *descriptor* that describes the visual appearance of the feature and its neighborhood.

Definition 3.3: (Feature Descriptor) A feature descriptor is a vector used to describe the visual appearance of a feature and its close neighborhood. Feature descriptors are used to measure feature poses in camera images.

Feature descriptors are required to be robust and distinctive. The same feature should produce similar feature descriptors even under different measurement conditions and different features should result in different feature descriptors. The need to have distinctive feature descriptors disqualifies some features to get represented in a VSLAM system, namely features that would have similar feature descriptors. Therefore, we aim to keep track of so-called *interest points*.

Definition 3.4: (Interest Point) An interest point is a feature that has a distinctive feature descriptor.

Internally, a feature is represented as by a feature descriptor and a feature pose. Some values of the feature representation are part of the state $\mathbf{x}$ and thus not given explicitly but by means of a probability density function $p(\mathbf{x})$. Usually, this is at least the pose of the feature.

3.2.2. Interface to the IEKF

To interface a VSLAM system to the IEKF, two functions have to be provided: A state transition function $g(.)$ and a measurement function $h(.)$. Given that, it remains to perform measurements and revise the state for consistency.

The outer loop of a VSLAM algorithm interfacing the IEKF is shown in listing 3. There we assumed a stereo camera setup as this is the more general scenario. The loop encloses the following four functions (besides the image grabbing, of course):

- `ReviseState`: Due to linearisation errors and measurement noise that is not explained by the measurement model, some features might have bad estimates of their pose. In this step, a heuristic tries to detect such features and remove them from the state. The initialisation of new features is also carried out in this step. The initialisation of a feature is the first measurement of a yet unknown feature. The feature descriptor of this measurement is stored for later reference and the feature pose is inserted into the state under consideration of the measurement and camera pose uncertainty. According to the new layout of the state $\mathbf{x}_t$, the state transition function $g_t(.)$ is defined, as well as the transition covariance $\mathbf{R}_t$. The set of feature descriptors $\boldsymbol{D}$ is changed analogously.

- `Predict`: This is the prediction step of the IEKF (compare listing 1 on page 22). This requires the state transition function $g_t(.)$, the covariance matrix $\mathbf{R}_t$ modelling the uncertainties of the state transition, and an action $\mathbf{u}_t$. Usually, VSLAM systems are passive, i.e., they do not perform actions and therefore $\mathbf{u}_t = \varnothing$.

Input : Initial state estimate $\boldsymbol{\mu}_0$, $\boldsymbol{\Sigma}_0$
Output: Final state estimate $\boldsymbol{\mu}_t$, $\boldsymbol{\Sigma}_t$

$\boldsymbol{D} \leftarrow \varnothing$; `// feature descriptors of tracked features`
$t \leftarrow 0$;
while *running* **do**
  $t \leftarrow t + 1$;
  $(I_l, I_r) \leftarrow$ `GrabCameraImages()`;
  $(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, g_t, \mathbf{R}_t, \boldsymbol{D}) \leftarrow$ `ReviseState(`$\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, I_l, I_r, \boldsymbol{D}$`)`;
  $(\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t) \leftarrow$ `Predict(`$\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \varnothing, g_t, \mathbf{R}_t$`)`;
  $(\mathbf{z}_t, h_t, \mathbf{Q}_t) \leftarrow$ `PerformMeasurements(`$\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t, I_l, I_r, D$`)`;
  $(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \leftarrow$ `IterateUpdate(`$\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t, \mathbf{z}_t, h_t, \mathbf{Q}_t$`)`;
end
return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

Listing 3: VSLAM using the IEKF.

- `PerformMeasurements`: In this step the projected poses of the currently visible features are sought in the camera images. To find the projected poses the feature descriptors are used, as we will explain later. The results of this search make up the measurement vector $\mathbf{z}_t$. Since not all features can be measurable all the time the measurement function $h_t(.)$ has to be constructed accordingly to the measurements made. According to the structure of $h(.)$, the measurement noise covariance matrix $\mathbf{Q}_t$ is defined as well in this step.
- `IterateUpdate`: This is the update step of the IEKF (compare listing 2 on page 22). This requires the measurement function $h_t(.)$ of the previous step, a covariance matrix $\mathbf{Q}_t$ modelling the measurement noise, and the actual measurements $\mathbf{z}$.

The implementations of the functions `Predict` and `IterateUpdate` are given by the IEKF framework (see section 2.1.4). The implementations of the remaining functions `ReviseState` and `PerformMeasurements` are a matter of the concrete VSLAM system and the features it uses. In the following section we will briefly discuss these functions for a plain VSLAM system that uses point features only.

3.3. Point VSLAM

Most VSLAM systems build sparse maps containing *point feature* poses. Although there are currently efforts to represent other types of features as well in VSLAM systems, point features remain to be a popular choice. In contrast to other features and their descriptors, interest points for point features can easily be extracted, predicted and matched in camera images. In fact, the measurement function of a point feature is the straight forward projection of the point's world coordinates to the camera images.

Definition 3.1: (Point Feature) A *point feature* is a point in the world that lies on a surface. With respect to a world coordinate system a point feature has a position $\mathbf{p}$ and a normal $\mathbf{n}$, which is the surface's normal at position $\mathbf{p}$. It is assumed that point features do not move.

Common used descriptors for point features are the so-called *point feature templates*. These templates are basically small image patches of the neighborhood of point features as seen on initialisation of the features.

Definition 3.2: (Point Feature Template) A *point feature template* $\boldsymbol{t}$ is a point feature descriptor which describes the visual appearance of the surface area around a point feature position $\mathbf{p}$. It is a quadratic gray scale image with edge length $2n + 1$ centered on the projected point feature position:

$$\boldsymbol{t} : \Omega_{\boldsymbol{t}} \mapsto \mathbb{R} \tag{3.1}$$

$$\Omega_{\boldsymbol{t}} = \mathbb{Z}_{[-n,n]} \times \mathbb{Z}_{[-n,n]} \tag{3.2}$$

For this descriptor it is assumed that the surface of the point feature is locally planar around $\mathbf{p}$ and does not change its appearance.

Feature templates provide a fast but heavily constrained point feature descriptor. They assume that the neighborhood of a point feature is shaped like a plane and they are not invariant to rotation, scale and viewpoint changes. These invariances have to be achieved by consideration of the current estimates of the point feature and camera pose. As we will point out later in this section, this requires a good estimate of the current state to perform measurements at all. This is a major drawback of VSLAM systems using point features only. Once they lost track of the camera pose they are unable to recover.

The setup of an internal feature representation for a feature that is currently not tracked is called a *feature initialisation*. Based on an initial measurement of an interest point the pose of the respective point feature is getting estimated roughly. The feature descriptor that led to this first measurement is stored for subsequent measurements. Usually, only the position of the point feature is considered uncertain.

Consequently, the *state* $\mathbf{x}_t$ of a VSLAM system with point features consists of the current camera pose and the positions of each currently tracked point feature. The coordinates are given with respect to a fixed world coordinate system. Since the camera is allowed to move, the state also contains the velocity of the camera. This will be used to predict the position of the camera given the previous state.

Definition 3.3: (State) A state $\mathbf{x}_t$ at time t is a vector comprising the current camera pose $\mathcal{C}_t$, the velocity of the camera $\Delta\mathcal{C}_t$ and n point feature positions $\mathbf{p}^1, \ldots, \mathbf{p}^n$:

$$\mathbf{x}_t = (\mathcal{C}_t, \Delta\mathcal{C}_t, \mathbf{p}^1, \ldots, \mathbf{p}^n)^\mathsf{T}. \tag{3.3}$$

An estimate of the state at a certain point in time is called a *map*:

Definition 3.4: (Map) A map is a multivariate normal distribution over all possible states $\mathbf{x}_t$ at time t:

$$p(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t). \tag{3.4}$$

Input : $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, I_l, I_r, \boldsymbol{T})$
Output: $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, g_t, \mathbf{R}_t, \boldsymbol{T})$

1 $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{T}) \leftarrow$ `InitNewFeatures` $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, I_l, I_r, \boldsymbol{T})$;
2 $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{T}) \leftarrow$ `RemoveBadFeatures` $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{T})$;
3 $(g_t, \mathbf{R}_t) \leftarrow$ `GetStateTransition` $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$;
4 **return** $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, g_t, \mathbf{R}_t, \boldsymbol{T})$

Listing 4: function `ReviseState`

Input : $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, I_l, I_r, \boldsymbol{T})$
Output: $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{T})$

1 **if** *initialisation required* **then**
2 $P_r \leftarrow$ `ExtractInterestPoints` (I_r);
3 $\boldsymbol{T}_P \leftarrow$ `GetDescriptors` (P_r);
4 $P_l \leftarrow$ `FindDescriptors` $(\boldsymbol{T}_P, I_l)$;
5 $(P_{new}, \boldsymbol{T}_{new}) \leftarrow$ `SelectNewFeatures` $(P_l, P_r, \boldsymbol{T}_P)$;
6 $\boldsymbol{T} \leftarrow \boldsymbol{T} \cup \boldsymbol{T}_{new}$;
7 $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \leftarrow$ `InsertInState` $(P_{new}, \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$;
8 **end**
9 **return** $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{T})$

Listing 5: function `InitNewFeatures`

3.3.1. State Revision

As already mentioned in the previous section, the state revision step is responsible for the initialisation of new features and the removal of features with bad pose estimates. Both this tasks are guided by heuristics that decide what a bad pose estimate is, how often an initialisation of new features should be made, and where in the image new features should be initialised. There is no general consensus on these heuristics throughout the literature and we will not give concrete implementations of them here.

The outline of the state revision function is given in listing 4. In the following parts each of the three functions will be introduced.

Point Feature Initialisation

The function `InitNewFeatures` is given in listing 5. Based on an heuristic the VSLAM system periodically initialises new point features, as denoted by *initialisation required* in line 1. The heuristic should aim to ensure that new point features are getting initialised when there are only a few known features visible. This happens when the camera explores an area that was not seen before.

Whenever the heuristic decides to initialise new features, interest points P_r are getting extracted from the right camera image. In the case of point features, the FAST [32] algorithm provides robust interest points with respect to the feature template descriptor.

For each interest point in P_r a feature template is extracted and stored in $\boldsymbol{T}_P$. A feature template consists of a quadratic gray-scale image that is cropped from the right camera image. The center of the cropped area is the image position of the respective interest point.

```
Input  : (μ_t, Σ_t, T)
Output: (μ_t, Σ_t, T)
1 (P_bad, T_bad) ← SelectBadFeatures(μ_t, Σ_t);
2 T ← T \ T_bad;
3 (μ_t, Σ_t) ← RemoveFromState(P_bad, μ_t, Σ_t);
4 return (μ_t, Σ_t, T)
```

Listing 6: function `RemoveBadFeatures`

Based on the feature templates $\boldsymbol{T}_P$ stereo matches P_l are sought in the left camera image. Since we assume rectified camera images this search is restricted to one row per interest point. The method for finding the feature template in the left image will be described below when we talk about point feature measurements.

From all interest points that could be found in both camera images some are getting selected as P_{new} together with their feature templates $\boldsymbol{T}_new$ for the initialisation of new features. The selection criteria are also a matter of a heuristic. This heuristic should be subject to the distance to already tracked point features.

Based on the image positions P_{new} of the selected interest points, the initial pose estimates of the respective point features are set up. For that, the uncertainty of the current camera pose and the initial measurement has to be considered.

Point Feature Removal

The function `RemoveBadFeatures` is given in listing 6. A heuristic decides which features have a bad pose estimate and should be removed. Usually, this heuristic involves counting of successful and failed measurement attempts per feature.

Once a set of features $\boldsymbol{T}_{bad}$ has been selected for removal, their corresponding feature templates are discarded and the poses P_{bad} are removed from the state.

State Transition Function

Accordingly to the layout of the current state estimate, the state transition function has to be defined. In VSLAM with point features the state consists of the camera pose, its velocity, and the pose of each point feature that is currently tracked (see definition 3.3). Since we assume that point features do not move, the poses of the point features are passed through unchanged by the state transition function.

However, the camera does move. The current velocity $\Delta\mathcal{C}_t$ of the camera is stored in the state as well. It is assumed that the velocity of the camera does not change, i.e.:

$$\mathcal{C}_{t+1} = \mathcal{C}_t + \Delta\mathcal{C}_t \tag{3.5}$$

$$\Delta\mathcal{C}_{t+1} = \Delta\mathcal{C}_t \tag{3.6}$$

It follows that the state transition function $g_t(.)$ at time t with n point feature poses in the state is:

$$g_t(\mathbf{x}) = (\mathcal{C}_t + \Delta\mathcal{C}_t, \Delta\mathcal{C}_t, \mathbf{p}^1, \ldots, \mathbf{p}^n)^\mathsf{T} \tag{3.7}$$

Evidently, this function is not realistic. If the velocity of the camera would never change, the camera would only move in one direction or not move at all. Therefore

Input : $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, I_l, I_r, \boldsymbol{T})$
Output: $(\mathbf{z}_t, h_t, \mathbf{Q}_t)$

$\mathbf{z}_t = \varnothing$
for $i \leftarrow 1$ **to** $|\boldsymbol{T}|$ **do**
  $(\Omega_l, \Omega_r) \leftarrow$ `PredictMeasurement` $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \mathbf{p}^i)$;
  if *not a candidate* **then**
    try next point feature;
  end
  $\bar{\boldsymbol{t}} \leftarrow$ `WarpTemplate` $(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \boldsymbol{t}_i)$;
  $\mathbf{p}_r^* \leftarrow$ `MatchTemplate` $(\bar{\boldsymbol{t}}, \Omega_r, I_r)$;
  if *score under threshold* **then**
    $d^* \leftarrow$ `GetDisparity` $(\bar{\boldsymbol{t}}, \mathbf{p}_r^*, \Omega_l, I_l)$;
    if *score under threshold* **then**
      $\mathbf{z}_t \leftarrow (\mathbf{z}_t^\mathsf{T}, \mathbf{p}^{*\mathsf{T}}, d^*)^\mathsf{T}$;
    end
  end
end
$(h_t, \mathbf{Q}_t) \leftarrow$ `GetMeasurementFunction` $(\mathbf{z}_t)$;
return $(\mathbf{z}_t, h_t, \mathbf{Q}_t)$

Listing 7: function `PerformMeasurements`

every acceleration of the camera - be it positive or negative - is modelled as noise:

$$\mathbf{R}_{\mathcal{C},\mathcal{C}} = \mathbf{1}_{[6\times 6]}\mathbf{c} \tag{3.8}$$

$$\mathbf{R}_{\Delta\mathcal{C},\Delta\mathcal{C}} = \mathbf{1}_{[6\times 6]}\Delta\mathbf{c} \tag{3.9}$$

Here, $\Delta\mathbf{c}$ and $\mathbf{c}$ contain the variances of the individual components of $\Delta\mathcal{C}$ and $\mathcal{C}$, respectively. Please note that we omitted the subscript t for better readability.

The remaining entries of $\mathbf{R}$ are just set to zero. Since the point features are assumed to be immobile there is no uncertainty added about their position as time elapses.

$$\mathbf{R}_t = \begin{bmatrix} \mathbf{R}_{\Delta\mathcal{C},\Delta\mathcal{C}} & & \\ & \mathbf{R}_{\mathcal{C},\mathcal{C}} & \\ & & \mathbf{0}_{[3n\times 3n]} \end{bmatrix} \tag{3.10}$$

The return values of the function `GetStateTransition` are precisely the function $g_t(.)$ and the matrix $\mathbf{R}_t$ as defined here.

3.3.2. Point Feature Measurement

The second yet uncovered function of listing 3 is `PerformMeasurements`. This function carries out the measurements to obtain $\mathbf{z}_t$ and defines the measurement function $h_t(.)$, which would predict the measurements given a state. It also returns the matrix $\mathbf{Q}_t$, which models the measurement noise. The outline of this function is given in listing 7. In the following, we will explain the functions used in it.

Measurement Prediction

To measure point feature poses in camera images the search area has to be restricted. Feature templates as point feature descriptors are computationally expensive. An exhaustive search of each feature template of each point feature in the state in the current camera images would be impractical. Instead, the uncertainty of the pose estimates of both the point features and the camera is projected to the camera images. This results in a pdf for the image pose of each point feature. From this pdf an elliptical area enclosing the most likely image positions $\Omega_l, \Omega_r \subset \Omega_I$ is cut out for the left and the right camera image.

Candidate Selection

A heuristic picks a set of measurement candidates from all currently visible point features. In our pseudo-code we represent this heuristic by the check in line 4. The heuristic should ensure that the point features are all visible, equally spread over the camera image, and that different point features are getting selected per IEKF cycle.

Template Warping

To search the feature template $\boldsymbol{t}_i$ of a candidate point feature in the uncertainty ellipse of the right camera image, the template has to be *warped*: Based on the current pose estimates, the viewpoint changes to the initial observation of the point feature are predicted. Since feature templates assume that point features lie on a locally planar surface, the viewpoint change can be described by a homography $\mathbf{H}$.

$$\bar{\boldsymbol{t}}(\mathbf{p}^{\mathcal{I}}) \stackrel{\text{def}}{=} \boldsymbol{t}(\mathbf{H}\mathbf{p}^{\mathcal{I}}) \tag{3.11}$$

Template Matching

To determine the best position of the warped feature template, the area in Ω_r is sought that looks the most similar to the warped feature template. For that, the *zero-mean sum of squared difference* (ZSSD, see [10]) between the feature template and the camera image is evaluated at each position $\mathbf{p}^{\mathcal{I}} \in \Omega_r$ inside the uncertainty ellipse. The sum of squared differences (SSD) of a feature template $\bar{\boldsymbol{t}}$ at position $\mathbf{p}^{\mathcal{I}}$ in an image I is given in the next equation. In the remainder of this function description all points are given with respect to the image coordinate system and therefore we will omit the superscript $\mathcal{I}$.

$$\mathrm{ssd}(\mathbf{p}, \bar{\boldsymbol{t}}) = \sum_{\mathbf{q} \in \Omega_{\bar{\boldsymbol{t}}}} \left(\bar{\boldsymbol{t}}(\mathbf{q}) - I(\mathbf{p}+\mathbf{q})\right)^2 \tag{3.12}$$

However, to prevent additive lighting changes of the observed patch to contribute to the SSD, the feature template and the image pixels that it will be compared to are getting modified such that the mean intensity of both is zero. The modification of the warped feature template has to be carried out once for each measurement attempt:

$$\tilde{\boldsymbol{t}}(\mathbf{q}) \stackrel{\text{def}}{=} \bar{\boldsymbol{t}}(\mathbf{q}) - \bar{\boldsymbol{t}}_{mean} \tag{3.13}$$

Here, $\bar{\boldsymbol{t}}_{mean}$ is the mean gray value of the warped feature template:

$$\bar{\boldsymbol{t}}_{mean} = \frac{1}{|\Omega_{\bar{\boldsymbol{t}}}|} \sum_{\mathbf{q} \in \Omega_{\bar{\boldsymbol{t}}}} \bar{\boldsymbol{t}}(\mathbf{q}) \tag{3.14}$$

The modification of the image pixel values has to be carried out for each image position $\mathbf{p}$ at which the normalised warped feature template is sought because of the different intensity means at different positions:

$$\tilde{I}_{\mathbf{p}} \stackrel{\text{def}}{=} I(\mathbf{p}) - I_{mean}(\mathbf{p}) \tag{3.15}$$

In this case the mean gray value depends on the search position $\mathbf{p}$ and the elongation of $\bar{\boldsymbol{t}}$:

$$I_{mean}(\mathbf{p}) = \frac{1}{|\Omega_{\bar{\boldsymbol{t}}}|} \sum_{\mathbf{q} \in \Omega_{\bar{\boldsymbol{t}}}} I(\mathbf{p} + \mathbf{q}) \tag{3.16}$$

Now, the ZSSD of a modified warped feature template $\tilde{\boldsymbol{t}}$ at position $\mathbf{p}$ in the modified image $\tilde{I}_{\mathbf{p}}$ is given as:

$$\mathrm{zssd}(\mathbf{p}, \tilde{\boldsymbol{t}}) = \sum_{\mathbf{q} \in \Omega_{\tilde{\boldsymbol{t}}}} \left(\tilde{\boldsymbol{t}}(\mathbf{q}) - \tilde{I}_{\mathbf{p}}(\mathbf{p} + \mathbf{q}) \right)^2 \tag{3.17}$$

The warped feature template is considered to be found at the image position $\mathbf{p}^*$ with the smallest ZSSD:

$$\mathbf{p}^* = \underset{\mathbf{p} \in \Omega_e}{\arg\min} \ \ \mathrm{zssd}(\mathbf{p}, \tilde{\boldsymbol{t}}) \tag{3.18}$$

Disparity Measurement

If the associated ZSSD of $\mathbf{p}^*$ falls below a threshold the feature template is also searched in the left camera image. This search is restricted to the segment of the epipolar line of the already found position that intersects the uncertainty ellipse of the measurement prediction in the left camera image. The best position of the warped feature template is given by the disparity d^*. If the score of the template matching is again under a threshold the point feature is considered to have been successfully measured.

Measurement Appending

The procedure explained above is carried out for each measurement candidate. The resulting image positions $\mathbf{p}_i^*$ and the corresponding disparities d_i^* constitute the measurement vector $\mathbf{z}_t$:

$$\mathbf{z}_t = (\mathbf{p}_1^*, d_1^*, \ldots, \mathbf{p}_n^*, d_n^*) \tag{3.19}$$

where n is the number of successful measurements. In out pseudo-code in listing 7 this appending is carried out in line 12.

Measurement Function

Accordingly to the features measured, the measurement function has to be defined. Assuming m successful measurements $\mathbf{z}_t^i$ of true feature positions $\mathbf{p}_i$, the measurement function is given as:

$$\mathbf{z}_t = h(\mathbf{x}_t) = \begin{pmatrix} h_1(\mathbf{x}_t) \\ \vdots \\ h_m(\mathbf{x}_t) \end{pmatrix} \tag{3.20}$$

$$h_i(\mathbf{x}_t) = \pi_S(\mathbf{p}_i^{\mathcal{C}}) \tag{3.21}$$

where π_S is the stereo projection of a point to the camera plane and $\mathbf{p}_i^{\mathcal{C}}$ is the pose of feature i in the camera coordinate frame, which is given by the camera pose $\mathcal{C}$.

The same per-feature separation is carried out for the design of the measurement uncertainty covariance $\mathbf{Q}_t$:

$$\mathbf{Q}_t = \begin{bmatrix} \mathbf{Q}_1 & & \\ & \ddots & \\ & & \mathbf{Q}_m \end{bmatrix} \tag{3.22}$$

$$\mathbf{Q}_i = \mathbf{1}_{[3\times 3]}\mathbf{d} \tag{3.23}$$

where $\mathbf{d}$ is a vector containing the variances of the individual components of a measurement $\mathbf{z}_i$. It is assumed that these variances are shared among all measurements. All other entries of $\mathbf{Q}_i$ are zero.

Finally, in line 16 in listing 7, the measurements $\mathbf{z}_t$, the measurement function $h_t(.)$, and the measurement uncertainty $\mathbf{Q}_t$ are getting returned.

3.4. Used Slam System

For the implementation of the object model and for comparative experiments we use the VSLAM system by Tobias Pietzsch.

This system is basically an implementation of the point VSLAM described in the previous chapter. The main difference is a performance enhancement technique called *inverse depth bundles* [30]. It tackles the problem of the rapid growth of the state as more and more point features get initialised. As we already pointed out in section 2.1, the time needed for an update of the IEKF grows quadratically with the state size.

The observation that led to the improvement was that usually several point features are initialised from one camera image. Instead of adding the 3D poses of the new features directly to the state, the pose of the camera at the time of observation is added to the state. This camera pose is called an *anchor*. The individual point feature positions can now be given as the depth on a ray, which goes from the anchor center through the anchor camera plane. The intersection with the anchor camera plane is the image location of the feature initialisation and the depth determines the 3D position of the point feature. This model is illustrated in figure 3.1.

By arguing that there is no uncertainty about the image position of the feature initialisation, the only uncertain value per feature is the depth on the ray. The inverse of this depth is added to the state. Thus, the number of components that are needed to express the position of one point feature dramatically reduces as more features share one anchor. $\mathbf{x}$

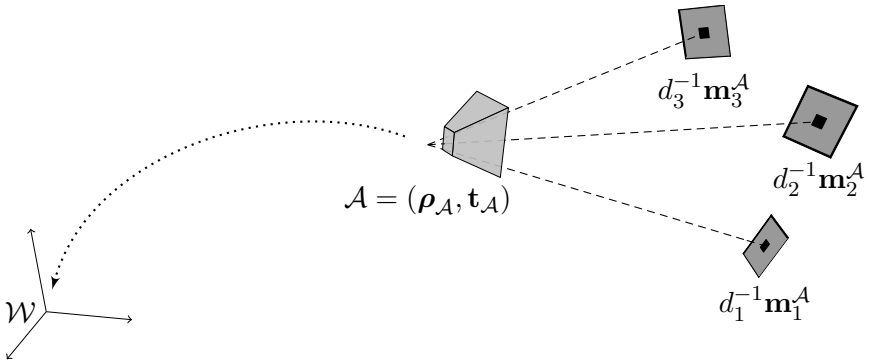


Figure 3.1.: Schematic view of the bundle model. Point features are initialised from one shared anchor pose $\mathcal{A}$, which is added to the state $\mathbf{x}$. The pose of the point features is represented with respect to this anchor pose. For that, the inverse depth d_i on a ray $\mathbf{m}_i^{\mathcal{A}}$ through the image plane is added to the state. Since the ray is represented with respect to the anchor frame $\mathcal{A}$, there is no uncertainty about it its direction and hence it is not added to the state.

4. Object Tracking

The object tracking is realised by means of the *object model*. This model extends the conventional point feature model of the reference implementation. The key concept of the new model is to group point features that belong to the same object. This grouping is currently done prior to the VSLAM run in a learning step, thus generating a database of object descriptors. During the VSLAM run we aim to keep track of the object poses instead of keeping track of several point features positions separately. Compared to the conventional point feature model this results in denser maps with smaller sizes and thus faster IEKF updates.

In the first two sections of this chapter we will give a formal description of the object model and explain the learning procedure. In section 4.3 we will see how objects are measured during a VSLAM run in detail and in section 4.4 we derive the Jacobian of the measurement function that is needed in the update step of the IEKF.

4.1. Model Description

The object model is an extension to the conventional point feature model. Our model introduces a new descriptor for objects which groups point feature descriptors together if their corresponding point features belong to the same object. The point feature descriptors making up an object descriptor are learned from exhaustive object descriptions in advance. During the VSLAM run we assume that there is no uncertainty about the relative positions between the point features that belong to the same object but only about the pose of the object as a whole. With this assumption made there is no need to track the positions of the point features individually because they are perfectly correlated if they belong to the same object. In contrast to the conventional point feature model this leads to a different handling of the feature positions because they are not represented as active parts of the state.

Definition 4.1: (Object) An *object* is a feature. It is a world entity that can clearly be bordered from its environment by its shape and texture. With respect to a world coordinate system an object has a pose $\mathcal{O}$.

An object descriptor is defined as:

Definition 4.2: (Object Descriptor) An *object descriptor* is a feature descriptor. It describes the visual appearance of an object by a collection of templates $\boldsymbol{T}$ and positions $\boldsymbol{P}$ of point features that lie on the surface of the object.

$$\boldsymbol{o} = (\boldsymbol{T}, \boldsymbol{P})$$

We assume that objects do not move. Usually, moving entities in the world pose a problem to VSLAM systems because it is hard to model movement. This is especially true if the VSLAM system is just tracking point features without knowledge of object boundaries. Having an object model would ease the modelling of movement. In our work, however, we stick to the fixed world assumption to get an objective view on the pure model performance. Non-moving objects allow for more accurate pose estimates and loop closing.

The proposed model allows to track several object poses simultaneously. Each pose is part of the state vector. It follows that these poses can be correlated with every other entity in the state, according to the measurements encountered. This leads to the following state and map definitions:

Definition 4.3: (State) A state $\mathbf{x}_t$ of the object model at time t is a vector comprising the current camera pose $\mathcal{C}_t$ and m object poses $\mathcal{O}^1, \ldots, \mathcal{O}^m$:

$$\mathbf{x}_t = \left(\mathcal{C}_t, \mathcal{O}^1, \ldots, \mathcal{O}^m\right)^\mathsf{T}.$$

Although we assume that objects do not move it is possible that we estimate different poses of the objects at different points in time. An estimate of the state at a certain point in time is called a *map*.

Definition 4.4: (Map) A map in the object model is a multivariate normal distribution over all possible states $\mathbf{x}_t$ at time t:

$$p(\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t).$$

Ideally, the sequence of the estimates converges to the true state. In reality, however, our estimations get beclouded by inaccurate measurements, linearisation errors of the underlying model and violated assumptions.

It can easily be seen from the definitions given above that an increase of the number of point feature descriptors per object descriptors does not increase the state size. This allows us to exhaustively learn the appearance of an object resulting in a dense representation. With such a representation we can perform accurate tracking of the object pose at no additional cost.

The learning of an object descriptor will be covered in the next section, followed by section 4.3 in which we describe the tracking of an object pose in detail.

4.2. Learning of Object Descriptors

Object descriptors represent point features that belong to the same object. We assume that the position of the point features with respect to the object position is known without uncertainty (see definition 4.2). This assumptions renders the online learning of object descriptors almost impossible for two reasons: First, by just measuring point features we do not know which point features belong to one object in the world. Second, each measurement is corrupted by noise and therefore we will not be able to provide the point feature positions without uncertainty, although eventually the uncertainty converges to zero.

Therefore, we decided to learn object descriptors offline. The outline of the procedure is as follows: First, we create an exact object description using the scene description language of the POV-Rayray tracer. This description is then used to create a *learn sequence*, which is an artificial sequence exploring the synthetic object from different viewpoints. This sequence is used with the VSLAM system instead of images from the camera to let the system track point features on the surface of the object. For each initialised feature we restore its exact position with respect to the object pose after the VSLAM run. Eventually, we clean up the point features we found in a post-processing step.

For the creation of the learn sequence and the tracing of the ground truth we use the Slamdunk evaluation framework, which we will introduce briefly in the next section. In the subsections 4.2.2 and 4.2.3 we will describe the creation of the learn sequence and the subsequent VSLAM run on it. The post-processing of the learnt features is handled in section 4.2.4.

4.2.1. Slamdunk

Slamdunk is an evaluation framework for VSLAM developed by Tobias Pietzsch and the author [17, 18]. The motivation for building the framework was the possibility to easily set up and perform VSLAM experiments on rendered image sequences. To allow other researchers to use Slamdunk as well we focused on the following requirements. The framework is freely available, that means, built on free software only. It should be easy to use, i.e., its adaptation to existing VSLAM implementations should be straightforward. Moreover, the framework is extensible to encourage contributions from the community.

For image generation, we use the POV-Ray command line ray tracer [1], which is free software and allows rendering of realistic images. More importantly, POV-Ray allows to calculate the intersection point of a ray with the whole scene. This feature is used for ground truth generation. Moreover, a range of high detail scene descriptions is freely available for this renderer.

A large part of our framework deals with automatising the processes of creating image sequences and evaluating experiment results. For the interface between the evaluation framework and the VSLAM system, we provide a lightweight API in C++ that allows access to the created sequence images and straightforward data logging of the VSLAM process.

For system integration and easy extendability, all relevant data throughout the framework is stored in an XML format. This includes camera and trajectory definitions, log data, ground truth, and intermediate evaluation results. The log data is processed using an extensible set of Python scripts.

Our evaluation framework consists of four components: a repository, a rendering system, an interface for VSLAM systems and an evaluation system (see Figure 4.1). The repository is used to store common items such as the descriptions of 3D scenes, camera trajectories and camera definitions which are needed by both the rendering system and the evaluation system. The rendering system is used to create image sequences from a specific setup of repository items. Using the interface (the lightweight C++-API), each of these sequences can be used instead of a regular camera (mono or stereo) to perform experiments on the VSLAM system under consideration. This interface can also be used to easily create structured log data during the experiment which are required for

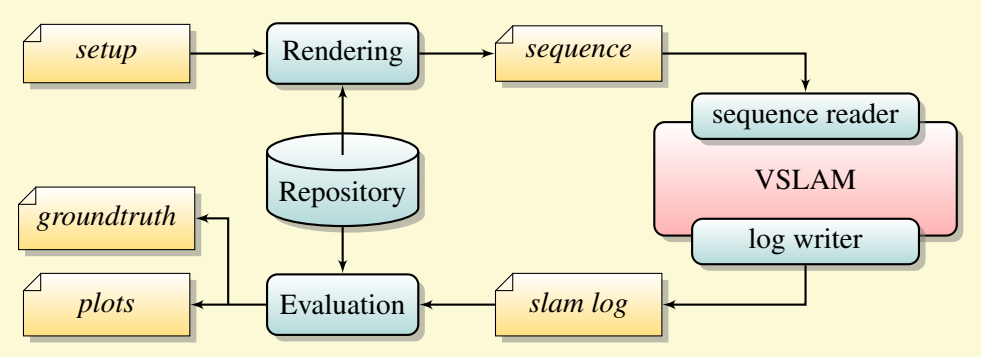


Figure 4.1.: Overview of the evaluation framework. An image sequence is rendered and provided as input to the VSLAM system under evaluation. A log of the systems operation is created and processed afterwards by the *evaluation system*. Together with the exactly known scene structure and trajectory, the log is used to generate ground truth for feature positions and measurements. The final result is a number of error plots comparing logged estimated values and ground truth. Interface classes *sequence reader* and *log writer* are provided to allow easy adaption to existing systems.

the evaluation system afterwards. For example, the log data includes estimated location of camera and/or map features, 2D image measurements, and feature initialisation events. The evaluation system consists of an extensible set of Python scripts that perform various evaluation tasks, e.g., the computation of ground truth and the creation of data plots for the experiments.

4.2.2. Creation of Learn Sequence

After an object description has been created using POV-Ray, a learn sequence is rendered exposing the synthetic object from different viewpoints. For that, we use artificial trajectories that try to explore the synthetic object from different viewpoints.

In figure 4.2 we see an example of a trajectory circulating around the object's x- and y-axis. Instead of using a trajectory that lets the camera move on a smooth path we have also been successful using a random trajectory: In each frame of the sequence, the camera is set to a random position with a randomly picked distance to the object in a certain interval. An example of a sequence originating from such a trajectory can be seen in figure 4.3.

Both approaches have their advantages: A smooth and predictable trajectory allows for actual point feature tracking in the subsequent step of the learning procedure. Thus it is possible to get statistics about individual features such as measurement failure ratios.

A random trajectory, on the other hand, eliminates the chance for biases during learning of the point features. These biases are introduced by smooth trajectories because they explore the object asymmetrically: when designing an artificial trajectory, one has to decide for certain viewing directions and distances to the object. Another advantage of a random trajectory is that the length of the learn sequence is shortened dramatically: Within only three seconds of a random sequence, the object would have been seen from 90 different viewpoints and distances.

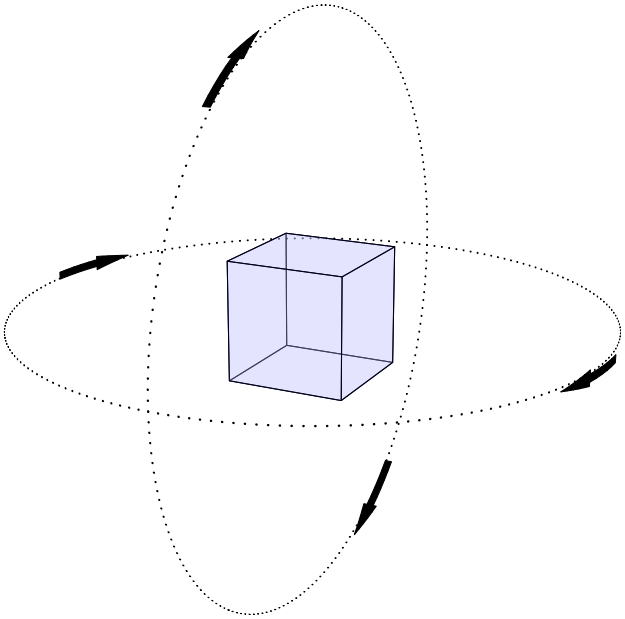


Figure 4.2.: Example of a smooth learn trajectory circulating around the object.

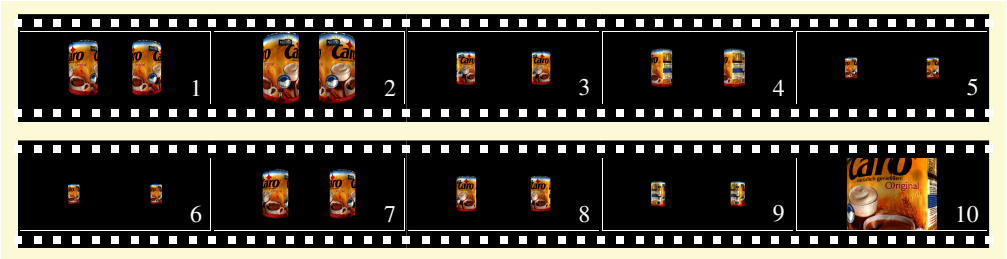


Figure 4.3.: The first ten frames of a learn sequence from a random trajectory. Because of the distinctive shape of the Caro tin, the camera viewpoints have been restricted to lie on the x-z-plane.

A compromise between both approaches would be a smooth trajectory that explores the object from every possible viewpoint and distance in a certain interval. However, the rendering of the sequence and the learning would be impractical.

We show in section 6.3.1 that the loss of statistics is not as crucial to the tracking as a biased object descriptor. Therefore, we favor the random learn trajectories.

4.2.3. Learning of Feature Templates

A slightly modified version of the VSLAM system described in section 3.4 is run on the learn sequence. The system is modified such that it initialises as much point features as possible. In the case of a smooth learning trajectory that means that new features are getting initialised as soon as only a few of the currently tracked features are visible. If the sequence was rendered using a random trajectory, the system fails to keep track of any features and therefore initialises features in every frame.

Using the Slamdunk evaluation framework, each point feature initialisation is logged with its image position and frame number. The point feature templates are stored as well.

These log data is processed by the Slamdunk evaluation framework, together with the object description, to generate the ground truth feature positions with respect to the object coordinate frame.

4.2.4. Post-Processing

Not every point feature is equally suitable for the object descriptor. Some of the point features might have been measured only a few times during learning, thus indicating weak point feature templates. Therefore we discard every point feature that has been measured less often then the median number of measurements per point feature. As pointed out earlier, this filtering is just possible if we use a smooth trajectory for the learn sequence.

Another problem are so called *ghost point features*, i.e., point features that got initialised because of the contrast between the object boundary and the background. The descriptors of these point features can not rely on the assumptions made about point features and are in particular not robust to viewpoint and background changes. We try to tackle these ghost point features by inspection of the point feature template: The background pixels are getting cropped from the template image to avoid their contribution to the ZSSD score.

4.3. The Measurement Process

The measurement process of the object model depends on the current state estimate. Due to the fact that we try to measure objects in camera images in real time we need to know which point features to expect in which areas of the camera images. An exhaustive search of every feature template of every object descriptor at every pixel position in both camera images would be unfeasible. Instead, we need to be able to narrow the areas in which certain point features can be visible.

To find out where a certain point feature is potentially visible, we need "good" state estimates. Whether or not we have a good state is often decided intuitively. Informally, we say to have a good state if both of the following criteria are true:

(a) The true projection of every point feature currently visible falls within the 3σ uncertainty ellipse of its estimated pose in the camera images.

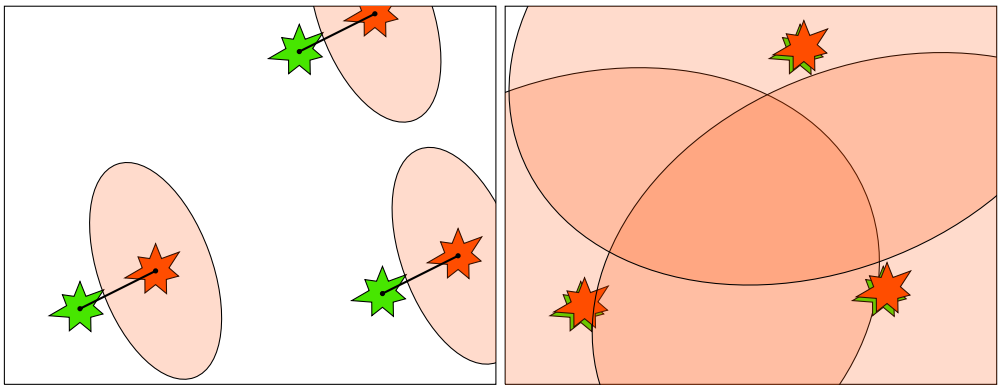
(b) The 3σ uncertainty ellipse of none of the point features visible exceeds an area that is to large too be searched in real time.

Examples of violations of these two criteria can be seen in figure 4.4.

If such a good state is not present, the system needs to recover[1]. However, we assume that the system is in a good state, which means in particular that some objects have already been initialised [2]. The measurement process can now be carried out as in listing 8, consisting of the following subfunctions:

[1] In section 5.6 we will see how object recognition can be used to recover a lost system.

[2] As we will point out in chapter 5.4, object initialisation is a non-trivial task that deserves its own chapter and is not handled here.



(a) bad estimate (b) inaccurate estimate

Figure 4.4.: Examples of bad state estimates. (a): The feature estimates (red) are too far away from the real feature positions (green) to get measured. (b): The uncertainty is too high to allow for real-time operation.

- `PredictMeasurement`: In this step we check, based on the current state estimate, which point features are visible and predict their image coordinates. We use these image coordinates together with the uncertainty of the respective object and camera pose to define search ellipses in both camera images.
- *Candidate Selection*: From all available point features we have to select some measurement candidates. In particular, we have to disregard point features that are currently not visible or expected to be seen from an inappropriate distance or angle.
- `WarpTemplate`: Also based on the current state estimate, we predict the appearance of the point feature template in both camera images. Since we assume that point features lie on locally planar surfaces we search for the homography relating the pixels of the point feature template and the camera images.
- `MatchTemplate` and `GetDisparity`: Within the search ellipses we try to find the warped point feature template images as an ZSSD minimum response. On success, i.e., if the ZSSD minimum falls below a threshold, we add the found image positions to the measurement vector.
- `RemoveOutliers`: The IEKF state update is not robust, i.e., we need to take care of measurement outlier removal first. Outliers occur in the case of repeated patterns, moving objects, reflections from non-Lambertian surfaces or weak feature templates. Usually, however, they do not fit to the other measurements made. The *joint compatibility branch and bound* (JCBB) method aims to reject these false measurements.
- `GetMeasurementFunction`: According to the measurements made, the measurement function and the covariance matrix modelling the measurement noise has to be defined. It will be used in the IEKF update function.

In the following subsections we will focus on each of these steps in detail. In subsection 4.3.1 we will introduce the measurement function that is used in `PredictMeasurement` and is also the return value of `GetMeasurementFunction`. In

```
Input  : (μ_t, Σ_t, I_l, I_r, O)
Output: (z_t, h_t, Q_t)
z_t = ∅
for j ← 1 to |O| do
    (T, P) ← o_j
    for i ← 1 to |T| do
        (p^I, Ω_l, Ω_r) ← PredictMeasurement(μ_t, Σ_t, p_i^O);
        if not a candidate then
            try next point feature;
        end
        t̄ ← WarpTemplate(μ_t, Σ_t, t_i);
        p^{I*} ← MatchTemplate(t̄, Ω_r, I_r);
        if score under threshold then
            d* ← GetDisparity(t̄, p*, Ω_l, I_l);
            if score still under threshold then
                z_t ← (z_t^T, p^{I*T}, d*)^T;
            end
        end
    end
end
z_t ← RemoveOutliers(z_t, μ_t, Σ_t);
(h_t, Q_t) ← GetMeasurementFunction(z_t);
return (z_t, h_t, Q_t)
```

Listing 8: function `PerformMeasurements`

the subsections 4.3.3 and 4.3.4 we will derive the mentioned homography for template warping and show how we use it to perform the actual measurements. Our outlier removal with the JCBB method is discussed in subsection 4.3.5.

4.3.1. Measurement Prediction

We need to have a measurement function (i.e., a function that predicts the measurements given a state) for two reasons: First, as mentioned above, we need to know where to search for which point feature in the camera images. Second, since we are using a variant of the extended Kalaman filter, we need the measurement function to compute the so called Kalman gain (see section 2.1.3).

In general, the measurement function is a function of the state $\mathbf{x}$ that returns the predicted measurements $\mathbf{z}$ as a vector:

$$\mathbf{z} = h(\mathbf{x}) \tag{4.1}$$

The measurable values in this model are the image positions of the point features that belong to an object. Each point feature template $\boldsymbol{t}_i$ of each object descriptor $\boldsymbol{o}_j$ can be measured if its corresponding point feature is visible. Since this holds for every object

descriptor in the same way, we can decompose $h(\mathbf{x})$ for each object that is currently tracked:

$$h(\mathbf{x}) = \begin{pmatrix} h_1(\mathbf{x}) \\ \vdots \\ h_k(\mathbf{x}) \end{pmatrix} \tag{4.2}$$

Here, k is the number of currently tracked objects and h_j is the measurement function for object descriptor $\boldsymbol{o}_j$. This function can be further decomposed into the measurement functions for the individual point feature templates:

$$h_j(\mathbf{x}) = \begin{pmatrix} h_{j,1}(\mathbf{x}) \\ \vdots \\ h_{j,n(j)}(\mathbf{x}) \end{pmatrix} \tag{4.3}$$

Here, $n(j)$ is the number of features that belong to object descriptor $\boldsymbol{o}_j$ and $h_{j,i}$ is the measurement function for point feature template $\boldsymbol{t}_i$ of object descriptor $\boldsymbol{o}_j$. This function provides a vector $(u, v, d)^\mathsf{T}$ consisting of the image coordinates of the corresponding point feature in the right camera image and the disparity to the left camera image. To focus on this function in more detail we will assume that we deal with only one object descriptor for a moment, i.e., we will omit the subscript j in the following.

Given a state $\mathbf{x}$ which provides the pose $\mathcal{O}$ of an object with descriptor $\boldsymbol{o} = (\boldsymbol{T}, \boldsymbol{P})$ and the pose of the current camera $\mathcal{C}$, the predicted image position of each point feature position $\mathbf{p}_i^{\mathcal{O}}$ in $\boldsymbol{P}$ is the stereo projection onto the camera plane:

$$h_i(\mathbf{x}) = \mathbf{p}_i^{\mathcal{I}} = \pi_S(\mathbf{p}_i^{\mathcal{C}}) \tag{4.4}$$

For the projection, we need $\mathbf{p}_i$ to be represented in the camera coordinate system. Having $\mathbf{p}_i^{\mathcal{O}}$, $\mathcal{O}$ and $\mathcal{C}$ this can be obtained by the following transformation (where we omitted the subscript i to enhance readability):

$$\begin{aligned}
\mathbf{p}^{\mathcal{C}} &= \mathcal{O}^{\mathcal{C}}(\mathbf{p}^{\mathcal{O}}) && (4.5)\\
&= \left(\mathcal{C}^{-1} \cdot \mathcal{O}\right)(\mathbf{p}^{\mathcal{O}}) && (4.6)\\
&= \left((-\boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{t}_{\mathcal{C}}), \boldsymbol{\rho}_{\mathcal{C}}^{-1}) \cdot (\mathbf{t}_{\mathcal{O}}, \boldsymbol{\rho}_{\mathcal{O}})\right)(\mathbf{p}^{\mathcal{O}}) && (4.7)\\
&= \left(\boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{t}_{\mathcal{O}} - \mathbf{t}_{\mathcal{C}}), \boldsymbol{\rho}_{\mathcal{C}}^{-1} \cdot \boldsymbol{\rho}_{\mathcal{O}}\right)(\mathbf{p}^{\mathcal{O}}) && (4.8)\\
&= \boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{t}_{\mathcal{O}} - \mathbf{t}_{\mathcal{C}}) + \left(\boldsymbol{\rho}_{\mathcal{C}}^{-1} \cdot \boldsymbol{\rho}_{\mathcal{O}}\right)(\mathbf{p}^{\mathcal{O}}) && (4.9)\\
&= \boldsymbol{\rho}_{\mathcal{C}}^{-1}\left(\underbrace{(\mathbf{t}_{\mathcal{O}} - \mathbf{t}_{\mathcal{C}}) + \boldsymbol{\rho}_{\mathcal{O}}(\mathbf{p}^{\mathcal{O}})}_{\mathbf{k}}\right) && (4.10)
\end{aligned}$$

Here, $\mathbf{k}$ is the vector from the current camera position to the feature position in world coordinates. We will refer to this vector later when deriving the Jacobian of $h(.)$ by $\mathbf{x}$.

Now, this measurement function is not just used in the IEKF update function but also to predict the most likely area of a measurement given a state estimate. For that, all uncertainties that affect a measurement are propagated through the measurement function resulting in a pdf for each measurement. For this propagation the measurement

function gets linearised via a first order Taylor expansion as in the case of the IEKF update. The therefore needed Jacobian $\frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$ will be derived in section 4.4. The resulting measurement distribution is a Gaussian, i.e.:

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{h}_i, \mathbf{S}_i) \tag{4.11}$$

The mean of the measurement distribution $\mathbf{h}_i$ is just the value of the measurement function at the mean of the current state estimate:

$$\mathbf{h}_i = h_i(\boldsymbol{\mu}_t) \tag{4.12}$$

The measurement covariance matrix $\mathbf{S}$ modelling the uncertainty of the measurement is affected by the uncertainties of the current state estimate $\boldsymbol{\Sigma}_t$ and the expected measurement noise covariance $\mathbf{Q}$. According to the linear version of $h(.)$ these uncertainties constitute $\mathbf{S}$ in the following way:

$$\mathbf{S}_i = \mathbf{H}_i \boldsymbol{\Sigma}_t \mathbf{H}_i^\mathsf{T} + \mathbf{Q}, \tag{4.13}$$

where $\mathbf{H}_i$ is the Jacobian of $h_i(.)$ at $\boldsymbol{\mu}_t$.

Given this measurement distribution, the most likely measurements $\boldsymbol{Z}_i$ are extracted for each point feature, i.e., the measurements having a Mahalanobis distance smaller than three:

$$\boldsymbol{Z}_i = \left\{ \mathbf{z} \in \Omega_S \;\middle|\; (\mathbf{z} - \mathbf{h}_i)\mathbf{S}_i^{-1}(\mathbf{z} - \mathbf{h}_i)^\mathsf{T} < 9 \right\} \tag{4.14}$$

Here, Ω_S is the domain of all possible measurements (u, v, d). The search of a point feature template is restricted to the set of the most likely measurements by determination of the pixels in the left and right camera image that belong to this set:

$$\Omega_r = \{(u, v) \mid (u, v, d) \in \boldsymbol{Z}_i\} \tag{4.15}$$

$$\Omega_l = \{(u + d, v) \mid (u, v, d) \in \boldsymbol{Z}_i\} \tag{4.16}$$

4.3.2. Candidate Selection

An important step in the measurement process is to select appropriate point features. Based on the estimates obtained by the measurement function we discard point features that appear to be not visible, to far away, to close to the camera or be seen from an inappropriate angle.

From the remaining point features we randomly select a fixed number of candidates[3]. For that we implemented two methods: Purely random selection and a random selection that favours features that are far away from already measured features.

We found that random selection of candidates (with or without prior) ensures that we most likely measure different point features between two frames, provided a large number of point features per object. In contrast to measuring the same point features over and over again this helps to reduce systematic measurement errors (which occur, for example, in the case of repeated structures in the images).

Surprisingly, the purely random method introduced a bias when used with an object descriptor obtained from a random learn trajectory. The reason for that are strong point features, i.e., point features that lie on a distinctive corner. These point features got initialised during learning more often then other features, hence they are overrepresented

[3]The exact number depends on the speed of the VSLAM system. We are using about five point features here.

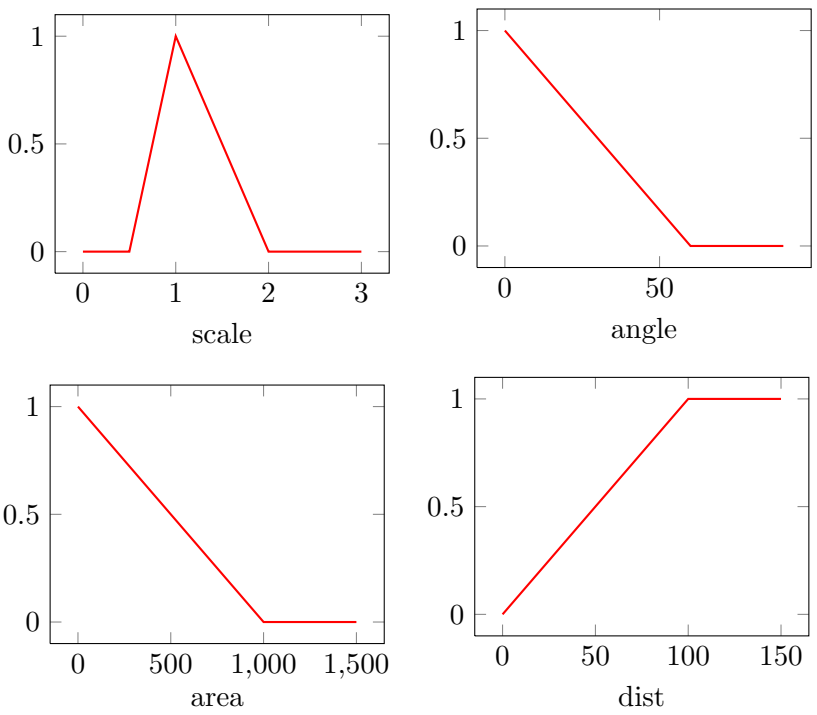


Figure 4.5.: The components of the score function for the guided random candidate selection strategy.

in the object descriptor. A random selection strategy without a prior would chose these features more often then others because there are more candidates for it. This results in a lot of identical measurements which pose a problem to the outlier removal that we will discuss later: Against the background of a few perfectly correlated measurements all other measurements appear to be outliers.

However, we have been able to fix that issue with a random selection strategy incorporating a prior: The candidates are still getting selected randomly, but with different probabilities. The probability for a point feature to get selected depends on a score function. This score function considers the required scaling, the viewing angle, the size of the search region, and the distance to already measured point features. We will refer to this selection strategy as the *guided random measurement candidate selection strategy*.

The score function for a template $\boldsymbol{t}$ is given as:

$$\text{score}(\boldsymbol{t}) = \text{scale}(s_{\boldsymbol{t}}) \cdot \text{angle}(\alpha_{\boldsymbol{t}}) \cdot \text{area}(A_{\boldsymbol{t}}) \cdot \text{dist}(d_{\boldsymbol{t}}) \tag{4.17}$$

Here, $s_{\boldsymbol{t}}$, $\alpha_{\boldsymbol{t}}$, and $A_{\boldsymbol{t}}$ are the predicted scale change, viewing angle, and search area for each template $\boldsymbol{t}$, respectively. The distance $d_{\boldsymbol{t}}$ is the distance in the camera image between the predicted position of the current template and the closest successful measurement done so far. The functions scale, angle, area, and dist are piecewise linear functions, as can be seen in figure 4.5.

The probability of selecting a template $\boldsymbol{t}$ is now given as:

$$p(\boldsymbol{t}) = \frac{\text{score}(\boldsymbol{t})}{\sum_{\boldsymbol{t}' \in \boldsymbol{T}} \text{score}(\boldsymbol{t}')} \tag{4.18}$$

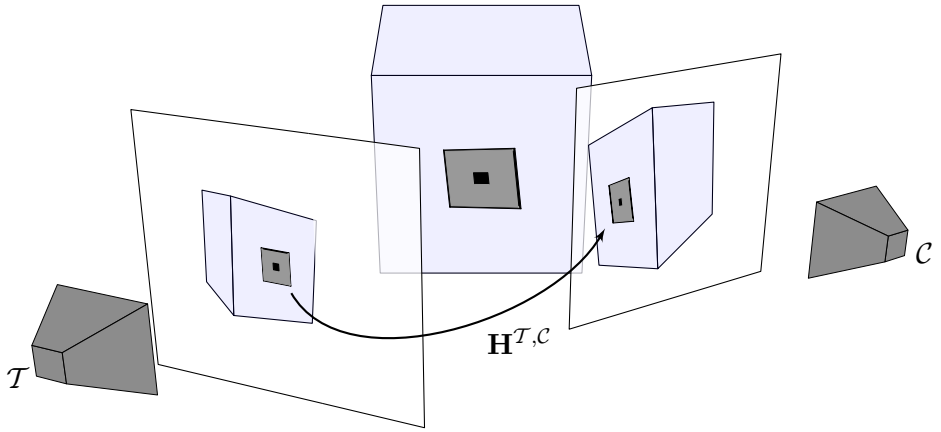


Figure 4.6.: Illustration of the template warping task. A template (square) has been observed during learning from a camera at pose $\mathcal{T}$. During the VSLAM run, the appearance of the template has to be predicted when seen from a camera at pose $\mathcal{C}$. Since we assume that the neighborhood of a point feature is planar, the function relating the pixels of both views can be described by a homography $\mathbf{H}^{\mathcal{T},\mathcal{C}}$.

4.3.3. Template Warping

During learning, each point feature template $\boldsymbol{t}_i$ is initialised by a camera with pose $\mathcal{T}_i$ (see section 4.2). For each point feature template, the position $\mathbf{p}$ and surface normal $\mathbf{n}$ relative to $\mathcal{T}_i$ is stored, as well as the feature center and outline in the camera image.

To predict the appearance of the point feature templates in the current camera image (with the current camera located at $\mathcal{C}$) it is necessary to compute the homographies $\mathbf{H}^{\mathcal{C},\mathcal{T}_i}$ mapping pixels from the feature templates to the current camera image (see figure 4.3.3). This procedure is the same for every feature template, thus we will omit the subscript i in the following.

First, we need to know the position of the template camera with respect to the current camera frame:

$$\mathcal{T}^{\mathcal{C}} = \mathcal{C}^{-1} \cdot \mathcal{T} = \left(\boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}}), \boldsymbol{\rho}_{\mathcal{C}}^{-1} \cdot \boldsymbol{\rho}_{\mathcal{T}}\right) \tag{4.19}$$

The surface a point feature lies on is assumed to be locally planar. The respective plane is defined by the point feature position $\mathbf{p}^{\mathcal{T}}$ and the surface normal $\mathbf{n}^{\mathcal{T}}$. All points $\mathbf{q}^{\mathcal{T}}$ on that plane satisfy:

$$\mathbf{n}^{\mathsf{T}}\mathbf{q} = \mathbf{n}^{\mathsf{T}}\mathbf{p} \tag{4.20}$$

(The superscript $\mathcal{T}$ has been omitted for better readability.)

Let's assume for a moment we normalised the image points of the template camera to homogeneous coordinates $\mathbf{u}^{\mathcal{T}}$ taking into account the camera calibration $\mathbf{K}_{\boldsymbol{T}}$. Each of these points define a ray $k\mathbf{u}$ (through the camera center, which is the origin in $\mathcal{T}$) that intersects the plane defined by the feature template. The point of intersection can be found by inserting the ray definition in the equation above:

$$k\mathbf{n}^{\mathsf{T}}\mathbf{u} = \mathbf{n}^{\mathsf{T}}\mathbf{p} \tag{4.21}$$

which yields:

$$k = \frac{\mathbf{n}^{\mathsf{T}}\mathbf{p}}{\mathbf{n}^{\mathsf{T}}\mathbf{u}} \tag{4.22}$$

The point $k\mathbf{u}$ can now be transformed into the current camera frame $\mathcal{C}$:

$$\mathcal{T}^{\mathcal{C}}(k\mathbf{u}) = \boldsymbol{\rho}_{\mathcal{C}}^{-1} \cdot \boldsymbol{\rho}_{\mathcal{T}} \frac{\mathbf{n}^{\mathsf{T}}\mathbf{p}}{\mathbf{n}^{\mathsf{T}}\mathbf{u}}\mathbf{u} + \boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}}) \tag{4.23}$$

$$= \boldsymbol{\rho}_{\mathcal{C}}^{-1} \left(\boldsymbol{\rho}_{\mathcal{T}} \frac{\mathbf{n}^{\mathsf{T}}\mathbf{p}}{\mathbf{n}^{\mathsf{T}}\mathbf{u}}\mathbf{u} + (\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}}) \right) \tag{4.24}$$

We can interpret the result of this transformation as the homogeneous representation of the normalised image coordinates of $(k\mathbf{u})$ in the current camera image. Since homogeneous coordinates are defined up to scale we can simplify the above expression by multiplying it with $\mathbf{n}^{\mathsf{T}}\mathbf{u}$:

$$\mathcal{T}^{\mathcal{C}}(k\mathbf{u})\mathbf{n}^{\mathsf{T}}\mathbf{u} = \boldsymbol{\rho}_{\mathcal{C}}^{-1} \left(\boldsymbol{\rho}_{\mathcal{T}} \frac{\mathbf{n}^{\mathsf{T}}\mathbf{p}}{\mathbf{n}^{\mathsf{T}}\mathbf{u}}\mathbf{u} + (\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}}) \right) \mathbf{n}^{\mathsf{T}}\mathbf{u} \tag{4.25}$$

$$= \boldsymbol{\rho}_{\mathcal{C}}^{-1} \left(\boldsymbol{\rho}_{\mathcal{T}} \mathbf{n}^{\mathsf{T}}\mathbf{p}\mathbf{u} + (\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}})\mathbf{n}^{\mathsf{T}}\mathbf{u} \right) \tag{4.26}$$

$$= \boldsymbol{\rho}_{\mathcal{C}}^{-1} \left(\boldsymbol{\rho}_{\mathcal{T}} \mathbf{n}^{\mathsf{T}}\mathbf{p} + (\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}})\mathbf{n}^{\mathsf{T}} \right) \mathbf{u} \tag{4.27}$$

Given that, we have obtained a function mapping homogeneous normalised image coordinates $\mathbf{u}$ of the template camera image to homogeneous normalised image coordinates of the current camera image. This yields the definition of the sought homography between the camera images:

$$\mathbf{H}_N^{\mathcal{C},\mathcal{T}} = \boldsymbol{\rho}_{\mathcal{C}}^{-1} \left(\boldsymbol{\rho}_{\mathcal{T}} \mathbf{n}^{\mathsf{T}}\mathbf{p} + (\mathbf{t}_{\mathcal{T}} - \mathbf{t}_{\mathcal{C}})\mathbf{n}^{\mathsf{T}} \right) \tag{4.28}$$

Note, that this homography assumes that the image coordinates have been normalised (denoted by the subscript N). To yield the final homography taking into account the calibrations of both the template camera and the current camera we have to pad the normalised homography with the respective fundamental matrices:

$$\mathbf{H}^{\mathcal{C},\mathcal{T}} = \mathbf{K}_{\boldsymbol{C}} \mathbf{H}_N^{\mathcal{C},\mathcal{T}} \mathbf{K}_{\boldsymbol{T}}^{-1} \tag{4.29}$$

4.3.4. Feature Measurements

The measurement of point features in the object model is very similar to the measurement of the point features described in section 3.2. We give a short summary here, basically pointing out the differences to the point feature model.

Given the predicted measurements regions $\boldsymbol{Z}_i$ and the homographies describing the template view warping we are able to read the actual measurements $\mathbf{z}_i$. The measurements can be carried out for each feature individually as long as the layout of the resulting $\mathbf{z}$ is consistent with the prediction of the measurement function $h(.)$.

This is done by searching point feature templates in the camera images. For that, we search in an area around the predicted position of the point feature to account for the uncertainty of the state estimate. The area is the 3σ uncertainty ellipsoid of the point feature position projected to the camera images that we derived in section 4.3.1. For each point feature, we search for the ZSSD response of its corresponding feature template within this area. To accelerate this procedure, we restrict the possible point feature template positions to corners in the image, since the point feature templates have been initialised on corners as well. If the ZSSD minimum falls below a threshold in the left and right image we consider the point feature to be found and insert the position of the ZSSD minimum in the measurement vector $\mathbf{z}$.

For each not measured point feature (non-candidates or not measured candidates) we fill the respective components of $\mathbf{z}$ with zeros. We will see in the following how we make sure that these "measurements" do not contribute to the state estimate.

4.3.5. Outlier Removal

The measurements obtained by the procedure explained above are not only corrupted by noise that is accounted for in the measurement model. In fact, we get measurement errors that are not explained by the model because of repeated structures in the camera images, wrong state estimates (and therefore wrong predictions), as well as errors originating from violations of the model assumptions like non-planar surfaces, moving objects, illumination changes and non-Lambertian surfaces. These false measurements that are not explained by the measurement model are called *outliers*.

Outliers pose a problem to the IEKF procedure. The IEKF requires the system under consideration to be a Gaussian linear system. In particular, that means that the expected measurements are modelled as a Gaussian distribution. It is due to the nature of the Gaussian distribution that outliers are not explained. In fact, outliers influences the estimation of the real state quadratically. Therefore we need a robust way of getting rid of these outliers before they get passed on to the IEKF updates.

The basic observation for removing outliers is that they usually are not consistent with the other measurements. In terms of the measurement model that means that a measurement without outliers is more likely than with them. However, finding the set of inliers that maximise the measurement likelihood is an NP problem due to the number of possible combinations. The *joint compatibility branch and bound* (JCBB) method provides a generic method to solve the *data association* problem, i.e., the mapping of features to measurements and the rejection of false measurements [29]. In our implementation, however, we use a simplified version of the JCBB method [7]. This version just aims to find the set of inliers, i.e., it does not solve the data association problem. We justify this choice by the observation that we hardly ever encounter wrong data associations.

The outliers found by the JCBB method are just discarded. The corresponding components of $\mathbf{z}$ are filled with zeros and the features are treated as if they would not have been measured at all.

4.4. IEKF Update

To be able to use the measurements we made so far for the state update we need to interface them to the IEKF update procedure. For that we have to provide the state transition function $g(.)$, the measurement function $h(.)$, and the Jacobians $\mathbf{G}$ and $\mathbf{H}$ of both functions (see section 2.1.3). Since we assume that meta features do not move, the state transition function $g(.)$ and its Jacobian $\mathbf{G}$ remain the same as in section 3.2. In the following, we will derive the Jacobian $\mathbf{H}$ of $h(.)$ by $\mathbf{x}$.

As pointed out in subsection 4.3.4, the measurement function $h(.)$ can be decomposed in functions of the currently tracked objects:

$$\mathbf{H} = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial h_1(\mathbf{x})}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}} \end{bmatrix} \tag{4.30}$$

Here, k is the number of object poses in the state and $h_j(.)$ is the measurement function for object descriptor $\boldsymbol{o}_j$. The Jacobians of these functions can be further decomposed into Jacobians of the individual point feature measurements:

$$\frac{\partial h_j(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial h_{j,1}(\mathbf{x})}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial h_{j,n(j)}(\mathbf{x})}{\partial \mathbf{x}} \end{pmatrix} \tag{4.31}$$

Here, $n(j)$ is the number of point features that belong to object descriptor $\boldsymbol{o}_j$ and $h_{j,i}(.)$ is the measurement function for the point feature position $\vec{p}_i$ of object descriptor $\boldsymbol{o}_j$. As we will see soon, the Jacobian of $h_{j,i}(.)$ by $\mathbf{x}$ does not depend on other entities than the point feature position $\vec{p}_i$ of object descriptor $\boldsymbol{o}_j$. Therefore, we will omit the subscripts j and i in the following to enhance readability.

As stated above, we need to build the Jacobian of $h(.)$ with respect to the state $\mathbf{x}$. However, the measurement of a single point feature does not depend on the whole state but only on the current estimate of the camera pose $\mathcal{C}$ and the current estimate of the pose $\mathcal{O}$ of the object the feature belongs to.

$$\frac{\partial h}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial h}{\partial \mathcal{C}} & 0 & \cdots & 0 & \frac{\partial h}{\partial \mathcal{O}} & 0 & \cdots \end{bmatrix} \tag{4.32}$$

The partial derivative of $h(.)$ by the current pose estimates $\mathcal{C}$ and $\mathcal{O}$ can be further separated into the translation part $\mathbf{t}$ and the rotation part $\boldsymbol{\rho}$ of the respective pose:

$$\frac{\partial h}{\partial \mathcal{C}} = \begin{bmatrix} \frac{\partial h}{\partial \mathbf{t}_\mathcal{C}} & \frac{\partial h}{\partial \boldsymbol{\rho}_\mathcal{C}} \end{bmatrix} \tag{4.33}$$

$$\frac{\partial h}{\partial \mathcal{O}} = \begin{bmatrix} \frac{\partial h}{\partial \mathbf{t}_\mathcal{O}} & \frac{\partial h}{\partial \boldsymbol{\rho}_\mathcal{O}} \end{bmatrix} \tag{4.34}$$

$$\frac{\partial h}{\partial \mathbf{t}_\mathcal{C}} = \frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\partial \mathbf{p}^\mathcal{C}}{\partial \mathbf{t}_\mathcal{C}} \qquad \text{by (4.4)} \tag{4.35}$$

$$= \frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\boldsymbol{\rho}_\mathcal{C}^{-1}(\mathbf{k})}{\partial \mathbf{k}} \frac{\partial \mathbf{k}}{\partial \mathbf{t}_\mathcal{C}} \qquad \text{by (4.10)} \tag{4.36}$$

The Jacobian of $\pi_S(.)$ can be found in appendix C, equation C.4. The Jacobian of the rotation function of a point $\mathbf{k}$ can be found in appendix C, equation C.27. The last part of the decomposition of the Jacobian, namely the Jacobian of $\mathbf{k}$ by $t_\mathcal{C}$, is the negative unit matrix (compare the definition of $\mathbf{k}$ in equation 4.10) and therefore cancels out just changing the sign of the Jacobian:

$$\frac{\partial h}{\partial \mathbf{t}_\mathcal{C}} = -\frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\boldsymbol{\rho}_\mathcal{C}^{-1}\mathbf{k}}{\partial \mathbf{k}} \tag{4.37}$$

In a similar manner, the Jacobian of $h(.)$ by $\boldsymbol{\rho}_\mathcal{C}$ can be derived:

$$\frac{\partial h}{\partial \boldsymbol{\rho}_\mathcal{C}} = \frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\partial \mathbf{p}^\mathcal{C}}{\partial \boldsymbol{\rho}_\mathcal{C}} \tag{4.38}$$

$$= \frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\partial \boldsymbol{\rho}_\mathcal{C}^{-1}(\mathbf{k})}{\partial \boldsymbol{\rho}_\mathcal{C}} \tag{4.39}$$

$$= \frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\partial \boldsymbol{\rho}_\mathcal{C}^{-1}(\mathbf{k})}{\partial \boldsymbol{\rho}_\mathcal{C}^{-1}} \frac{\partial \boldsymbol{\rho}_\mathcal{C}^{-1}}{\partial \boldsymbol{\rho}_\mathcal{C}} \tag{4.40}$$

$$= -\frac{\partial \pi_S(\mathbf{p}^\mathcal{C})}{\partial \mathbf{p}^\mathcal{C}} \frac{\partial \boldsymbol{\rho}_\mathcal{C}^{-1}(\mathbf{k})}{\partial \boldsymbol{\rho}_\mathcal{C}^{-1}} \tag{4.41}$$

The first part of this decomposition is the same as in equation 4.37. The second part can be found in appendix C in equation C.16.

As already pointed out in equation 4.34, the Jacobian of $h(.)$ by the pose of the meta feature can also be decomposed in the Jacobians by the translation $\mathbf{t}_{\mathcal{O}}$ and the rotation $\boldsymbol{\rho}_{\mathcal{O}}$:

$$\frac{\partial h}{\partial \mathbf{t}_{\mathcal{O}}} = \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\partial \mathbf{p}^{\mathcal{C}}}{\partial \mathbf{t}_{\mathcal{O}}} \tag{4.42}$$

$$= \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{k})}{\partial \mathbf{k}} \frac{\partial \mathbf{k}}{\partial \mathbf{t}_{\mathcal{O}}} \tag{4.43}$$

$$= \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{k})}{\partial \mathbf{k}} \tag{4.44}$$

$$\frac{\partial h}{\partial \boldsymbol{\rho}_{\mathcal{O}}} = \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\partial \mathbf{p}^{\mathcal{C}}}{\partial \boldsymbol{\rho}_{\mathcal{O}}} \tag{4.45}$$

$$= \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\partial \boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{k})}{\partial \boldsymbol{\rho}_{\mathcal{O}}} \tag{4.46}$$

$$= \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\partial \boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{k})}{\partial \mathbf{k}} \frac{\partial \mathbf{k}}{\partial \boldsymbol{\rho}_{\mathcal{O}}} \tag{4.47}$$

Considering the definition of $\mathbf{k}$ (see equation 4.10) it is evident that its Jacobian by $\boldsymbol{\rho}_{\mathcal{O}}$ only depends on $\boldsymbol{\rho}_{\mathcal{O}}(\mathbf{p}^{\mathcal{O}})$ for which we provide the Jacobian in appendix C in equation C.16. Thus we can present the last component of the Jacobian of the measurement function:

$$\frac{\partial h}{\partial \boldsymbol{\rho}_{\mathcal{O}}} = \frac{\partial \pi_S(\mathbf{p}^{\mathcal{C}})}{\partial \mathbf{p}^{\mathcal{C}}} \frac{\partial \boldsymbol{\rho}_{\mathcal{C}}^{-1}(\mathbf{k})}{\partial \mathbf{k}} \frac{\partial \boldsymbol{\rho}_{\mathcal{O}}(\mathbf{p}^{\mathcal{O}})}{\partial \boldsymbol{\rho}_{\mathcal{O}}} \tag{4.48}$$

The derivation of the Jacobian so far assumed that each of the point features got measured. As we argued earlier, this is neither possible nor desirable. The respective entries in $\mathbf{z}$ of the not measured point features have been filled with zeros, as described in section 4.3.4. Since there is no functional dependency between these zero fill-ins and the current state, the derivation of these components by the state is zero, i.e.,

$$\frac{\partial h}{\partial \mathbf{x}} = \mathbf{0}_{[3 \times n]}, \tag{4.49}$$

where n is the size of the state.

5. Object Recognition

Object recognition is the task to detect a specific object and its pose in one or more images. The recognition needs to be robust with respect to viewpoint and illumination changes. In contrast to plain object *detection* we are also interested in the pose of the object, i.e., its two- or three-dimensional position and orientation.

A lot of approaches for object recognition exist. Usually, feature descriptors are extracted from the image and used for the pose estimation. The feature descriptors used reach from raw pixel values (e.g., local self-similarities [34]) over interest point descriptors (e.g., SIFT [25], SURF [3], or Histogram Intensity Patches [38]) to contours (represented explicitly, e.g., as B-splines [9] or implicitly as level-sets [8]). While most of the approaches are shallow, i.e., try to estimate the pose directly on the feature image positions, there are more sophisticated approaches that group the features hierarchically (e.g., by composition of simpler features [14] or by means of an image grammar [42]).

However, not all of them are suitable for real-time applications. Especially statistical models suffer from large computation times. The most popular choice for fast object recognition are shallow, point feature based approaches.

In general, a point feature based approach consists of the following steps: In an offline learning step, an object description is generated. This can be a complete 3D reconstruction or just sample images of the object. The object description is searched for interest points, i.e., visually remarkable points on the objects surface, from which feature descriptors are getting extracted. Later, during the recognition process, the same technique is used to extract interest points and their feature descriptors from the current camera image. This descriptors are compared to the ones obtained during learning. If there is enough similarity, *matches* are made up: Each descriptor from the camera image is assigned to one descriptor of the object. Based on these correspondences together with the respective positions of the features belonging to the descriptors, the pose of the object can be estimated using an outlier robust method.

This approach was made popular by Lowe with his work on robust feature descriptors [25]. The fact that recent publications still follow this outline (although with different feature descriptors and pose estimators) depicts the success of this approach [38].

We use a very similar method for object recognition as the one introduced by Lowe. For that, we extended the model described in the previous chapter to store SIFT feature descriptors as well. This extension is introduced in the following section. The learning and recognition processes of the objects are presented in sections 5.2 and 5.3. We use the outcome of the recognition process to initialise object poses in the state. In section 5.4 we describe in detail how we accomplish that with consideration of the uncertainty of the system at the time of observation. Finally, we show in section 5.5 how we can use pose estimates of already initialised objects as measurements in the IEKF framework.

5.1. Model Extension

We extended the object model introduced in chapter 4 in two ways to support online object recognition: The object descriptor contains now more distinctive point feature descriptors besides the feature templates and the model was combined with the bundle feature model described in section 3.4.

The additional feature descriptors are necessary to estimate the pose of an object given one image only. The feature templates that are used during the VSLAM run to perform the measurements are not sufficient for two reasons:

First, they are not robust to viewpoint changes. In order to find point features by matching feature templates we need to have a rough idea from which angle we see the feature template to predict its appearance. This is not possible if we are trying to find an object with unknown pose.

Second, matching feature templates is too slow for real time operation. Without rough knowledge of the object pose we would need to search the whole camera image for *each* feature template of *each* object. Feature template matching is just possible if we can restrict the search area and search for specific feature templates only.

To perform the object recognition we decided to use SIFT feature descriptors. Although originally SIFT [25] denotes a feature *descriptor*, it is commonly a synonym for a robust method for both interest point detection and feature description. Interest points found by this method are often called *SIFT features*.

SIFT feature descriptors have the advantage of being invariant to rotation, scale, and some amount of affine transformation. Furthermore, they behave quite robust against illumination changes.

Definition 5.1: (SIFT descriptor) A SIFT descriptor is a 128-dimensional point feature descriptor $\boldsymbol{s}$ describing the visual appearance of the local neighborhood of a point feature in a scale and rotation invariant fashion.

The interest points for SIFT descriptors are found at minima and maxima of a so-called *difference of Gaussian* (DoG) pyramid of the image under consideration. This pyramid is constructed as follows: The image is repeatedly downsampled by a factor of two using a Gaussian kernel. The difference between two subsequent versions of the image build one layer of the pyramid[1]. Local minima and maxima in this pyramid are found by considering all neighboring pixels in the same image and in the previous and subsequent layer. The interest point positions found by this method are refined by interpolating the surrounding pixels: The DoG image containing an extrema is approximated by a second order Taylor expansion around the found position and sought for extrema itself [26]. The derivatives of the DoG images used for the Taylor expansion also give valuable information about the quality of the found interest point. If its contrast is too low or it lies on an edge, it is discarded. Finally, for the remaining interest points, the principal orientation is determined to ensure rotational invariance.

The SIFT descriptor is a 128-dimensional vector describing the local properties of the found interest points considering their scale and orientation. It consists of 16 histograms which are arranged around the interest point in a 4×4 array. Each histogram is computed from 16 samples of the principal orientation and the derivation magnitude

[1] In fact, in terms of the image size in pixels the result is not a pyramid but a stack. The term pyramid refers to the fact that the scale of the images is halved with every step to the top of the stack. However, since this is not a pyramid as well, it is a metaphor only.

of the underlying pixels. These samples are assigned to 8 bins, thus the size of the descriptor is 16×8.

We extract SIFT descriptors both during learning and recognition. While the performance of this operation during learning is not so important, it is crucial during the recognition process. For that we decided to use the SiftGPU library by Changchang Wu [46], which performs the extraction on the graphics card.

Definition 5.2: (Extended Object Descriptor) An extended object descriptor is a tuple

$$\boldsymbol{o} = (\boldsymbol{T}, \boldsymbol{S}, \boldsymbol{P}, \boldsymbol{Q}) \tag{5.1}$$

of feature templates $\boldsymbol{T}$, SIFT feature descriptors $\boldsymbol{S}$ and sets of the respective 3D positions $\boldsymbol{P}$ and $\boldsymbol{Q}$ of the point features that belong to the descriptors $\boldsymbol{T}$ and $\boldsymbol{S}$:

$$\boldsymbol{T} = \{\boldsymbol{t}_1, \cdots, \boldsymbol{t}_n\} \tag{5.2}$$
$$\boldsymbol{S} = \{\boldsymbol{s}_1, \cdots, \boldsymbol{s}_m\} \tag{5.3}$$
$$\boldsymbol{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_n\} \tag{5.4}$$
$$\boldsymbol{Q} = \{\mathbf{q}_1, \cdots, \mathbf{q}_m\} \tag{5.5}$$

where n is the number of feature templates and m is the number of SIFT feature descriptors for the object descriptor $\boldsymbol{o}$.

The second extension to the model is the combination of the object model with the bundle feature model. This was necessary to allow the system to keep track of the camera pose even if no object has been observed so far or the tracked objects are currently invisible. For an introduction to the bundle feature model see section 3.4.

Definition 5.3: (State) A state $\mathbf{x}_t$ of the extended object model at time t is a vector comprising the current camera pose $\mathcal{C}_t$, l anchor poses $\mathcal{A}^i$, m inverse depth parameters d^j, and n object poses $\mathcal{O}^k$:

$$\mathbf{x}_t = \left(\mathcal{C}_t, \mathcal{A}^1, \ldots, \mathcal{A}^l, d^1, \ldots, d^m, \mathcal{O}^1, \ldots, \mathcal{O}^n\right)^\mathsf{T}.$$

5.2. Learning

The aim of the learning is to build an object database that stores SIFT feature descriptors with the positions of the respective point features for each object. The positions of the point features are to be given relative to the pose of the object they belong to. Since this is very similar to the learning of the point feature templates (see section 4.2) we use the same image sequences for that.

Definition 5.1: (Object Database) An object database is a set of object descriptors

$$D = \{\boldsymbol{o}_1, \cdots, \boldsymbol{o}_n\} \tag{5.6}$$

where n is the number of objects in the database.

The VSLAM system that was used for learning the meta features was modified such that it additionally searches for SIFT interest points in every camera frame. The found image position and the descriptor is logged and the Slamdunk evaluation framework is used afterwards to trace the ground truth position again.

The learning procedure is carried out for each object individually, thus constituting the object database.

5.3. Recognition Process

The task of the recognition process is to estimate the pose of an object in a camera image. This task can be carried out independently of the normal VSLAM operation and runs in an own thread. Periodically, the VSLAM thread selects a *keyframe* $\boldsymbol{k}$ that is passed to the recognition thread. A keyframe consists of the current camera pose and camera image. We will refer to this special camera pose as an *anchor* because it is technically the same as an anchor in the bundle feature model. The recognition thread performs the pose estimation on the given keyframe and returns its result, if there is one, to the VSLAM process. After processing this result, a new keyframe is selected.

Definition 5.1: (Keyframe) A keyframe consist of an anchor pose $\mathcal{A}$ and the camera image $I_{\mathcal{A}}$ observed at the anchor's position:

$$\boldsymbol{k} = (\mathcal{A}, I_{\mathcal{A}})$$

The anchor pose $\mathcal{A}$ is active, i.e., it appears in the state.

The pose estimation runs in two steps: First, a candidate object descriptor $\boldsymbol{o}_c$ from the object database is sought. After that, the pose $\mathcal{O}^{\mathcal{A}}$ of the corresponding object with respect to the anchor pose $\mathcal{A}$ is estimated.

For the candidate selection, SIFT interest points are extracted from $I_{\mathcal{A}}$ providing a set of descriptors $\tilde{\boldsymbol{S}}$ and corresponding image positions $\tilde{\boldsymbol{P}}$:

$$\tilde{\boldsymbol{S}} = \{\tilde{\boldsymbol{s}}_1, \cdots, \tilde{\boldsymbol{s}}_n\} \tag{5.7}$$

$$\tilde{\boldsymbol{P}} = \{\tilde{\mathbf{p}}_1^{\mathcal{I}}, \cdots, \tilde{\mathbf{p}}_n^{\mathcal{I}}\} \tag{5.8}$$

where n is the number of found SIFT interest points. The nearest neighbors of the SIFT descriptors $\tilde{\boldsymbol{S}}$ with respect to the Euclidean distance are then sought in each set of SIFT descriptors $\boldsymbol{S}_i$ of the object database D. Whenever two SIFT descriptors are close enough to each other they build a *match*. Thus we can provide a set of matches M_i for each object descriptor $\boldsymbol{o}_i$ to the currently visible SIFT descriptors. For that, we first create a set of indices of nearest neighbors for each SIFT descriptor found in the image:

$$N_i = \{(\mathrm{nn}_i(1), 1), \cdots, (\mathrm{nn}_i(n), n)\} \tag{5.9}$$

The nearest neighbor index $\mathrm{nn}_i(j)$ for SIFT descriptor j is given as

$$\mathrm{nn}_i(j) = \arg\min_k \ |\boldsymbol{s}_k^i - \tilde{\boldsymbol{s}}_j|, \tag{5.10}$$

where $\boldsymbol{s}_k^i$ denotes the SIFT descriptor k of object descriptor $\boldsymbol{o}_i$. The set of matches M_i is constructed by filtering out neighbors who's distance exceeds a threshold d_{min}:

$$M_i = \{(k, j) \in N_i \,\big|\, |\boldsymbol{s}_k^i - \tilde{\boldsymbol{s}}_j| \leq d_{min}\} \tag{5.11}$$

Input : 2D-3D point correspondences C
Output : Pose estimate $\mathcal{O}$
Parameters: number of iterations k_{max}, inlier threshold δ_{min}

for $k \leftarrow 1$ **to** k_{max} **do**
  // randomly select three point correspondences
  $R_k \leftarrow$ `RandomSelect(`C, 3`);`
  // calculate pose
  $\mathcal{O}_k \leftarrow$ `GetPose(`R_k`);`
  // get score of this pose
  $s_k \leftarrow \sum_{(\mathbf{q}^{\mathcal{O}}, \tilde{\mathbf{p}}^{\mathcal{I}}) \in C} |\pi(\mathcal{O}_k(\mathbf{q}^{\mathcal{O}})) - \tilde{\mathbf{p}}^{\mathcal{I}}|^2$;
  // save best estimate so far
  if $s_k \leq s^*$ **then**
    $s^* \leftarrow s_k$;
    $\mathcal{O}^* \leftarrow \mathcal{O}_k$;
  end
end
// compute set of inliers
$I \leftarrow \left\{ (\mathbf{q}^{\mathcal{O}}, \tilde{\mathbf{p}}^{\mathcal{I}}) \in C \;\middle|\; |\pi(\mathcal{O}^*(\mathbf{q}^{\mathcal{O}})) - \tilde{\mathbf{p}}^{\mathcal{I}}| \leq \delta_{min} \right\}$;
// calculate pose from inliers
$\mathcal{O} \leftarrow$ `GetPose(`I`);`
return $\mathcal{O}$

Listing 9: The MSAC algorithm for pose estimation.

The object descriptor $\boldsymbol{o}_c$ with the highest number of matches per SIFT feature descriptor is considered present in the current image.

$$c = \arg\max_i \frac{|M_i|}{|S_i|} \tag{5.12}$$

From the matches M_c we generate a set of point correspondences C between the 3D positions $\boldsymbol{Q}_c$ of the SIFT interest points $\boldsymbol{S}_c$ and the 2D image positions $\tilde{\boldsymbol{P}}$ of the SIFT descriptors $\tilde{\boldsymbol{S}}$.

$$C = \{(\mathbf{q}_k^{\mathcal{O}}, \tilde{\mathbf{p}}_j^{\mathcal{I}}) \mid (k, j) \in M_i\} \tag{5.13}$$

Here, $\mathcal{O}$ is the coordinate frame of the candidate object and $\mathcal{I}$ is the image coordinate frame.

Given these correspondences we can estimate the pose of the object relative to the anchor pose $\mathcal{A}$, yielding $\mathcal{O}^{\mathcal{A}}$. However, due to the outliers that found their way into C we will need a robust method for that. Here, we decided to use MSAC [41], a variant of the RANSAC [15] algorithm, for the pose estimation. The complete algorithm is given in listing 9.

The basic idea of the MSAC algorithm is to repeatedly and randomly select three point correspondences that are used to calculate the pose of the object. Each of the poses obtained this way is considered a hypothesis. A hypothesis is the better, the smaller the projection error of all point correspondences is. The projection error of a point correspondence is the distance between its 2D image position and the projected 3D position. The projection is carried out with respect to the pose estimate, thus scoring each hypothesis. The best hypothesis after a fixed number of iterations is then used to

determine the set of inliers, i.e., all correspondences with a projection error smaller than a threshold. Eventually, only these inliers are used to estimate the pose of the object.

The calculation of the pose by a set of point correspondences carried out by the function `GetPose` in listing 9 is twofold: If there are just three point correspondences, the pose can be determined in a closed fashion. We use the method introduced by Fischler [15], which transforms the problem of the pose estimation to the solving of a quartic polynomial, thus delivering up to four poses for three matches.

If there are more than three correspondences, in general, there is no unique solution anymore since the dimension of the measurement exceeds the degree of freedom of a pose. In this case the best pose is found as the one minimizing the sum of squared projection errors of all point correspondences:

$$\mathcal{O}^{\mathcal{A}} = \underset{\mathcal{O}^{\mathcal{A}} \in SE(3)}{\arg\min} \sum_{(\mathbf{q}^{\mathcal{O}}, \tilde{\mathbf{p}}^{\mathcal{I}})} \left| \pi \left(\mathcal{O}^{\mathcal{A}} \left(\mathbf{q}^{\mathcal{O}} \right) \right) - \tilde{\mathbf{p}}^{\mathcal{I}} \right|^2 \tag{5.14}$$

The problem of estimating a pose given $n > 3$ point correspondences is commonly known as the PnP-problem in the literature. In our work we implemented the method introduced by Moreno-Noguer et al. [28]. In contrast to other approaches, their method is non-iterative and ensures a complexity of $O(n)$.

5.4. Object Initialisation

Whenever the object recognition thread delivers a pose estimate $\mathcal{O}^{\mathcal{A}}$ for an object that has not been observed so far, we add the respective object pose $\mathcal{O}$ to the state $\mathbf{x}$. The insertion involves changing $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ accordingly to the uncertainties and correlation of the state.

Since we want to keep track of the world coordinates of the object, we need to transform the estimated pose to the world coordinate frame. The estimated pose is given with respect to the anchor pose $\mathcal{A}$ the object was observed from. Therefore, we have to concatenate both poses to get the desired pose:

$$\mathcal{O} = \overline{h}(\mathcal{A}, \mathcal{O}^{\mathcal{A}}) = \mathcal{A} \circ \mathcal{O}^{\mathcal{A}} \tag{5.15}$$

We will call this transformation the *reverse measurement function* since it provides a part of the state given a measurement.

The reverse measurement function depends on two uncertain values: The anchor pose $\mathcal{A}$ and the pose measurement $\mathcal{O}^{\mathcal{A}}$ both underly uncertainty. The mean and covariance of the anchor pose estimate is given in the state as $\boldsymbol{\mu}_{\mathcal{A}}$ and $\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{A}}$. The mean of the pose measurement is the measurement itself and the uncertainty a designed covariance matrix $\mathbf{R}$ with diagonal entries $\boldsymbol{\xi}$ that account for the measurement noise of each component of the object pose:

$$\mathbf{R} = \boldsymbol{\xi} \times \mathbf{1}_{[6\times 6]} \tag{5.16}$$

These uncertainties have to be considered when adding the object pose to the state. The object pose in world coordinates, as a function of $\mathcal{A}$ and $\mathcal{O}^{\mathcal{A}}$, inherits the uncertainty of both arguments, i.e., of $\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{A}}$ and $\mathbf{R}$. The resulting covariance of $\mathcal{O}$ is the

sum of both covariances transformed via linearised versions of $\overline{h}(.)$. The linearisation is carried out at the means of the estimates of $\mathcal{A}$ and $\mathcal{O}^{\mathcal{A}}$:

$$\mathbf{\Sigma}_{\mathcal{O}\mathcal{O}} = \mathbf{A}\mathbf{\Sigma}_{\mathcal{A}\mathcal{A}}\mathbf{A}^{\mathsf{T}} + \mathbf{O}\mathbf{R}\mathbf{O}^{\mathsf{T}} \tag{5.17}$$

with

$$\mathbf{A} = \left.\frac{\partial g(\mathcal{A}, \mathcal{O}^{\mathcal{A}})}{\partial \mathcal{A}}\right|_{\boldsymbol{\mu}_{\mathcal{A}}} \tag{5.18}$$

$$\mathbf{O} = \left.\frac{\partial g(\mathcal{A}, \mathcal{O}^{\mathcal{A}})}{\partial \mathcal{O}^{\mathcal{A}}}\right|_{\mathcal{O}^{\mathcal{A}}} \tag{5.19}$$

The Jacobians $\mathbf{A}$ and $\mathbf{O}$ for the composition of two poses are given in appendix C in equations C.49 and C.54.

On creation of the anchor its pose was correlated with the state and updated with the other entities in the state. That implies that the pose estimation is correlated with the state as well. This correlation can be determined by propagating the anchor pose correlation through the reverse measurement function using the linearisation with respect to $\mathcal{A}$ again:

$$\mathbf{\Sigma}_{\mathcal{O}\mathbf{x}} = \mathbf{A}\mathbf{\Sigma}_{\mathcal{A}\mathbf{x}} \tag{5.20}$$

Here, $\Sigma_{\alpha\mathbf{x}}$ denotes the correlation matrix of the entity α with the whole state $\mathbf{x}$.

The new state with the inserted object pose $\mathcal{O}$ is now given as:

$$\boldsymbol{\mu}' = \begin{pmatrix} \boldsymbol{\mu} \\ \mathcal{O} \end{pmatrix} \tag{5.21}$$

$$\mathbf{\Sigma}' = \begin{bmatrix} \mathbf{\Sigma} & \mathbf{\Sigma}_{\mathcal{O}\mathbf{x}}^{\mathsf{T}} \\ \mathbf{\Sigma}_{\mathcal{O}\mathbf{x}} & \mathbf{\Sigma}_{\mathcal{O}\mathcal{O}} \end{bmatrix} \tag{5.22}$$

After the insertion of the object pose, the keyframe anchor gets removed from the state. The only purpose of the anchor was to track the pose of the camera observing the object and is not needed anymore. For the removal, $\mu_{\mathcal{A}}$ and $\Sigma_{\mathcal{A}\mathbf{x}}$ are just getting deleted from the pdf of $\mathbf{x}$. Note that $\mathcal{O}$ is still correlated to the state via the other entities the anchor was previously correlated to.

5.5. Pose Measurements

The pose estimates of the recognition thread can also be used to deliver repeated measurements of the meta features poses. This can help to refine the camera pose estimate and even relocate the system once it lost track.

In contrast to feature measurements via active vision, the pose estimation does not rely on sufficiently good state estimates. As with the initialisation of meta features, the pose measurement runs independently of the VSLAM system. Changes to the state estimate do not affect the recognition performance.

Pose estimates from the recognition process are handled as measurements. This introduces a new type of measurements, namely *pose measurements*. Pose measurements are always relative to the keyframes in which they were made.

Definition 5.1: (Pose Measurements) A pose measurement $\hat{\mathbf{z}}$ is the relative pose $\mathcal{O}^{\mathcal{A}}$ of an object pose $\mathcal{O}$. A pose measurement is carried out relative to the coordinate frame $\mathcal{A}$ of a keyframe $\boldsymbol{k}$.

The measurement function of the object measurement model was extended for pose measurements. Given the true state $\mathbf{x}$, the ideal pose measurement $\hat{\mathbf{z}}_i$ of an object pose $\mathcal{O}_i$ observed by a keyframe $\boldsymbol{k} = (\mathcal{A}, I_{\mathcal{A}})$ is provided by the pose measurement function $\hat{h}_i(.)$:

$$\hat{\mathbf{z}}_i = \hat{h}_i(\mathbf{x}) \stackrel{\text{def}}{=} \mathcal{A}^{-1}\mathcal{O}_i \tag{5.23}$$

Thus, the measurement function $h(.)$ (introduced in section 4.3.1) now also predicts pose measurements:

$$h(\mathbf{x}) = \begin{pmatrix} h_1(\mathbf{x}) \\ \vdots \\ h_k(\mathbf{x}) \\ \hat{h}_1(\mathbf{x}) \\ \vdots \\ \hat{h}_k(\mathbf{x}) \end{pmatrix} \tag{5.24}$$

To use the measurements in the IEKF, the measurement function has to be linearised. The linearisation is done via a first order Taylor expansion around the current state estimate (as described in section 2.1.3). For that, we need the Jacobian $\mathbf{H}$ of $h(.)$ by $\mathbf{x}$. Expectedly, we can decompose $\mathbf{H}$ in the Jacobians of the individual meta features measurement functions in the same manner as we did in section 4.4.

$$\mathbf{H} = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial h_1(\mathbf{x})}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}} \\ \frac{\partial \hat{h}_1(\mathbf{x})}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial \hat{h}_k(\mathbf{x})}{\partial \mathbf{x}} \end{bmatrix} \tag{5.25}$$

It remains to derive the Jacobian of $\hat{h}_i$ by $\mathbf{x}$. In the following, we omit the subscript i for better readability.

The Jacobian of the measurement function is sparse. It does only depend on the pose $\mathcal{O}$ of the object under consideration and the pose of the anchor $\mathcal{A}$ that was used for the keyframe.

$$\frac{\partial h}{\partial \mathbf{x}} = \begin{bmatrix} 0 & \cdots & 0 & \frac{\partial h}{\partial \mathcal{O}} & 0 & \cdots & 0 & \frac{\partial h}{\partial \mathcal{A}} & 0 & \cdots \end{bmatrix} \tag{5.26}$$

The Jacobian of $\hat{h}(.)$ by $\mathcal{O}$ can be written as:

$$\frac{\partial \hat{h}}{\partial \mathcal{O}} = \frac{\partial \mathcal{A}^{-1} \circ \mathcal{O}}{\partial \mathcal{O}} \tag{5.27}$$

which is given in equation C.54 in appendix C.

The Jacobian of $\hat{h}(.)$ by $\mathcal{A}$ requires some more step due to the inversion of the anchor pose $\mathcal{A}$:

$$\frac{\partial \hat{h}}{\partial \mathcal{A}} = \frac{\partial \mathcal{A}^{-1} \circ \mathcal{O}}{\partial \mathcal{A}} = \frac{\partial \mathcal{A}^{-1} \circ \mathcal{O}}{\partial \mathcal{A}^{-1}} \frac{\partial \mathcal{A}^{-1}}{\partial \mathcal{A}} \tag{5.28}$$

Both Jacobians of the product can be found in appendix C in equations C.49 and C.44.

5.6. Relocation

Whenever the system is considered lost, we can use a pose measurement to relocate the camera. Following a very simple strategy, we consider the VSLAM system as being lost if there were no successful point feature measurements in two consecutive frames.

However, the object recognition thread continues its work. As soon as a pose measurement becomes available we reset the camera pose to the estimated pose of the anchor that was used to perform the object recognition. Since we don't know where the camera was at the time of the key-frame selection we have to estimate the anchor pose. This is done by considering the current estimate of the object pose and its uncertainty.

We can write the new camera pose as a function of the true object pose and the true object pose relative to the camera, i.e., the true pose measurement:

$$\mathcal{C} = f(\mathcal{O}, \mathcal{O}^{\mathcal{C}}) = \mathcal{O} \circ (\mathcal{O}^{\mathcal{C}})^{-1} \tag{5.29}$$

Of course, both the object pose $\mathcal{O}$ and the pose measurement $\mathcal{O}^{\mathcal{C}}$ are just estimates underlying uncertainty. The mean of the new camera pose is retrieved by evaluating $f(.)$ at the position of the mean of the object pose $\boldsymbol{\mu}_{\mathcal{O}}$ and the mean of the pose measurement $\mathcal{O}_{\mathcal{C}}$. To propagate the uncertainties $\mathbf{\Sigma}_{\mathcal{O}\mathcal{O}}$ and $\mathbf{R}$ of the object pose and the pose measurement to the camera pose estimate as well, we linearise $f(.)$ at at this position:

$$\mathbf{\Sigma}_{\mathcal{C}\mathcal{C}} = \mathbf{P}\mathbf{\Sigma}_{\mathcal{O}\mathcal{O}}\mathbf{P}^{\mathsf{T}} + \mathbf{O}\mathbf{R}\mathbf{O}^{\mathsf{T}} \tag{5.30}$$

with

$$\mathbf{P} = \left.\frac{\partial f(\mathcal{O}, \mathcal{O}^{\mathcal{C}})}{\partial \mathcal{O}}\right|_{\boldsymbol{\mu}_{\mathcal{O}}} \tag{5.31}$$

$$\mathbf{O} = \left.\frac{\partial f(\mathcal{O}, \mathcal{O}^{\mathcal{C}})}{\partial \mathcal{O}^{\mathcal{C}}}\right|_{\mathcal{O}^{\mathcal{C}}} \tag{5.32}$$

The Jacobian $\mathbf{P}$ is given in appendix C in equations C.49. The Jacobian $\mathbf{O}$ can be decomposed as:

$$\frac{\partial f(\mathcal{O}, \mathcal{O}^{\mathcal{C}})}{\partial \mathcal{O}^{\mathcal{C}}} = \frac{\partial \mathcal{O} \circ (\mathcal{O}^{\mathcal{C}})^{-1}}{\partial (\mathcal{O}^{\mathcal{C}})^{-1}} \frac{\partial (\mathcal{O}^{\mathcal{C}})^{-1}}{\partial \mathcal{O}^{\mathcal{C}}} \tag{5.33}$$

Both factors can be found in appendix C in equations C.54 and C.44.

Since the pose of the object we measured is also correlated to every other entity in the state we can correlate the new camera pose with the whole state as well. For that we use the linearised version of $f(.)$ again:

$$\mathbf{\Sigma}_{\mathcal{C}\mathbf{x}} = \mathbf{P}\mathbf{\Sigma}_{\mathcal{O}\mathbf{x}} \tag{5.34}$$

Here, $\Sigma_{\alpha\mathbf{x}}$ denotes the correlation matrix of the entity α with the whole state $\mathbf{x}$.

6. Experiments

In this chapter we present experimental results of our approach on both synthetic and real images. For that, we created and recorded seven sequences in which up to two objects are visible. For the synthetic sequences we are able to provide an evaluation of the errors in the camera and object pose estimates.

In the following section we will give details about the experimental setup, i.e., the objects, sequences, and VSLAM configurations that have been used. The results and evaluation of individual VSLAM runs will be presented in section 6.2. We conclude this chapter with a discussion of the results in section 6.3.

6.1. Setup

6.1.1. Objects

To test our implementation we built object descriptors for two objects: A Caro-tin[1] and a packet (see figure 6.1).

Both objects have been rebuilt with POV-Ray by hand to create the object descriptors. For the Caro-tin we built a smooth and a random object descriptor, for the packet just a random descriptor (see section 4.2.2 for details).

6.1.2. Sequences

For the experiments we used two types of sequences: Four synthetic sequences (A, B, C, D) for comparison of the results against a ground-truth and three recorded sequences (RA, RB, RC) to show the performance on real footage. All synthetic sequences have been generated using the Slamdunk evaluation framework, introduced in section 4.2.1.

Despite the still sequences A and RA (in which the camera is not moving at all), all sequences have intentionally been created to include some difficulties to point out weaknesses of our approach and to see how we can overcome them.

The synthetic sequences have been rendered using a simple scene that resembles an office workplace. The scene consists of just a few textured planes. Although this corresponds to our assumption that point features lie on planar surfaces, there are not much interest points to take advantage of that. Thus, without an object as additional feature, robust tracking in this office scene is a challenging task on its own.

All the sequences are stereo sequences. The real sequences have been recorded using a Bumblebee® stereo camera and were rectified according to the calibration of the camera. For the synthetic sequences we used the intrinsic camera parameters of the Bumblebee® camera after rectification. In the sample images given below we show the right camera images only.

[1] Caro is a German coffee substitute made from barley malt and is free of caffeine. It tastes best when prepared with equal portions of water and milk.

(a) real Caro-tin (b) rebuilt Caro-tin

(c) real packet (d) rebuilt packet

Figure 6.1.: The objects used for experiments.

Sequence A

Figure 6.2.: Sample images from sequence A.

In this sequence, the camera is not moving. The Caro-tin object is clearly visible being 55 cm in front of the camera on a desk (see figure 6.2). The length of this sequence is 60 seconds.

This sequence serves for stability tests: The system is expected to perform robustly and constantly for the length of the sequence.

Figure 6.3.: Sample images from sequence B.

Sequence B

In this sequence, the camera is slowly exploring the Caro-tin object from a close distance (see figure 6.3). The camera trajectory was extracted from a real VSLAM run and smoothed afterwards. Thus, the assumptions about the camera motion are hardly violated in this sequence. The length of this sequence is 92 seconds.

However, there are two critical parts in this sequence: First, at about 35 seconds, the camera is approaching the Caro-tin very closely – much closer than the tin was observed during learning. This is to see how scale invariant the feature and pose measurements actually are. Second, at about 78 seconds in the sequence, the object completely disappears from the camera's field of view leaving just a featureless desktop. It should be impossible for the VSLAM system to keep track in these frames. The following reappearance of the object is meant to test the relocation strategy of our implementation.

Sequence C

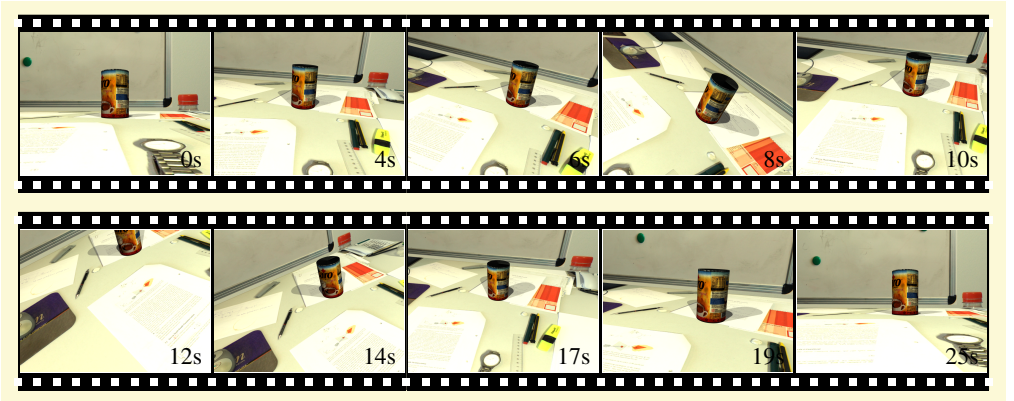


Figure 6.4.: Sample images from sequence C.

This sequence is similar to the previous one, except that the camera is moving much faster and changing directions unexpectedly (see figure 6.4). Also, the distance to the object is bigger. This sequence has a length of just 25 seconds, although the camera is travelling further than in the previous sequence.

We use this sequence to test the performance of our approach under conditions that would typically arise with a hand-held camera. At about 12 seconds in the sequence, the object is almost disappearing from the camera's field of view, but still remains visible. In the last three seconds of the sequence the camera "jumps" forth and back by approximately 5 cm. This is because the camera trajectory was extracted from a real VSLAM run. In this case, the VSLAM run produced some wrong estimates in the last frames, thus changing the camera position rapidly. We decided not to smooth this trajectory and instead use this jumping to test the robustness of our approach against unexpected movement of the camera.

Sequence D

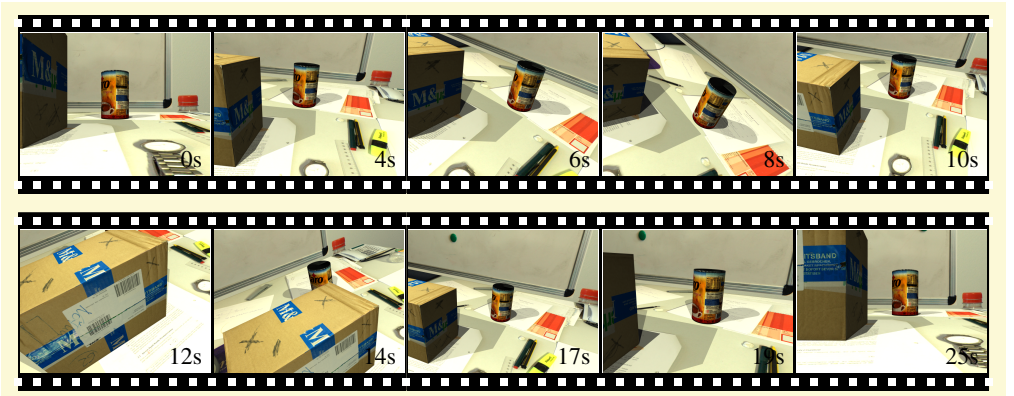


Figure 6.5.: Sample images from sequence D.

This sequence is an extension of sequence C. Instead of just the Caro-tin, the packet object is visible here as well (see figure 6.5). No other changes were made.

This sequence serves to test the tracking of multiple objects and observe their interfering. At about 12 seconds in the sequence, the packet completely occludes the Caro-tin.

Sequence RA

Figure 6.6.: Sample images from sequence RA.

This is the equivalent to sequence A. Again, the camera is not moving (see figure 6.6). The only change between individual frames of this sequence is due to camera noise. The length of this sequence is 5 seconds.

We use this sequence to show that our approach is working steadily on real images as well.

Sequence RB

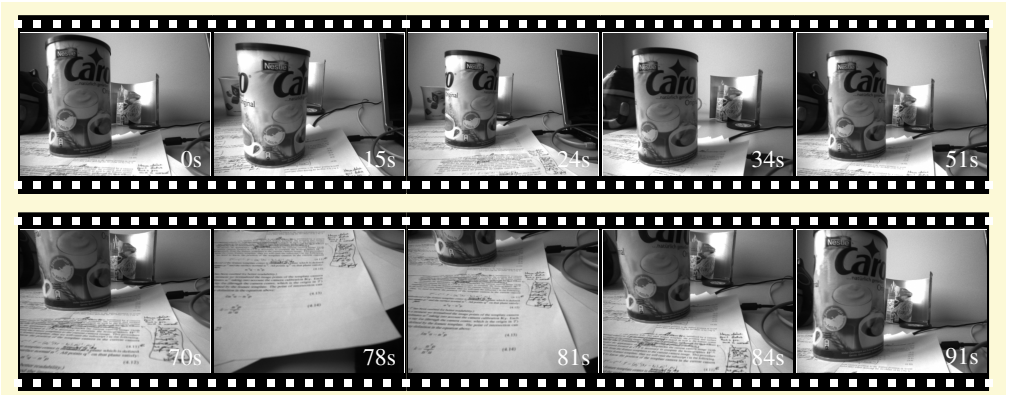


Figure 6.7.: Sample images from sequence RB.

This sequence was used to extract the camera trajectory for sequence B. Thus, the movement of the camera is almost the same (see figure 6.7). The object is placed in a similar distance to the camera as in sequence B. Again, at about 78 seconds in the sequence, the object disappears.

We use this sequence to show the performance of our relocation strategy on real images.

Sequence RC

Figure 6.8.: Sample images from sequence RC.

In this sequence the Caro-tin object is placed on a feature-rich desktop. The camera is slowly approaching the object, exploring it from close distance and moving back again to the starting point (see figure 6.8). The object is visible all the time.

We use this sequence to see when the object will be recognised in the first half and how long it can be tracked after its position is known in the second half.

scenario	OI	PM	RL	OF	BF	GC	RD
I				✓		✓	✓
II				✓		✓	
III				✓			✓
IV					✓	✓	
V	✓	✓		✓	✓	✓	✓
VI	✓			✓	✓	✓	✓
VII	✓	✓			✓	✓	✓
VIII	✓	✓	✓	✓	✓	✓	✓

Table 6.1.: overview of all scenarios used for experiments

6.1.3. Scenarios

In order to test individual components of our implementation on the sequences introduced above, we came up with several *scenarios*. Each scenario represents a different setup of our VSLAM system, e.g., the use of a certain object descriptor or whether the relocation strategy should be active.

An overview of all used scenarios can be found in table 6.1. The abbreviations used are:

OI Indicates whether automatic object initialisation (see section 5.4) should be used. If this is not set for scenarios that track objects, the object pose is initialised manually to the true pose without uncertainty.

PM Indicates whether pose measurements (see section 5.5) should be considered.

RL Indicates whether the camera should be relocated once the system is lost (see section 5.6).

OF If set, point features belonging to object descriptors are getting measured (see section 4.3.4).

BF If set, bundle features are getting initialised and measured.

GC Indicates that the selection of point feature measurement candidates is guided. If this is not set, the candidates are selected purely randomly (see section 4.3.2).

RD If set, a random object descriptor is used. Otherwise, a smooth object descriptor is used (see section 4.2.2).

6.1.4. Evaluation Measures

In the following we will present two evaluation measures for the experiments, namely, the position and rotation error of a pose estimate.

To get the position error, we compare the mean of the current state estimate with the real position given by the ground-truth. The error is just the distance in millimeters between both positions.

For the rotation error we compare the mean of the state estimate and the ground-truth rotation as well. Here, the error is given in degrees: We calculate a rotation $\boldsymbol{\rho}_d$ that, if

concatenated with the estimated rotation $\boldsymbol{\rho}_e$, results in the ground-truth rotation $\boldsymbol{\rho}_g$, i.e., $\boldsymbol{\rho}_e \circ \boldsymbol{\rho}_d = \boldsymbol{\rho}_g$. Now, the rotation $\boldsymbol{\rho}_d$ can be seen as a rotation of θ degrees about an axis $\mathbf{n}$. We consider the value of θ as the error of the rotation estimate.

To visually evaluate the results of the tracking on real and synthetic sequences we provide some sample images of the experiments. These images are gray-scale images of the sequences with the objects painted over at their estimated pose with respect to the camera. Each point feature on the surface of the object is represented by a blue dot and a transparent patch of the feature template. Colored circles around these point features indicate a measurement attempt of the respective point feature. The colored circles outline the search area, which is the projected uncertainty of the object and camera pose. A red cross inside a circle marks the position of the best match.

6.2. Results

The experiments we carried out to test the performance of our approach can be divided into three categories: Plain object tracking (i.e., just point features), object tracking with pose measurements, and camera relocation. Within each category we tried to inspect individual aspects of our implementation.

In the following, we present the results of each category. In the object tracking category, we analysed the plain performance of the object point feature measurements without consideration of pose measurements. Pose measurements are investigated in detail in section 6.2.2. Finally, we will provide the results of our camera relocation strategy in section 6.2.3.

Please note that we decided to put all plots in appendix D, starting at page 94, to enhance the readability of this section.

6.2.1. Object Tracking

To test for the plain object tracking performance we applied scenario I (just object point features) on the sequences A, B, C, and D. Since in this scenario the object pose is given manually and therefore known without uncertainty, we evaluate the performance by means of the camera position and rotation error.

In a second trial we investigated the effect of different object descriptors (smooth and random) and how they behave with different measurement candidate selection strategies. Therefore, we used the scenarios I, II, and III together with the sequences A, B, and C.

Individual Sequences

Sequence A As can be seen in figure D.1, the system does not show a systematic error. The errors visible are due to measurement noise and in particular small rotation errors. Since a wrong estimate of the relative rotation of the object to the camera is correlated to the camera position as well, we observe position errors of up to eighty millimeters, which can be explained by the large distance of the camera to the object. Sample images of this experiment are shown in figure 6.9.

Sequence B For this experiment, we see in figure D.2 that the system behaves steadily with two exceptions:

Figure 6.9.: Sample images of sequence A, scenario I.

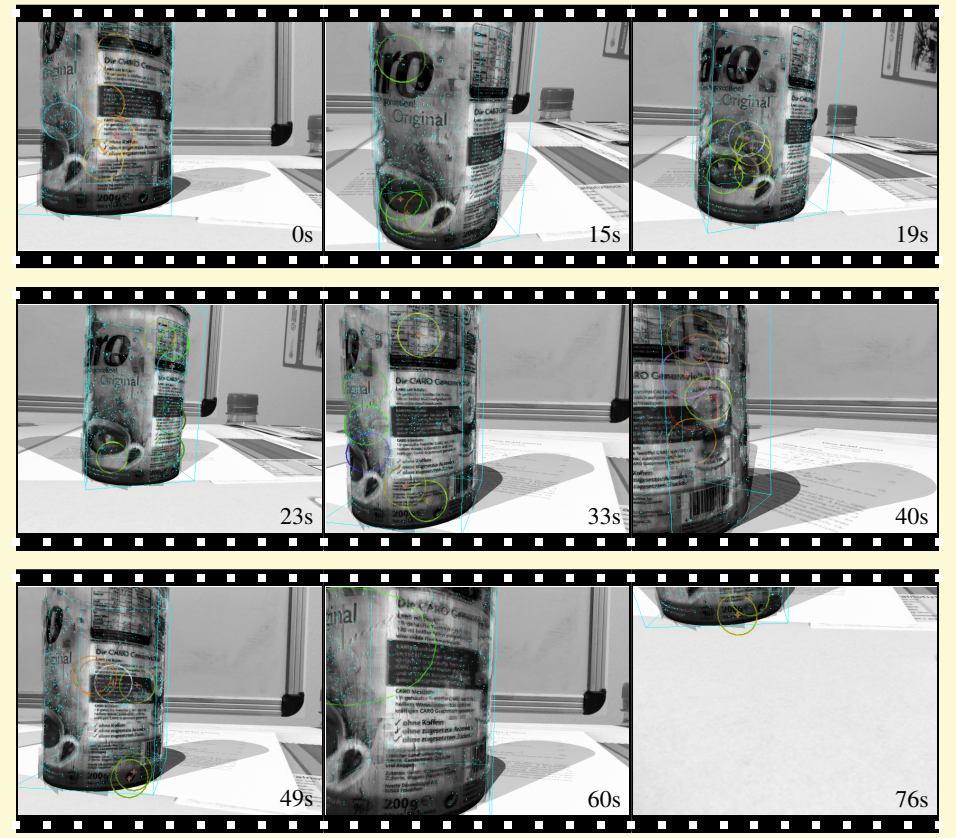


Figure 6.10.: Sample images of sequence B, scenario I.

First, at about 60 seconds in the sequence, the position and rotation error of the camera increases. As mentioned in the description of sequence B already, this is due to the closeup images of the object. Since the object was not observed from such a close distance during learning the system has to rely on the appropriate scaling of the point feature templates. However, as soon as the scale change is too big, the measurements get inaccurate.

The second issue we see is that the system loses track after 76 seconds when the object almost completely disappears from the camera's field of view, as can be seen in the sample images in figure 6.10. Anyway, this is the expected behavior when using a scenario without camera relocation support.

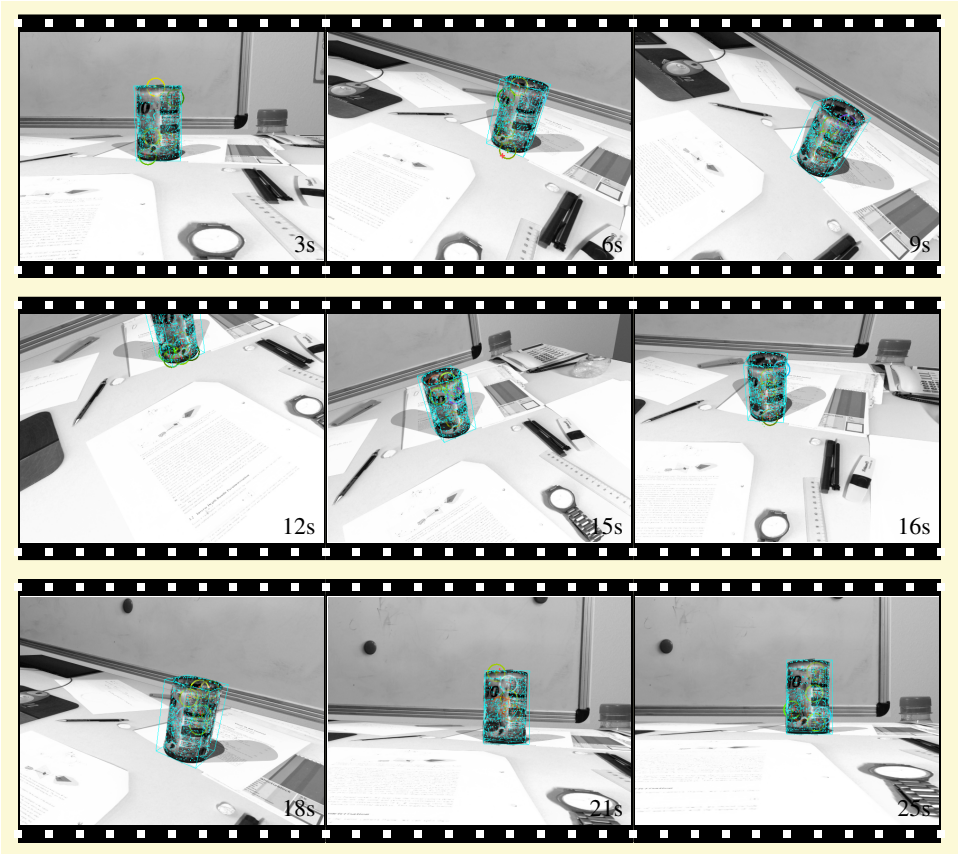


Figure 6.11.: Sample images of sequence C, scenario I.

Sequence C In sequence C we have a similar situation as in sequence A: Since the object is the only feature that is getting tracked and the distance between the camera and the object is quite large, the small rotation errors result in large position errors.

In both plots of figure D.3 we can see two remarkable peaks: One at approximately 13 seconds and another one close to the end of the sequence. The first one belongs to the part of the sequence in which the object is barely visible. The second one is due to the unexpected jumping of the camera in this sequence. In both cases we can see that the system recovers quickly. Sample images of this experiment can be found in figure 6.11.

Sequence D In this sequence, which is the same as sequence C with an additional object, we can see in figure D.4 that the camera position and rotation errors slightly decreased compared to sequence C. Anyway, the system loses track of the objects when the packet occludes the Caro-tin after 14 seconds (see figure 6.12). Occlusion is not handled in our implementation, thus the system tries to measure point features that are not visible and fails.

Measurement Candidate Selection

In the experiments performed above we used scenario I only, i.e., pure object tracking with random object descriptors and the guided candidate selection strategy introduced in section 4.3.2.

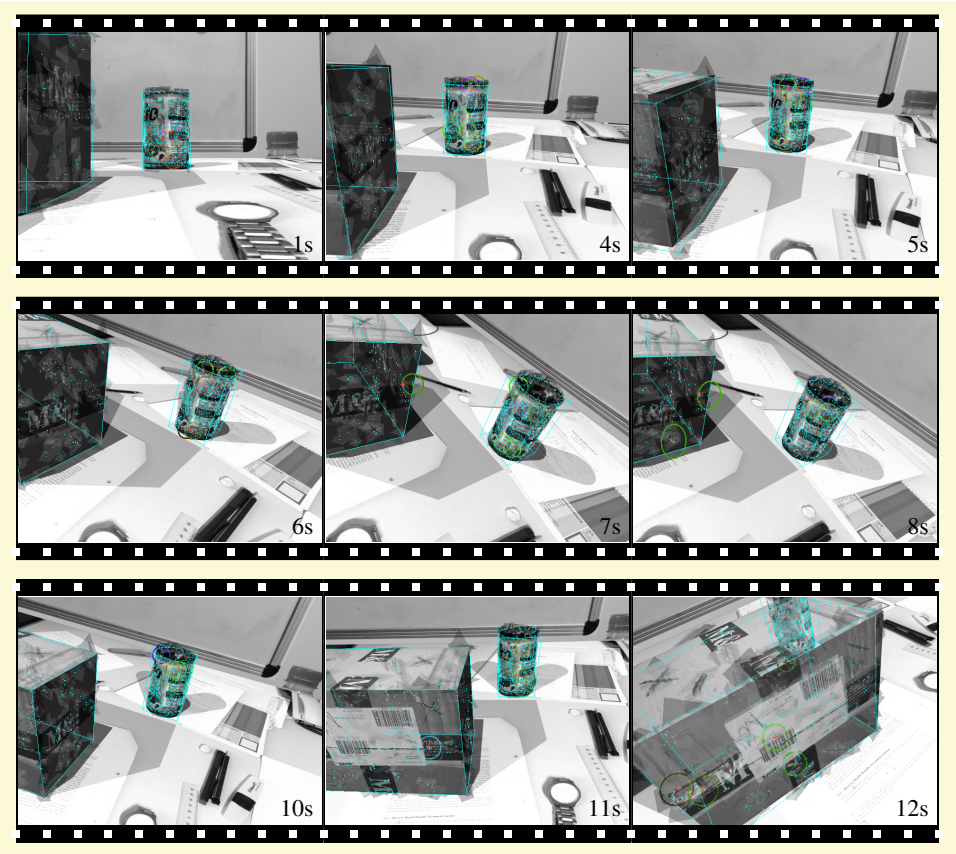


Figure 6.12.: Sample images of sequence D, scenario I.

To analyse the effect of a purely random candidate selection strategy, we used scenario II for comparison. The results on all sequences A, B, and C show that a purely random candidate selection causes larger pose estimation errors when used with a random object descriptor. As a representative example we provide the results of sequence B in figure D.5. We can see that there are several peaks of large errors when using scenario II, resulting in a loss of track after 40 seconds. It is evident from these experiments that the guided candidate selection is superior to the purely random one.

In another trial, we compared the performance of smooth object descriptors against random object descriptors. However, on the sequences A, C, and D tracking did not work at all. Due to the big distance of the camera to the object, none of the point features on the object were selected for measurement. This indicates the weak point of smooth object descriptors: The point features have just been learnt at a specific scale. On sequence B tracking worked in the beginning but failed as soon as the camera was moving too close to the object. Obviously, the random object descriptors are the better choice – even if they contain weak point features that have been filtered out in the smooth object descriptors.

Comparison to Bundle Features

To be able to interpret the results given above in terms of accuracy and state size we ran scenario IV (just bundle features, no object tracking) on the sequences A, B, and C

as well. In these runs, all bundle features have been initialised by the VSLAM system itself.

Except for sequence A, in which the camera is not moving at all, the pose estimates of the camera have been more precise when tracking an object. With sequence B the VSLAM system even lost track after 18 seconds. In figure D.6 we see the camera position error and the state size of sequence C. With bundle features only, the system loses track earlier than with object tracking. Until that point we can clearly see how the system initialises new bundle features and the state size grows. The drop-off of the state size between the initialisations is caused by weak bundle features that are getting removed from the state.

6.2.2. Pose Measurements

The object recognition introduced in chapter 5 serves three purposes: The initialisation of objects, the measurement of poses of objects that are in the state, and the relocation of the camera. The performance of these tasks all depend on the accuracy of the pose estimates from a single camera image.

In this section we provide experimental results that show the accuracy of the object initialisation and the effect that pose measurements have on the state estimate. Relocation will be handled separately in the next section.

Object Initialisation

The object initialisation runs asynchronously to the normal VSLAM operation in a separate thread and can happen anytime. The accuracy of the initial estimate of the object pose depends on the anchor pose estimate that was selected for a key-frame and on the object pose estimate in this key-frame itself.

The accuracy of pose estimates was evaluated on sequence B (containing close-up views) and sequence C (far distance). The results can be seen in figure D.7 and D.8.

It is evident that the accuracy of the pose estimates depends on the distance of the object to the camera. In sequence B we hardly have position estimate errors of more than two millimeters and just negligible rotation errors. However, on sequence C we observe position estimate errors of up to nine centimeters.

Pose Measurements

To analyse the influence of pose measurements on the tracking performance we applied the scenarios V (pose measurements and object features), VI (just object features), and VII (just pose measurements) on sequences A, B, C, and D. In all scenarios, the object initialisation is done automatically and bundle features are used.

Sequence A We could not observe any difference in the camera position and rotation estimate errors. However, the object position and rotation estimates evolved differently for each scenario, as can be seen in figure D.9. We can clearly see that there is a bias in the pose measurement that causes a position error of eight millimeters. This bias can be well explained by the distance of the camera to the object. Since the camera is not moving in this sequence the system has no chance to obtain other pose measurements that would eliminate the error. On the other hand, the object's point feature

measurements provide enough information to decrease the position error to less than one millimeter.

Sequence B Again, the error in the camera pose estimates evolves similarly amongst all tested scenarios. We observe the most interesting results in the object pose errors, as can be seen in figure D.10. Here we see that the run with scenario VII (just pose measurements) lost track after 18 seconds. This indicates how important the object feature measurements are for the robustness of the VSLAM system. Although the difference between scenario V and VI is not that big in this sequence, we can see that the final estimates of the position and rotation of the object are more accurate when pose measurements have not been considered at all.

Sequence C In this sequence the system loses track after 19 seconds if just pose measurements are considered. For scenario V (pose measurements and object features) we observe two periods of high camera pose errors from seven to ten seconds and from 13 to 16 seconds (see figure D.11). These errors are due to inaccurate pose measurements: With scenario VI (just object features) we don't observe big errors in these periods at all. By analysing the object rotation errors in figure D.12 we see again that the estimates are better when ignoring the pose measurements. However, in this sequence the position errors are smaller when pose measurements are considered as well.

Sequence D The results on sequence D confirm our previous observations: With just pose measurements, the system loses track again. As we can see in figures D.13 and D.14 disregarding the pose measurements results in smaller pose estimate errors. The only exception to that is the rotation error for the packet. However, considering the extremely small rotation error, the difference is negligible.

Real Sequences The scenarios V, VI, and VII have also been applied on the real sequences. As a representative example we see sample images from sequence RB in figure 6.13. On the real sequences we were not able to see a difference between scenario V (pose measurements and object features) and VI (just object features). However, as with the synthetic sequences, tracking failed when just pose measurements have been considered.

In sequence RB we see another benefit of using object features: Up to second 17, the object is too small in the camera image to cause a pose measurement. However, once it is initialised it can be tracked by its object features until the very end of the sequence, at which the camera has the same distance to the object as in the beginning.

6.2.3. Camera Relocation

To test the performance of the relocation strategy, we applied scenario VIII to the sequences B, D, and RB. With the scenarios used so far we were not able to keep track on these sequences all the time. In the sequences B and RB the object disappears after 76 seconds and in sequence D the Caro-tin object is occluded by the packet.

Sequence B For the relocation experiment we additionally blacked out some frames in this sequence, such that the system cannot continue tracking: One second at 20 and 30 seconds, and five seconds at 40 seconds. In figure D.15 we see how the camera

Figure 6.13.: Sample images of sequence RC, scenario V.

pose estimation error increases in this periods but recovers quickly as soon as the object becomes visible again. Between 50 and 58 seconds in the sequence the system is considered lost because the camera is too close to the object and a lot of measurement attempts fail. During this period, some of the previous experiments failed. In this run, however, the system recovers. The same happens when the object is not visible at 76 seconds in the sequence. As soon as the object reappears, the system recovers and continues tracking. Sample images of this experiment can be found in figure 6.14.

Sequence D Earlier experiments failed as soon as the packet completely occluded the Caro-tin in this sequence. Again, the relocation support allows the system to recover once the situation becomes unambiguous, as can be seen in the camera pose estimation errors in figure D.16. In these plots we can see another spike at about 24 seconds in the sequence, caused by the already mentioned jumping of the camera near the end of the sequence.

Sequence RB In the sample images of sequence RB in figure 6.15 we can see how the system relocates the camera after the object reappears at 86 seconds.

Figure 6.14.: Sample images of sequence B (some frames blacked out), scenario VIII.

6.3. Discussion

With the experiments performed we could show that object tracking, pose measurements, and the camera relocation work as expected. In comparison to plain bundle feature tracking, we have seen that the error of the camera pose estimate is smaller, even if just a single object is tracked. At the same time, the map size is considerably smaller, although much more point features are available for measurements. Additionally, the map contains more valuable information if objects are tracked. We saw in our relocation experiments how the VSLAM system can make use of that.

We will discuss some observations made in each of the experiment categories below.

6.3.1. Object Tracking

From the experiments on the plain object tracking by point features only, we saw that the combination of random object descriptors with the guided candidate selection strategy was the best.

The reason for this is quite obvious: During the learning of a random object descriptor, new point features get initialised at every single frame. Features on a strong corner may therefore be added to the object descriptor several times and in particular more often than features that are not recognised as interest points in all views. The result is an over-representation of strong interest points. Now during the measurement it is more likely to select several feature descriptors as candidates that all belong to the same

Figure 6.15.: Sample images of sequence RB, scenario VIII.

feature. Thus, the same measurement is incorporated in the update step several times. Whenever this measurement fails (due to noise or some violated assumptions) it fails for each candidate. Methods like JCBB can not detect such an error since there *is* a consensus between these measurements. This way an erroneous measurement finds its way into the update step and may corrupt the whole state estimate, as we were able to see in our experiments.

6.3.2. Pose Measurements

The experiments on the pose measurements indicated that incorporation of pose measurements is not advantageous all the time. The tracking of the object's point features alone gave better results on the estimate of the camera and object pose.

The reason for that can be found in the way SIFT-descriptors work: As we explained in section 5.1, SIFT-descriptors are invariant to rotation and scale, but not to affine transformations in general. However, during a VSLAM run point features of an object are hardly ever observed from the same viewpoint as during learning. This results in incomparable SIFT descriptors and therefore in wrong matches as well as in wrong interest point positions. Point feature templates, on the other hand, do not suffer from affine transformations. Via the template warping explained in section 4.3.3 we are able to account for viewpoint changes as long as we have a good estimate of the surface normal of the feature. Our object descriptors provide these surface normals. Therefore, as long as the pose estimate of the object is accurate enough, point feature templates are

indeed invariant to viewpoint changes. We believe that it is due to this invariance that the point feature templates allow for a better tracking.

6.3.3. Camera Relocation

The weakest point of every SLAM system is the loss of track due to unexpected camera movements or some kind of violated assumptions. With our relocation strategy we saw a way to recover the system from such a state.

As soon as an object was recognized, its estimated pose relative to the camera at the time of observation was used to relocate the camera. However, due to the unknown camera movement between the time of observation and the actual recognition, the location uncertainty is quite large after the relocation. Anyway, the fast measurements of object point features help to decrease the uncertainty within about five frames to a level that we also observe during normal operation. Thus, we can say that a combination of object recognition and point features on objects are a good choice for a robust and fast relocation.

7. Conclusions

With this work we were able to show how object recognition can be integrated in VSLAM. In contrast to previous works, our implementation is able to handle non-planar objects and distinguishes between object tracking and object recognition. As we could show on real and synthetic sequences, the object tracking ensured a more accurate estimation of the object poses in contrast to just object recognition. The object recognition was used to initialise objects, measure their pose asynchronously via delayed measurements, and to relocate the camera once the system lost track.

Furthermore, we could see in our experiments that the objects serve as rich features for the VSLAM system. Since they are represented by their pose in the state this helped to reduce the map size, compared to the amount of point features a VSLAM system would need to locate the camera equally accurate. Besides that, objects provide more information to the user than point features. This is of special interest to further applications that could use the map estimate to interact with objects.

Another benefit of the object recognition in VSLAM is that it does not rely on a good state estimate to produce measurements. This allowed for camera relocation, as we could show in experiments as well.

However, we think there are several ways by which our work could be improved. First of all, the precision of the object recognition may be increased by using viewpoint invariant feature descriptors instead of SIFT.

Another enhancement could be achieved if objects wouldn't be considered as non-moving. In fact, this would require just a little change to the system: Besides their pose, objects would have a velocity as well, which would be measured via the IEKF framework likewise. On the other hand, moving objects wouldn't allow for camera relocation as we proposed it.

The real bottleneck of our approach is certainly the need for a 3D object model to learn the object descriptor. It would be desirable to replace the modelling by an automatic object reconstruction from sample images of the object only. A promising technique involving volumetric graph-cuts for that purpose has recently been published by Campbell et al. [4].

A. Vector Calculus

In the classical differential calculus the derivation of a function $y = f(x)$ with $f : \mathbb{R} \mapsto \mathbb{R}$ at position x is defined as:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{A.1}$$

If this limit exists, $f(x)$ is said to be *differentiable* at x.

A.1. Partial and Total Derivatives

The concept of the derivative of a scalar function can be generalised to a function of several arguments. Consider a function $y = f(\mathbf{x})$ with $f : \mathbb{R}^n \mapsto \mathbb{R}$. By selecting one component $\mathbf{x}_i$ of $\mathbf{x}$ we can get the *partial derivative* of y by $\mathbf{x}_i$ if we fix all other components $\mathbf{x}_{j \neq i}$:

$$\frac{\partial y}{\partial \mathbf{x}_i} = \lim_{\Delta \mathbf{x}_i \to 0} \frac{\Delta y}{\Delta \mathbf{x}_i} \tag{A.2}$$

$$= \lim_{\Delta \mathbf{x}_i \to 0} \frac{f(\mathbf{x}_1, \cdots, \mathbf{x}_i + \Delta \mathbf{x}_i, \cdots, \mathbf{x}_n) - f(\mathbf{x}_1, \cdots, \mathbf{x}_i, \cdots, \mathbf{x}_n)}{\Delta \mathbf{x}_i} \tag{A.3}$$

The partial derivative of y by the whole vector $\mathbf{x}$ is given as a vector containing the partial derivatives per component:

$$\frac{\partial y}{\partial \mathbf{x}^\mathsf{T}} = \begin{pmatrix} \frac{\partial y}{\partial \mathbf{x}_i} & \cdots & \frac{\partial y}{\partial \mathbf{x}_n} \end{pmatrix} \tag{A.4}$$

Note that we use ∂ to denote partial derivatives in contrast to d to denote *total derivatives*. The difference between both derivatives is that in the case of the partial derivative we disregard functional dependencies between the components of $\mathbf{x}$. We did this by considering every component but the one we are interested in as a constant. The total derivative, however, allows the components to depend on each other:

$$\frac{dy}{d\mathbf{x}_i} = \frac{\partial y}{\partial \mathbf{x}_i} + \sum_{j \neq i} \frac{\partial y}{\partial \mathbf{x}_j} \frac{d\mathbf{x}_j}{d\mathbf{x}_i} \tag{A.5}$$

It can easily be seen from this equation that the partial and total derivatives are equal if there are no functional dependencies between the components. This is particularly true if the function depends only on one scalar value. In this case we will favor the notation of the total derivative (as we already did in equation A.1).

Similarly to the univariable derivative we say that a function $f(\mathbf{x})$ is differentiable at $\mathbf{x}$ if the limit of equation A.3 exists for each component $\mathbf{x}_i$ of $\mathbf{x}$. Equivalently, we can say:

Definition A.1: (Differentiability) A function $y = f(\mathbf{x})$ with $f : \mathbb{R}^n \mapsto \mathbb{R}$ is said to be *differentiable* at the point $\mathbf{x}$ if:

$$\Delta y = \sum_{i=1}^{n} \frac{\partial y}{\partial \mathbf{x}_i} \Delta \mathbf{x}_i + \sum_{i=1}^{n} \epsilon_i \Delta \mathbf{x}_i \tag{A.6}$$

where all $\epsilon_i \to 0$ as $\Delta \mathbf{x} \to \mathbf{0}$.

Informally speaking, this definition requires the partial derivatives to be an approximation of the change of y with respect to a change of $\mathbf{x}$. The ϵ_i serve as a kind of error functions explaining the deviation of the approximation to the real change of y. This means that the ϵ_i are implicitly functions of $\Delta \mathbf{x}_i$. The crucial criteria for the partial derivations is now that the error functions need to converge to zero as $\Delta \mathbf{x}$ goes to the zero vector. This implies that the partial derivatives correctly explain the change in y in an infinitesimal small vicinity around of $\mathbf{x}$.

A.2. Vector Derivatives

Until now we just considered scalar functions with an arbitrary number of arguments. The concept of derivatives, however, can be generalised to vector functions as well. Suppose we have a function $\mathbf{y} = f(\mathbf{x})$ with $f : \mathbb{R}^n \mapsto \mathbb{R}^m$. The partial derivative of this function with respect to $\mathbf{x}$ is given as a matrix comprising the partial derivatives of each component of $\mathbf{y}$ by each component of $\mathbf{x}$:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}^\mathsf{T}} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{y}_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{y}_m}{\partial \mathbf{x}_n} \end{bmatrix} \tag{A.7}$$

The resulting matrix is called the *Jacobian* of $\mathbf{y}$ with respect to $\mathbf{x}$. The components are arranged such that the ith row corresponds to the ith component of $\mathbf{y}$ and the jth column corresponds to the jth component of $\mathbf{x}$. Please note that the transposition of the derivation variable is just a matter of taste (and can be found differently in the literature): We use it here to denote that the resulting matrix will have m rows and n columns.

A.3. Chain Rule

In this section we will generalise the concept of the univariable chain rule (which we assume to be known to the reader) to multivariable cases. To ease the generalisation we will perform it in several steps, each lifting one of the scalar variables involved in the previous step to a vector of arbitrary size. This will give as four versions of the chain rule to which we will refer as *first*, *second*, *third* and *fourth order chain rule*.

Consider a chain of functions $\mathbf{y} = f(\mathbf{z}) = f(g(\mathbf{x}))$ with $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $g : \mathbb{R}^p \mapsto \mathbb{R}^n$. In the case in which $m = n = p = 1$ the standard chain rule in differential calculus (see [2] or [36]) allows us to give the derivative of y by x as the following product:

$$\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx} \tag{A.8}$$

Please note that we use x, y and z instead of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ to account for the fact that these values are scalars. We will refer to this chain rule as the first order chain rule. For

the second order chain rule we will allow $\mathbf{z}$ to be of arbitrary size. For the third order chain rule we additionally allow $\mathbf{x}$ to be of arbitrary size and the fourth order chain rule allows all parameters to be of arbitrary size.

To derive the second order chain rule, consider a chain of functions $y = f(\mathbf{z}) = f(g(x))$ with $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R} \mapsto \mathbb{R}^n$.

Theorem 1.1: (Second order chain rule) The second order chain rule is given as:

$$\frac{dy}{dx} = \frac{\partial y}{\partial \mathbf{y}^\mathsf{T}} \frac{d\mathbf{y}}{dx} \tag{A.9}$$

Proof. The sought derivative of y by x is, according to equation A.1:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} \tag{A.10}$$

Assuming that $f(.)$ is differentiable we can rewrite Δy by means of equation A.6 and apply the limes operation to each summand:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \left(\sum_{i=1}^{n} \frac{\partial y}{\partial \mathbf{z}_i} \frac{\Delta \mathbf{z}_i}{\Delta x} + \sum_{i=1}^{n} \epsilon_i \frac{\Delta \mathbf{z}_i}{\Delta x} \right) \tag{A.11}$$

$$= \sum_{i=1}^{n} \frac{\partial y}{\partial \mathbf{z}_i} \frac{d\mathbf{z}_i}{dx} + \sum_{i=1}^{n} \left(\lim_{\Delta x \to 0} \epsilon_i \right) \frac{d\mathbf{z}_i}{dx} \tag{A.12}$$

It remains to show that the second sum vanishes for $\Delta x \to 0$. As stated above, the ϵ_i are implicitly functions of $\Delta \mathbf{z}_i$. By definition we require the ϵ_i to vanish for $\Delta \mathbf{z}_i \to 0$. We see that for $\Delta x \to 0$:

$$\lim_{\Delta x \to 0} \Delta \mathbf{z}_i = \lim_{\Delta x \to 0} \frac{\Delta \mathbf{z}_i}{\Delta x} \Delta x \tag{A.13}$$

$$= \frac{d\mathbf{z}_i}{dx} \lim_{\Delta x \to 0} \Delta x \tag{A.14}$$

$$= 0 \tag{A.15}$$

It follows that the ϵ_i are zero for $\Delta x \to 0$. The remaining sum is just the component representation of the dot-product, i.e., the product of a row- with a column-vector. □

Now consider a chain of functions $y = f(\mathbf{z}) = f(g(\mathbf{x}))$ with $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R}^p \mapsto \mathbb{R}^n$. We can see that y depends on the vector $\mathbf{x}$ now.

Theorem 1.2: (Third order chain rule) The third order chain rule is given as:

$$\frac{\partial y}{\partial \mathbf{x}^\mathsf{T}} = \frac{\partial y}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \tag{A.16}$$

Proof. The sought partial derivative of y by $\mathbf{x}$ can now be obtained by building the partial derivatives of y by each component $\mathbf{x}_i$ (and therefore considering all other components $\mathbf{x}_{j \neq i}$ to be fixed). The result is a vector containing all partial derivatives which

we can obtain by application of the second order chain rule. Rewriting this vector as the product of a vector and a matrix leads us to:

$$\frac{\partial y}{\partial \mathbf{x}^\mathsf{T}} = \left(\frac{\partial y}{\partial \mathbf{x}_1}, \cdots, \frac{\partial y}{\partial \mathbf{x}_m} \right) \tag{A.17}$$

$$= \left(\frac{\partial y}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}_1}, \cdots, \frac{\partial y}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}_m} \right) \tag{A.18}$$

$$= \frac{\partial y}{\partial \mathbf{z}^\mathsf{T}} \begin{bmatrix} \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{z}_n}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{z}_n}{\partial \mathbf{x}_m} \end{bmatrix} \tag{A.19}$$

$$= \frac{\partial y}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \tag{A.20}$$

□

Finally, we consider a chain of functions $\mathbf{y} = f(\mathbf{z}) = f(g(\mathbf{x}))$ with $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $g : \mathbb{R}^p \mapsto \mathbb{R}^n$.

Theorem 1.3: (Fourth order chain rule) The fourth order chain rule is given as:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}^\mathsf{T}} = \frac{\partial \mathbf{y}}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \tag{A.21}$$

Proof. Since both $\mathbf{x}$ and $\mathbf{y}$ are vectors of arbitrary size now, the partial derivative of $\mathbf{y}$ by $\mathbf{x}$ is defined as a Jacobian matrix (see equation A.7) with the column vectors given for each component of $\mathbf{y}$ by the third order chain rule. By rewriting we see that we can decompose this Jacobian in two Jacobians:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}^\mathsf{T}} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}^\mathsf{T}} \\ \vdots \\ \frac{\partial \mathbf{y}_p}{\partial \mathbf{x}^\mathsf{T}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \\ \vdots \\ \frac{\partial \mathbf{y}_p}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{z}^\mathsf{T}} \\ \vdots \\ \frac{\partial \mathbf{y}_p}{\partial \mathbf{z}^\mathsf{T}} \end{bmatrix} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \tag{A.22}$$

$$= \frac{\partial \mathbf{y}}{\partial \mathbf{z}^\mathsf{T}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}^\mathsf{T}} \tag{A.23}$$

□

A.4. Product Rule

Consider a scalar function $s = f(\mathbf{x})$ and a vector function $\mathbf{v} = g(\mathbf{x})$ with $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R}^n \mapsto \mathbb{R}^m$.

Theorem 1.1: (Scalar-Vector Product Rule) The partial derivative of the product $\mathbf{y}$ of a scalar s and a vector $\mathbf{v}$ by $\mathbf{x}$ is given as:

$$\frac{\partial s\mathbf{v}}{\partial \mathbf{x}^\mathsf{T}} = s\frac{\partial \mathbf{v}}{\partial \mathbf{x}^\mathsf{T}} + \mathbf{v}\frac{\partial s}{\partial \mathbf{x}^\mathsf{T}} \tag{A.24}$$

Proof. By considering the result of the product $s\mathbf{v}$ per component, the proof of the theorem stated above is straight forward by application of the scalar-scalar product rule:

$$\frac{\partial s\mathbf{v}}{\partial \mathbf{x}^\mathsf{T}} = \frac{\partial}{\partial \mathbf{x}^\mathsf{T}} \begin{pmatrix} s\mathbf{v}_1 \\ \vdots \\ s\mathbf{v}_m \end{pmatrix} \tag{A.25}$$

$$= \begin{bmatrix} \frac{\partial s\mathbf{v}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial s\mathbf{v}_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial s\mathbf{v}_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial s\mathbf{v}_m}{\partial \mathbf{x}_n} \end{bmatrix} = \begin{bmatrix} s\frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_1} + \mathbf{v}_1\frac{\partial s}{\partial \mathbf{x}_1} & \cdots & s\frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_n} + \mathbf{v}_1\frac{\partial s}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ s\frac{\partial \mathbf{v}_m}{\partial \mathbf{x}_1} + \mathbf{v}_m\frac{\partial s}{\partial \mathbf{x}_1} & \cdots & s\frac{\partial \mathbf{v}_m}{\partial \mathbf{x}_n} + \mathbf{v}_m\frac{\partial s}{\partial \mathbf{x}_n} \end{bmatrix} \tag{A.26}$$

$$= \begin{bmatrix} s\frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_1} & \cdots & s\frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ s\frac{\partial \mathbf{v}_m}{\partial \mathbf{x}_1} & \cdots & s\frac{\partial \mathbf{v}_m}{\partial \mathbf{x}_n} \end{bmatrix} + \begin{bmatrix} \mathbf{v}_1\frac{\partial s}{\partial \mathbf{x}_1} & \cdots & \mathbf{v}_1\frac{\partial s}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \mathbf{v}_m\frac{\partial s}{\partial \mathbf{x}_1} & \cdots & \mathbf{v}_m\frac{\partial s}{\partial \mathbf{x}_n} \end{bmatrix} \tag{A.27}$$

$$= s\frac{\partial \mathbf{v}}{\partial \mathbf{x}^\mathsf{T}} + \mathbf{v}\frac{\partial s}{\partial \mathbf{x}^\mathsf{T}} \tag{A.28}$$

□

B. Probability Theory

This section gives a brief introduction in probability theory and clarifies the notation used throughout this thesis.

The most basic concept in probability theory is that of *random variables*. A random variable can be assigned to multiple possible values - each with a certain probability. Random variables represent quantities which underly uncertainty, such as measurements, forecasts, or vague statements.

A random variable is denoted by a capital letter, e.g. X. A certain value of a random variable is denoted by the appropriate lower case letter, e.g. x. The set of all possible values for a random variable X is called its *sample space* and is denoted by Ω_X. This notation leads to the concept of a *random event*, denoted as $X = x$, meaning the event in which the random variable X takes a particular value $x \in \Omega_X$.

If the sample set for a random variable is discrete a probability can be assigned to each random event, written as $p(X = x)$. However, this probabilities can not be arbitrary: They have to be positive and need to sum up to one over the whole sample space.

Definition B.1: (Probability Function) A function $p : \Omega_X \mapsto \mathbb{R}$ is a probability function over a random variable X if:

$$p(X = x) \geq 0; \quad \forall x \tag{B.1}$$

$$\sum_x p(X = x) = 1 \tag{B.2}$$

where Ω_X is a countable set.

Throughout this thesis we will deal with continuous random variables, i.e. random variables with a continuous, real valued sample space $\Omega \subseteq \mathbb{R}^m$. In this case, instead of a probability function, we provide a *probability density function* (or short *pdf*) for each random event. The requirements for such a pdf are analogous to that for probability functions:

Definition B.2: (Probability Density Function) A function $p : \Omega_X \mapsto \mathbb{R}$ is a probability density function over a random variable X if:

$$p(X = \mathbf{x}) \geq 0; \quad \forall \mathbf{x} \tag{B.3}$$

$$\int_{\mathbf{x}} p(X = \mathbf{x}) d\mathbf{x} = 1 \tag{B.4}$$

where $\Omega_X \subseteq \mathbb{R}^m$.

Although we narrow our focus to pdfs now, the following explanations are also true for discrete probability functions. For convenience we will omit the explicit mentioning of the random variable and just write $p(\mathbf{x})$ instead of $p(X = \mathbf{x})$.

In the case of several random variables one may wish to express the probability of a so called joint event, that is, the probability of the occurrence of several events. A joint pdf is written as $p(\mathbf{x}^1, \ldots, \mathbf{x}^n)$ with each $\mathbf{x}^i$ being a value of a random variable X^i. If the joint pdf can be constructed as the product of individual pdfs, the random variables X^i are said to be *independent*:

Definition B.3: (Independence) Several random variables $X^1, \ldots, X^n$ are said to be independent of each other if their joint pdf can be decomposed as:

$$p(\mathbf{x}^1, \ldots, \mathbf{x}^n) = \prod_i p(\mathbf{x}^i) \tag{B.5}$$

If on the other hand random variables are dependent on each other, a given assignment of one random variable Y changes the pdf of another, dependent random variable X. The resulting pdf for X is called a conditional pdf on Y and is denoted by a vertical bar. It is completely defined by the joint pdf and the pdf of Y:

Definition B.4: (Conditional Probability Density Function) Given two random variables X and Y the pdf of X given an assignment of $Y = \mathbf{y}$ is:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{y})} \tag{B.6}$$

Note that in the case of independence between X and Y, $p(\mathbf{x}|\mathbf{y})$ is equal to $p(\mathbf{x})$ (following from equation B.5).

Regardless of the dependency relation between random variables, it is at least theoretical possible to recover a pdf of a certain random variable given a joint pdf involving this random variable:

Definition B.5: (Marginalisation Rule) The marginal pdf of a random variable X is defined as the integral of the joint pdf of X and another random variable Y over all possible values of Y:

$$p(\mathbf{x}) = \int_{\mathbf{y}} p(\mathbf{x}, \mathbf{y}) d\mathbf{y} \tag{B.7}$$

By putting together this equation and equation B.6 we obtain what became famous as the *Bayes rule*:

Definition B.6: (Bayes Rule) Given $p(\mathbf{y}|\mathbf{x})$ and $p(\mathbf{x})$ it is possible to obtain the conditional pdf $p(\mathbf{x}|\mathbf{y})$ as:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})} = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{\int'_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}')p(\mathbf{x}')d\mathbf{x}'} \tag{B.8}$$

C. Jacobians

C.1. Projection

Let a camera configuration be given as an intrinsic camera matrix $\mathbf{K}$ with f being the focal length and Δu and Δv being the coordinates of the principal point:

$$\mathbf{K} = \begin{bmatrix} f & 0 & \Delta u \\ 0 & f & \Delta v \\ 0 & 0 & 1 \end{bmatrix} \tag{C.1}$$

Accordingly, the projection $\mathbf{q} = (u, v)^\mathsf{T}$ of a point $\mathbf{p} = (x, y, z)^\mathsf{T}$ is given as:

$$\mathbf{q} = \pi_\mathbf{K}(\mathbf{p}) = \pi(\mathbf{K}\mathbf{p}) = \begin{pmatrix} \Delta u + fx/z \\ \Delta v + fy/z \end{pmatrix}; \quad z \neq 0 \tag{C.2}$$

Now we consider the case of a stereo camera setup, in which the left camera pose is retrieved by translating the right camera about b units on the x-axis of a camera centered coordinate system. In this case, the stereo projection $\mathbf{q}_S = (u, v, d)^\mathsf{T}$ of a point $\mathbf{p}$ can be described as its projection to the right camera image and the disparity, i.e. the difference in the u-coordinate, to the left camera image:

$$\mathbf{q}_S = \pi_S(\mathbf{p}) = \begin{pmatrix} \Delta u + fx/z \\ \Delta v + fy/z \\ fb/z \end{pmatrix}; \quad z \neq 0 \tag{C.3}$$

Therefore, the Jacobian of a stereo projection by a point $\mathbf{p}$ is given as:

$$\frac{\partial \pi_S(\mathbf{p})}{\partial \mathbf{p}} = \begin{bmatrix} f/z & 0 & -fx/z^2 \\ 0 & f/z & -fy/z^2 \\ 0 & 0 & -fb/z^2 \end{bmatrix} \tag{C.4}$$

C.2. Rotations

A rotation in exponential coordinates is given as a vector:

$$\boldsymbol{\rho} = (\boldsymbol{\rho}_x, \boldsymbol{\rho}_y, \boldsymbol{\rho}_y)^\mathsf{T} \tag{C.5}$$

The application of a rotation $\boldsymbol{\rho}$ in exponential coordinates to a point $\mathbf{p}$ is basically a rotation of this point about an axis $\mathbf{n}$ by θ degrees. The rotation axis is given by the direction of $\boldsymbol{\rho}$ directly. The number of degrees is the length of $\boldsymbol{\rho}$. Hence, the axis and angle representation can be derived as follows:

$$\mathbf{n} = \begin{cases} \frac{\boldsymbol{\rho}}{|\boldsymbol{\rho}|} & \text{if } |\boldsymbol{\rho}| > 0 \\ (1, 0, 0)^{-1} & \text{else} \end{cases} \tag{C.6}$$

$$\theta = |\boldsymbol{\rho}| \tag{C.7}$$

The result of rotating a point $\mathbf{p}$ about an axis $\mathbf{n}$ by θ degrees is:

$$\mathbf{q} = \cos\theta\mathbf{p} + (\mathbf{n} \cdot \mathbf{p})(1 - \cos\theta)\mathbf{n} + \sin\theta(\mathbf{p} \times \mathbf{n}) \tag{C.8}$$

C.2.1. Rotation of a Point by Rotation

The rotation of a point $\mathbf{p}$ by a rotation in exponential coordinates $\boldsymbol{\rho}$ can be carried out in several ways, each providing the same result. The straight-forward approach would be to convert the exponential coordinate representation to an axis-angle representation and use that to perform the rotation of the point $\mathbf{p}$ as described in section 2.2, equations 2.48 and 2.47 (reprinted here for convenience):

$$\mathbf{n} = \begin{cases} \frac{\boldsymbol{\rho}}{|\boldsymbol{\rho}|} & \text{if} |\boldsymbol{\rho}| > 0 \\ (1,0,0)^{-1} & \text{else} \end{cases} \tag{C.9}$$

$$\theta = |\boldsymbol{\rho}| \tag{C.10}$$

$$\mathbf{p}' = \cos\theta\mathbf{p} + (\mathbf{n}\cdot\mathbf{p})(1-\cos\theta)\mathbf{n} + \sin\theta(\mathbf{p}\times\mathbf{n}) \tag{C.11}$$

However, for the sake of deriving the Jacobians of this rotation function there are some reasons to follow another approach, namely, converting the rotation to a quaternion $\boldsymbol{q}$ first and applying the quaternion rotation to the point $\mathbf{p}$. These reasons are:

- As we will see the derivation of the Jacobians can nicely be separated in two matrices: one for the conversion to the quaternion and one for the application of the rotation. The intermediate steps do just involve the application of the product rule on scalars and vectors.
- The fact that the magnitude of the exponential coordinate rotation can get close to zero needs to be handled with care. Using the two-step approach we can take care of this issue where it occurs, that is, during the conversion to the quaternion.
- The Jacobians in our implementation are actually computed in this way.

A quaternion $\boldsymbol{q}$ representing the rotation of an exponential coordinate rotation $\boldsymbol{\rho}$ is defined as:

$$\boldsymbol{q} = \text{quat}(\boldsymbol{\rho}) \stackrel{\text{def}}{=} \begin{pmatrix} \cos\frac{|\boldsymbol{\rho}|}{2} \\ \frac{\sin\frac{|\boldsymbol{\rho}|}{2}}{|\boldsymbol{\rho}|}\boldsymbol{\rho} \end{pmatrix} \tag{C.12}$$

The according rotation matrix of a quaternion $\boldsymbol{q}$ is defined as:

$$\mathbf{R}_\mathbf{q} = \begin{bmatrix} 1-2\boldsymbol{q}_{yy}-2\boldsymbol{q}_{zz} & 2\boldsymbol{q}_{xy}-2\boldsymbol{q}_{\omega z} & 2\boldsymbol{q}_{\omega y}+2\boldsymbol{q}_{xz} \\ 2\boldsymbol{q}_{\omega z}+2\boldsymbol{q}_{xy} & 1-2\boldsymbol{q}_{xx}-2\boldsymbol{q}_{zz} & 2\boldsymbol{q}_{yz}-2\boldsymbol{q}_{\omega x} \\ 2\boldsymbol{q}_{xz}-2\boldsymbol{q}_{\omega y} & 2\boldsymbol{q}_{\omega x}+2\boldsymbol{q}_{yz} & 1-2\boldsymbol{q}_{xx}-2\boldsymbol{q}_{yy}) \end{bmatrix} \tag{C.13}$$

Where $\boldsymbol{q}_{\alpha\beta}$ is a shorthand notation for $\boldsymbol{q}_\alpha\boldsymbol{q}_\beta$ (with $\alpha,\beta \in \{\omega,x,y,z\}$).

The rotation of a point $\mathbf{p}$ by a quaternion $\boldsymbol{q}$ is the result of multiplying the rotation matrix $\mathbf{R}_{\boldsymbol{q}}$ with $\mathbf{p}$:

$$\mathbf{p}' = \mathbf{R}_{\boldsymbol{q}} \times \mathbf{p} \tag{C.14}$$

$$= 2\begin{pmatrix} (-\boldsymbol{q}_{yy}-\boldsymbol{q}_{zz})\mathbf{p}_x + (\boldsymbol{q}_{xy}-\boldsymbol{q}_{\omega z})\mathbf{p}_y + (\boldsymbol{q}_{\omega y}+\boldsymbol{q}_{xz})\mathbf{p}_z \\ (\boldsymbol{q}_{\omega z}+\boldsymbol{q}_{xy})\mathbf{p}_x + (-\boldsymbol{q}_{xx}-\boldsymbol{q}_{zz})\mathbf{p}_y + (\boldsymbol{q}_{yz}-\boldsymbol{q}_{\omega x})\mathbf{p}_z \\ (\boldsymbol{q}_{xz}-\boldsymbol{q}_{\omega y})\mathbf{p}_x + (\boldsymbol{q}_{\omega x}+\boldsymbol{q}_{yz})\mathbf{p}_y + (-\boldsymbol{q}_{xx}-\boldsymbol{q}_{yy})\mathbf{p}_z \end{pmatrix} + \mathbf{p} \tag{C.15}$$

Given these definitions it is possible to rewrite the Jacobian of the rotation of a point $\mathbf{p}$ about an axis and angle in exponential coordinates $\boldsymbol{\rho}$ by the rotation as a product of two Jacobians:

$$\frac{\partial\mathbf{p}'}{\partial\boldsymbol{\rho}^\mathsf{T}} = \frac{\partial\mathbf{p}'}{\partial\boldsymbol{q}^\mathsf{T}}\frac{\partial\boldsymbol{q}}{\partial\boldsymbol{\rho}^\mathsf{T}} \tag{C.16}$$

The first Jacobian is the derivative of a quaternion rotation of a point by the quaternion. The second Jacobian is the derivative of a quaternion rotation by a rotation in exponential coordinates.

The Jacobian of the rotated point $\mathbf{p}'$ by the quaternion can easily be derived from equation C.15:

$$\frac{\partial \mathbf{p}'}{\partial \boldsymbol{q}^\mathsf{T}} = \begin{bmatrix} -\boldsymbol{q}_z\mathbf{p}_y+\boldsymbol{q}_y\mathbf{p}_z & \boldsymbol{q}_y\mathbf{p}_y+\boldsymbol{q}_z\mathbf{p}_z & -2\boldsymbol{q}_y\mathbf{p}_x+\boldsymbol{q}_x\mathbf{p}_y+\boldsymbol{q}_\omega\mathbf{p}_z & -2\boldsymbol{q}_z\mathbf{p}_x-\boldsymbol{q}_\omega\mathbf{p}_y+\boldsymbol{q}_x\mathbf{p}_z \\ \boldsymbol{q}_z\mathbf{p}_x-\boldsymbol{q}_x\mathbf{p}_z & \boldsymbol{q}_y\mathbf{p}_x-2\boldsymbol{q}_x\mathbf{p}_y-\boldsymbol{q}_\omega\mathbf{p}_z & \boldsymbol{q}_x\mathbf{p}_x+\boldsymbol{q}_z\mathbf{p}_z & \boldsymbol{q}_\omega\mathbf{p}_x-2\boldsymbol{q}_z\mathbf{p}_y+\boldsymbol{q}_y\mathbf{p}_z \\ -\boldsymbol{q}_y\mathbf{p}_x+\boldsymbol{q}_x\mathbf{p}_y & \boldsymbol{q}_z\mathbf{p}_x+\boldsymbol{q}_\omega\mathbf{p}_y-2\boldsymbol{q}_x\mathbf{p}_z & -\boldsymbol{q}_\omega\mathbf{p}_x+\boldsymbol{q}_z\mathbf{p}_y-2\boldsymbol{q}_y\mathbf{p}_z & \boldsymbol{q}_x\mathbf{p}_x+\boldsymbol{q}_y\mathbf{p}_y \end{bmatrix} \tag{C.17}$$

The Jacobian of the quaternion rotation by the exponential coordinate roation involves some more steps. According to the definition of quaternions the Jacobian of a quaternion rotation $\boldsymbol{q}$ by a rotation in exponential coordinates $\boldsymbol{\rho}$ can be be split in two derivations: one for the ω-component of the quaternion, and one for the (x, y, z)-components.

$$\frac{\partial}{\partial \boldsymbol{\rho}^\mathsf{T}} \cos\frac{|\boldsymbol{\rho}|}{2} = -\frac{1}{2}\underbrace{\frac{\sin\frac{|\boldsymbol{\rho}|}{2}}{|\boldsymbol{\rho}|}}_{a}\boldsymbol{\rho}^\mathsf{T} \tag{C.18}$$

Here we see that the computation of the derivative of the ω-component of the quaternion by the exponential coordinate rotation depends on a scalar value which we will denote by a in the following. We will see in the next equations that the derivative of the (x, y, z)-components of the quaternion also involves a and another scalar value which we will denote by b. Note that we substitute $\sin\frac{|\boldsymbol{\rho}|}{2}$ with s and $\cos\frac{|\boldsymbol{\rho}|}{2}$ with c in the subsequent equations for better readability:

(C.19)

$$\frac{\partial}{\partial \boldsymbol{\rho}^\mathsf{T}} s\frac{\boldsymbol{\rho}}{|\boldsymbol{\rho}|} = \boldsymbol{\rho} \times \frac{\partial}{\partial \boldsymbol{\rho}^\mathsf{T}}\frac{s}{|\boldsymbol{\rho}|} + \frac{s}{|\boldsymbol{\rho}|}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{\rho}^\mathsf{T}} \qquad \text{product rule} \tag{C.20}$$

$$= \boldsymbol{\rho} \times \left(\frac{\partial s}{\partial \boldsymbol{\rho}^\mathsf{T}}\frac{1}{|\boldsymbol{\rho}|} + s\frac{\partial \frac{1}{|\boldsymbol{\rho}|}}{\partial \boldsymbol{\rho}^\mathsf{T}}\right) + \frac{s}{|\boldsymbol{\rho}|}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{\rho}^\mathsf{T}} \qquad \text{product rule} \tag{C.21}$$

$$= \boldsymbol{\rho} \times \left(\frac{c}{2|\boldsymbol{\rho}|^2}\boldsymbol{\rho}^\mathsf{T} - \frac{s}{|\boldsymbol{\rho}|^3}\boldsymbol{\rho}^\mathsf{T}\right) + \frac{s}{|\boldsymbol{\rho}|}\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{\rho}^\mathsf{T}} \tag{C.22}$$

$$= \underbrace{\left(\frac{c}{2|\boldsymbol{\rho}|^2} - \frac{s}{|\boldsymbol{\rho}|^3}\right)}_{b}\left(\boldsymbol{\rho} \times \boldsymbol{\rho}^\mathsf{T}\right) + \underbrace{\frac{s}{|\boldsymbol{\rho}|}}_{a}\mathbf{I}_3 \tag{C.23}$$

Keeping the definition of a and b in mind we are able to provide the whole Jacobian of a quaternion rotation $\boldsymbol{q}$ by an exponential coordinate rotation $\boldsymbol{\rho}$:

$$\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{\rho}^\mathsf{T}} = \begin{bmatrix} -\frac{a\boldsymbol{\rho}_x}{2} & -\frac{a\boldsymbol{\rho}_y}{2} & -\frac{a\boldsymbol{\rho}_z}{2} \\ b\boldsymbol{\rho}_x^2 - a & b\boldsymbol{\rho}_x\boldsymbol{\rho}_y & b\boldsymbol{\rho}_x\boldsymbol{\rho}_z \\ b\boldsymbol{\rho}_y\boldsymbol{\rho}_x & b\boldsymbol{\rho}_y^2 - a & b\boldsymbol{\rho}_y\boldsymbol{\rho}_z \\ b\boldsymbol{\rho}_z\boldsymbol{\rho}_x & b\boldsymbol{\rho}_z\boldsymbol{\rho}_y & b\boldsymbol{\rho}_z^2 - a \end{bmatrix} \tag{C.24}$$

From the derivations of a and b it is evident that the Jacobian of a quaternion rotation by a rotation in exponential coordinates cannot be calculated directly if the angle of rotation (and therefore $|\boldsymbol{\rho}|$) is zero. However, in any implementation the computation of the Jacobian for angles that are just *close enough* to zero is problematic already, due to

the numerical instabilities that arise in this case. To overcome this issue we approximate the critical fractions by a Taylor series expansion around zero and use their values for small angles.

$$a \approx \frac{1}{2} - \frac{|\boldsymbol{\rho}|^2}{48} + \frac{|\boldsymbol{\rho}|^4}{3840} + O(|\boldsymbol{\rho}|^5) \tag{C.25}$$

$$b \approx -\frac{1}{24} + \frac{|\boldsymbol{\rho}|^2}{960} - \frac{|\boldsymbol{\rho}|^4}{107520} + O(|\boldsymbol{\rho}|^5) \tag{C.26}$$

C.2.2. Rotation of a Point by Point

To derive the Jacobian of a rotation in exponential coordinates $\boldsymbol{\rho}$ of a point $\mathbf{p}$ by the point itself we use again the quaternion representation $\boldsymbol{q}$ of the rotation. In particular, we consider the rotation matrix associated with that quaternion as in equation C.15. The Jacobian of this operation by the point $\mathbf{p}$ is now straight forward:

$$\frac{\partial \mathbf{p}'}{\partial \mathbf{p}} = \frac{\partial \boldsymbol{\rho}(\mathbf{p})}{\partial \mathbf{p}} = \frac{\partial \mathbf{R}_q \times \mathbf{p}}{\partial \mathbf{p}} = \mathbf{R}_q \tag{C.27}$$

Again, we run into implementation problems for small rotation angles when converting from the exponential coordinate representation to the quaternion representation. In this case we use the Taylor series expansion from equation C.25 where we approximated the critical factor for the conversion around zero.

C.2.3. Composition of Rotations

The composition $\boldsymbol{\rho}$ of two rotations in exponential coordinates $\boldsymbol{\tau}$ and $\boldsymbol{\upsilon}$ is not straight forward. On the other hand, the composition of two quaternion rotations is just a quaternion multiplication. Therefore, we decided to convert the exponential rotation to quaternions before the composition. The quaternion of the composition of two exponential rotations $\boldsymbol{\tau}$ and $\boldsymbol{\upsilon}$ is given as:

$$\boldsymbol{q} = \mathrm{quat}(\boldsymbol{\tau}) \cdot \mathrm{quat}(\boldsymbol{\upsilon}) \tag{C.28}$$

where $\mathrm{quat}(.)$ the conversion function defined in equation C.12. The exponential representation of a quaternion can be obtained by inversion of the conversion function:

$$\boldsymbol{\rho} = \exp(\boldsymbol{q}) \stackrel{\text{def}}{=} \underbrace{\frac{2\cos^{-1}(\boldsymbol{q}_\omega)}{\sin(\cos^{-1}(\boldsymbol{q}_\omega))}}_{s(\boldsymbol{q}_\omega)} \begin{pmatrix} \boldsymbol{q}_x \\ \boldsymbol{q}_y \\ \boldsymbol{q}_z \end{pmatrix} \tag{C.29}$$

In the following, we will refer to the scalar factor of this function by $s(\boldsymbol{q}_\omega)$.

By First Operand

The Jacobian of the composition $\boldsymbol{\rho}$ by the first operand $\boldsymbol{\tau}$ can be decomposed as:

$$\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{\tau}} = \frac{\partial \exp(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial \mathrm{quat}(\boldsymbol{\tau})} \frac{\partial \mathrm{quat}(\boldsymbol{\tau})}{\partial \boldsymbol{\tau}} \tag{C.30}$$

Jacobian of $\exp(\boldsymbol{q})$ by $\boldsymbol{q}$:

$$\frac{\partial \exp(\boldsymbol{q})}{\partial \boldsymbol{q}} = \begin{bmatrix} \mathbf{J}_{\boldsymbol{q}_\omega} & \mathbf{J}_{\boldsymbol{q}_{xyz}} \end{bmatrix} \tag{C.31}$$

$$\mathbf{J}_{\boldsymbol{q}_\omega} = \frac{\partial s(\boldsymbol{q}_\omega)}{\partial \boldsymbol{q}_\omega} \begin{pmatrix} \boldsymbol{q}_x \\ \boldsymbol{q}_y \\ \boldsymbol{q}_z \end{pmatrix} \tag{C.32}$$

$$\frac{\partial s(\boldsymbol{q}_\omega)}{\partial \boldsymbol{q}_\omega} = \frac{2\cos^{-1}(\boldsymbol{q}_\omega)\boldsymbol{q}_\omega}{(1-\boldsymbol{q}_\omega^2)^{\frac{3}{2}}} - \frac{2}{1-\boldsymbol{q}_\omega^2} \tag{C.33}$$

$$\mathbf{J}_{\boldsymbol{q}_{xyz}} = \begin{bmatrix} s(\boldsymbol{q}_\omega) & 0 & 0 \\ 0 & s(\boldsymbol{q}_\omega) & 0 \\ 0 & 0 & s(\boldsymbol{q}_\omega) \end{bmatrix} \tag{C.34}$$

Jacobian of $\boldsymbol{q} = \boldsymbol{p} \cdot \boldsymbol{r}$ by $\boldsymbol{p}$:

$$\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{p} \cdot \boldsymbol{r}}{\partial \boldsymbol{p}} \tag{C.35}$$

$$= \begin{bmatrix} \boldsymbol{r}_\omega & -\boldsymbol{r}_x & -\boldsymbol{r}_y & -\boldsymbol{r}_z \\ \boldsymbol{r}_x & \boldsymbol{r}_\omega & \boldsymbol{r}_z & -\boldsymbol{r}_y \\ \boldsymbol{r}_y & -\boldsymbol{r}_z & \boldsymbol{r}_\omega & \boldsymbol{r}_x \\ \boldsymbol{r}_z & \boldsymbol{r}_y & -\boldsymbol{r}_x & \boldsymbol{r}_\omega \end{bmatrix} \tag{C.36}$$

By Second Operand

The Jacobian of the composition $\boldsymbol{\rho}$ by the second operand $\boldsymbol{\upsilon}$ can be decomposed as:

$$\frac{\partial \boldsymbol{\rho}}{\partial \boldsymbol{\upsilon}} = \frac{\partial \exp(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial \mathrm{quat}(\boldsymbol{\upsilon})} \frac{\partial \mathrm{quat}(\boldsymbol{\upsilon})}{\partial \boldsymbol{\upsilon}} \tag{C.37}$$

Jacobian of $\boldsymbol{q} = \boldsymbol{p} \cdot \boldsymbol{r}$ by $\boldsymbol{r}$:

$$\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{r}} = \frac{\partial \boldsymbol{p} \cdot \boldsymbol{r}}{\partial \boldsymbol{r}} \tag{C.38}$$

$$= \begin{bmatrix} \boldsymbol{p}_\omega & -\boldsymbol{p}_x & -\boldsymbol{p}_y & -\boldsymbol{p}_z \\ \boldsymbol{p}_x & \boldsymbol{p}_\omega & -\boldsymbol{p}_z & \boldsymbol{p}_y \\ \boldsymbol{p}_y & \boldsymbol{p}_z & \boldsymbol{p}_\omega & -\boldsymbol{p}_x \\ \boldsymbol{p}_z & -\boldsymbol{p}_y & \boldsymbol{p}_x & \boldsymbol{p}_\omega \end{bmatrix} \tag{C.39}$$

C.3. Poses

A pose (or frame) is given as a tuple:

$$\mathcal{F} = (\mathbf{t}_\mathcal{F}, \boldsymbol{\rho}_\mathcal{F}) \tag{C.40}$$

Application of pose/frame to point Poses can be used to transform points or vectors from one coordinate frame to another. More precisely, a pose $\mathcal{F}^\mathcal{G}$ transforms points and vectors from $\mathcal{F}$ to $\mathcal{G}$:

$$\mathbf{p}^\mathcal{G} = \mathcal{F}^\mathcal{G}(\mathbf{p}^\mathcal{F}) = \mathbf{t}_\mathcal{F}^\mathcal{G} + \boldsymbol{\rho}_\mathcal{F}^\mathcal{G}(\mathbf{p}^\mathcal{G}) \tag{C.41}$$

Inversion of a pose The inverse transformation of a pose $\mathcal{F}$ is denoted by $\mathcal{F}^{-1}$ and defined as:

$$\mathcal{F}^{-1} = (-\boldsymbol{\rho}_\mathcal{F}(-\mathbf{t}_\mathcal{F}), -\boldsymbol{\rho}_\mathcal{F}) \tag{C.42}$$

Composition of poses The transformation of two poses $\mathcal{G}^\mathcal{H}$ and $\mathcal{F}^\mathcal{G}$ can be concatenated, thus giving a new transformation. Equivalently, we can say the frame $\mathcal{F}^\mathcal{G}$ gets represented with respect to frame $\mathcal{H}$.

$$\mathcal{F}^\mathcal{H} = \mathcal{G}^\mathcal{H} \circ \mathcal{F}^\mathcal{G} = (\mathbf{t}_\mathcal{G}^\mathcal{H} + \boldsymbol{\rho}_\mathcal{G}^\mathcal{H}(\mathbf{t}_\mathcal{F}^\mathcal{G}), \boldsymbol{\rho}_\mathcal{G}^\mathcal{H} \circ \boldsymbol{\rho}_\mathcal{F}^\mathcal{G}) \tag{C.43}$$

C.3.1. Inversion of a Pose

Given an inverse pose $\mathcal{F}^{-1}$ its Jacobian by $\mathcal{F}$ can be decomposed into the respective translational and rotational parts:

$$\frac{\partial \mathcal{F}^{-1}}{\partial \mathcal{F}} = \begin{bmatrix} \mathbf{J}_{\mathbf{tt}} & \mathbf{J}_{\mathbf{t}\boldsymbol{\rho}} \\ \mathbf{J}_{\boldsymbol{\rho}\mathbf{t}} & \mathbf{J}_{\boldsymbol{\rho}\boldsymbol{\rho}} \end{bmatrix} \tag{C.44}$$

$$\mathbf{J}_{\mathbf{tt}} = \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}(-\mathbf{t}_{\mathcal{F}})}{\partial \mathbf{t}_{\mathcal{F}}} = \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}(-\mathbf{t}_{\mathcal{F}})}{\partial - \mathbf{t}_{\mathcal{F}}} \frac{\partial - \mathbf{t}_{\mathcal{F}}}{\partial \mathbf{t}_{\mathcal{F}}} = -\frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}(-\mathbf{t}_{\mathcal{F}})}{\partial - \mathbf{t}_{\mathcal{F}}} \tag{C.45}$$

which is given in equation C.27.

$$\mathbf{J}_{\mathbf{t}\boldsymbol{\rho}} = \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}(-\mathbf{t}_{\mathcal{F}})}{\partial \boldsymbol{\rho}_{\mathcal{F}}} = \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}(-\mathbf{t}_{\mathcal{F}})}{\partial - \boldsymbol{\rho}_{\mathcal{F}}} \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}}{\partial \boldsymbol{\rho}_{\mathcal{F}}} = -\frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}(-\mathbf{t}_{\mathcal{F}})}{\partial - \boldsymbol{\rho}_{\mathcal{F}}} \tag{C.46}$$

which is given in equation C.16.

$$\mathbf{J}_{\boldsymbol{\rho}\mathbf{t}} = \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}}{\partial \mathbf{t}_{\mathcal{F}}} = \mathbf{0}_{[3\times3]} \tag{C.47}$$

$$\mathbf{J}_{\boldsymbol{\rho}\boldsymbol{\rho}} = \frac{\partial - \boldsymbol{\rho}_{\mathcal{F}}}{\partial \boldsymbol{\rho}_{\mathcal{F}}} = -\mathbf{1}_{[3\times3]} \tag{C.48}$$

C.3.2. Composition of Poses

Let a pose composition be given as $\mathcal{R} = \mathcal{A} \circ \mathcal{B}$. For better readability, we omit the superscripts here and assume that the poses are compatible, i.e., the can be composed as defined above.

By First Operand

The Jacobian of $\mathcal{R}$ by $\mathcal{A}$ can be decomposed into the respective translational and rotational parts of $\mathcal{R}$ and $\mathcal{A}$:

$$\frac{\partial \mathcal{R}}{\partial \mathcal{A}} = \begin{bmatrix} \mathbf{J}_{\mathbf{tt}} & \mathbf{J}_{\mathbf{t}\boldsymbol{\rho}} \\ \mathbf{J}_{\boldsymbol{\rho}\mathbf{t}} & \mathbf{J}_{\boldsymbol{\rho}\boldsymbol{\rho}} \end{bmatrix} \tag{C.49}$$

$$\mathbf{J}_{\mathbf{tt}} = \frac{\partial \mathbf{t}_{\mathcal{R}}}{\partial \mathbf{t}_{\mathcal{A}}} = \frac{\partial \mathbf{t}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}(\mathbf{t}_{\mathcal{B}})}{\partial \mathbf{t}_{\mathcal{A}}} = \mathbf{1}_{[3\times3]} \tag{C.50}$$

$$\mathbf{J}_{\mathbf{t}\boldsymbol{\rho}} = \frac{\partial \mathbf{t}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}(\mathbf{t}_{\mathcal{B}})}{\partial \boldsymbol{\rho}_{\mathcal{A}}} = \frac{\partial \boldsymbol{\rho}_{\mathcal{A}}(\mathbf{t}_{\mathcal{B}})}{\partial \boldsymbol{\rho}_{\mathcal{A}}} \tag{C.51}$$

which is given in equation C.16.

$$\mathbf{J}_{\boldsymbol{\rho}\mathbf{t}} = \frac{\partial \boldsymbol{\rho}_{\mathcal{A}} \circ \boldsymbol{\rho}_{\mathcal{B}}}{\partial \mathbf{t}_{\mathcal{A}}} = \mathbf{0}_{[3\times3]} \tag{C.52}$$

$$\mathbf{J}_{\boldsymbol{\rho}\boldsymbol{\rho}} = \frac{\partial \boldsymbol{\rho}_{\mathcal{A}} \circ \boldsymbol{\rho}_{\mathcal{B}}}{\partial \boldsymbol{\rho}_{\mathcal{A}}} \tag{C.53}$$

which is given in equation C.30.

By Second Operand

$$\frac{\partial \mathcal{R}}{\partial \mathcal{B}} = \begin{bmatrix} \mathbf{J_{tt}} & \mathbf{J_{t\rho}} \\ \mathbf{J_{\rho t}} & \mathbf{J_{\rho\rho}} \end{bmatrix} \tag{C.54}$$

$$\mathbf{J_{tt}} = \frac{\partial \mathbf{t}_{\mathcal{R}}}{\partial \mathbf{t}_{\mathcal{B}}} = \frac{\partial \mathbf{t}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}(\mathbf{t}_{\mathcal{B}})}{\partial \mathbf{t}_{\mathcal{B}}} = \frac{\partial \boldsymbol{\rho}_{\mathcal{A}}(\mathbf{t}_{\mathcal{B}})}{\partial \mathbf{t}_{\mathcal{B}}} \tag{C.55}$$

which is given in equation C.27.

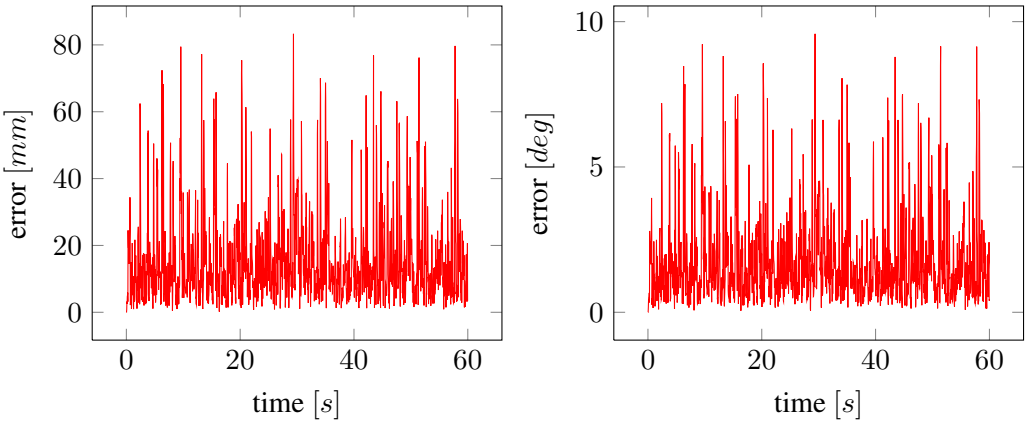
$$\mathbf{J_{t\rho}} = \frac{\partial \mathbf{t}_{\mathcal{A}} + \boldsymbol{\rho}_{\mathcal{A}}(\mathbf{t}_{\mathcal{B}})}{\partial \boldsymbol{\rho}_{\mathcal{B}}} = \mathbf{0}_{[3\times3]} \tag{C.56}$$

$$\mathbf{J_{\rho t}} = \frac{\partial \boldsymbol{\rho}_{\mathcal{A}} \circ \boldsymbol{\rho}_{\mathcal{B}}}{\partial \mathbf{t}_{\mathcal{B}}} = \mathbf{0}_{[3\times3]} \tag{C.57}$$

$$\mathbf{J_{\rho\rho}} = \frac{\partial \boldsymbol{\rho}_{\mathcal{A}} \circ \boldsymbol{\rho}_{\mathcal{B}}}{\partial \boldsymbol{\rho}_{\mathcal{B}}} \tag{C.58}$$

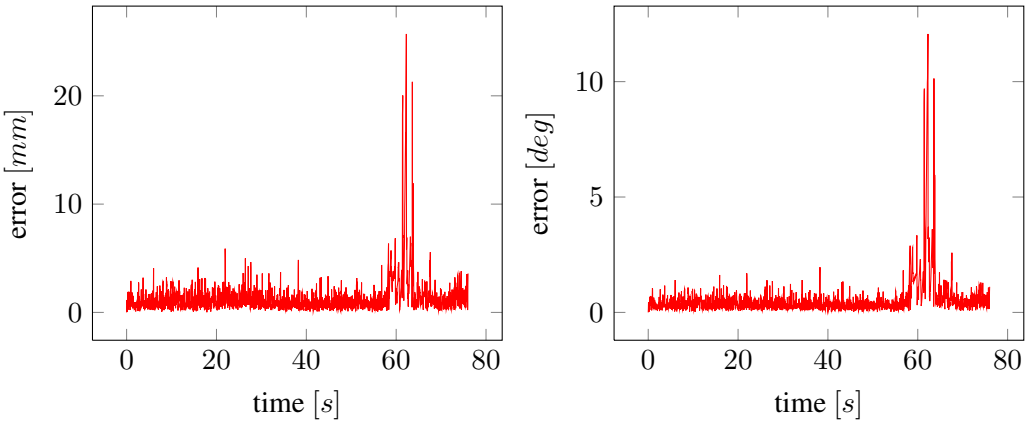
which is given in equation C.37.

D. Experimental Data



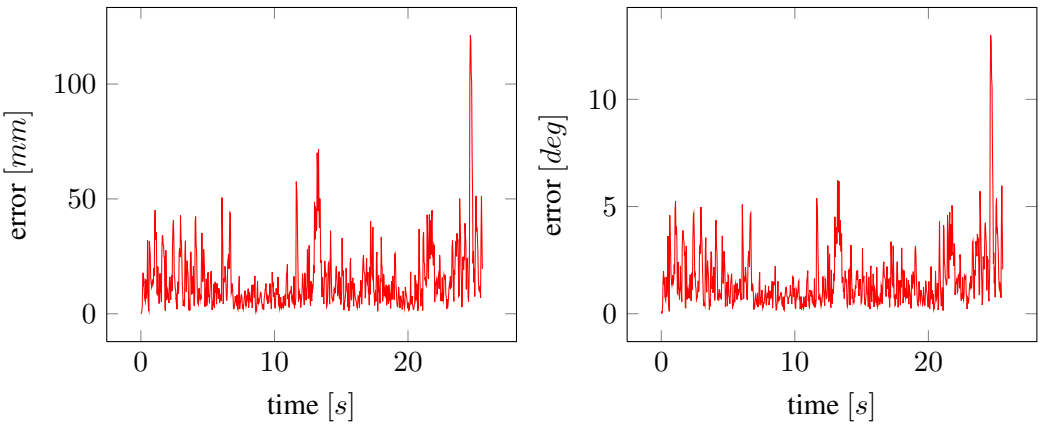
(a) camera position errors (b) camera rotation errors

Figure D.1.: Camera position and rotation errors on sequence A, scenario I. Due to the large distance between the camera and the object the position errors are relatively high. However, no systematic error can be observed - the system runs steadily for the length of the whole sequence.



(a) camera position errors (b) camera rotation errors

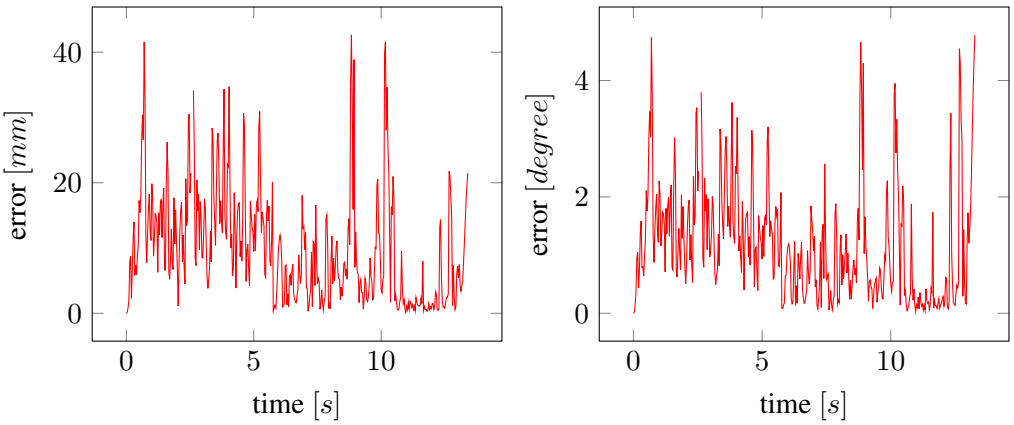
Figure D.2.: Camera position and rotation errors on sequence B, scenario I. The close-up views of the object allow for very accurate pose estimations. At about 60 seconds the camera gets too close to the object such that only a few point features of the object can be tracked. After 76 seconds the VSLAM system loses track when the object disappears from the field of view.



(a) camera position errors

(b) camera rotation errors

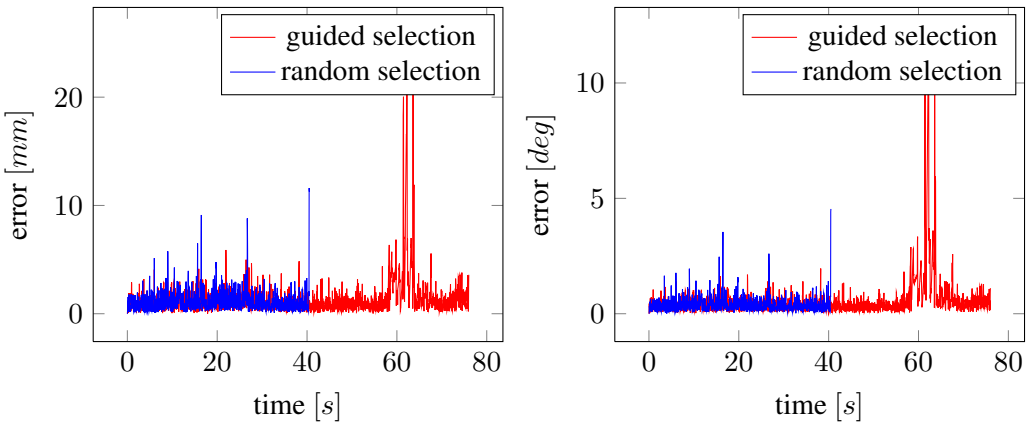
Figure D.3.: Camera position and rotation error on sequence C, scenario I. We observe an increase in the camera pose error at about 12 seconds when the object almost completely disappears from the field of view. At the end of the sequence, the camera jumping causes some bigger estimation errors.



(a) camera position errors

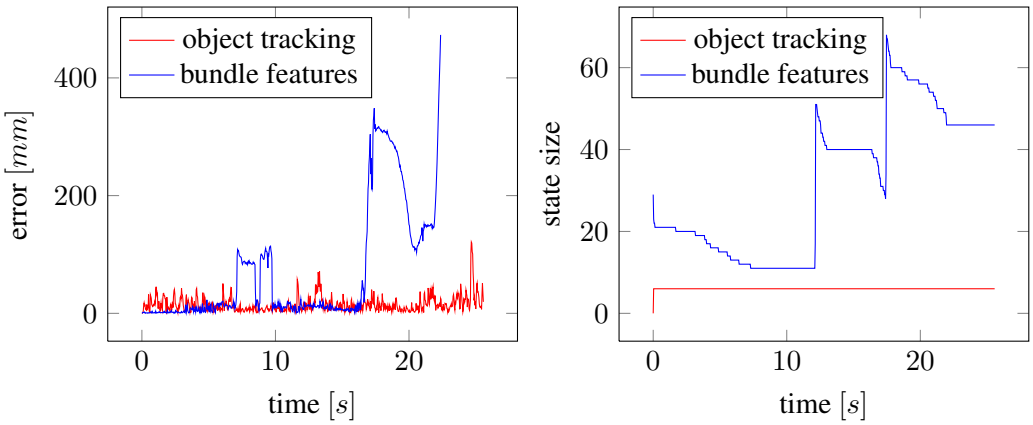
(b) camera rotation errors

Figure D.4.: Camera position and rotation error on sequence D, scenario I. In comparison to sequence C, the additional object decreases the camera pose errors. However, when the packet completely occludes the Caro-tin the system loses track.



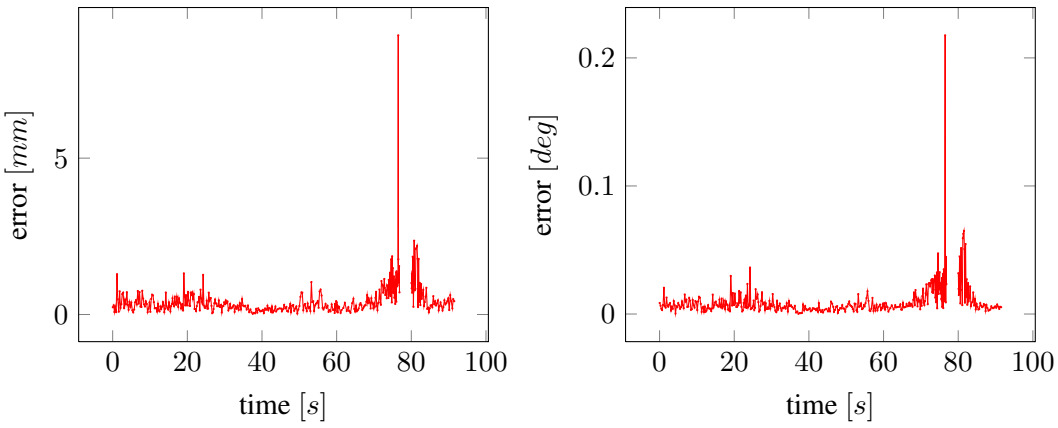
(a) camera position errors (b) camera rotation errors

Figure D.5.: Camera position and rotation error on sequence B, scenario I vs. scenario II. Here we see the difference between a guided and a purely random measurement candidate selection. Occasionally, the random selection causes some bigger estimation errors. After 40 seconds in this experiment the system can not recover from a serious pose estimation error.



(a) camera position errors (b) state sizes

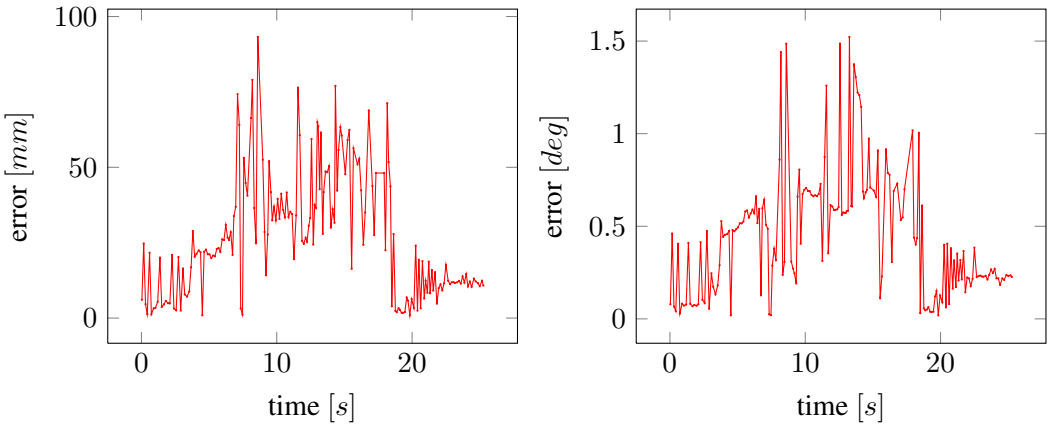
Figure D.6.: Camera position error and state size on sequence C, scenario I vs. scenario IV. In scenario I, just objects get tracked and in scenario IV just bundle features. One can clearly see that object tracking allows for more precise camera pose estimates while consuming less memory.



(a) object position errors

(b) object rotation errors

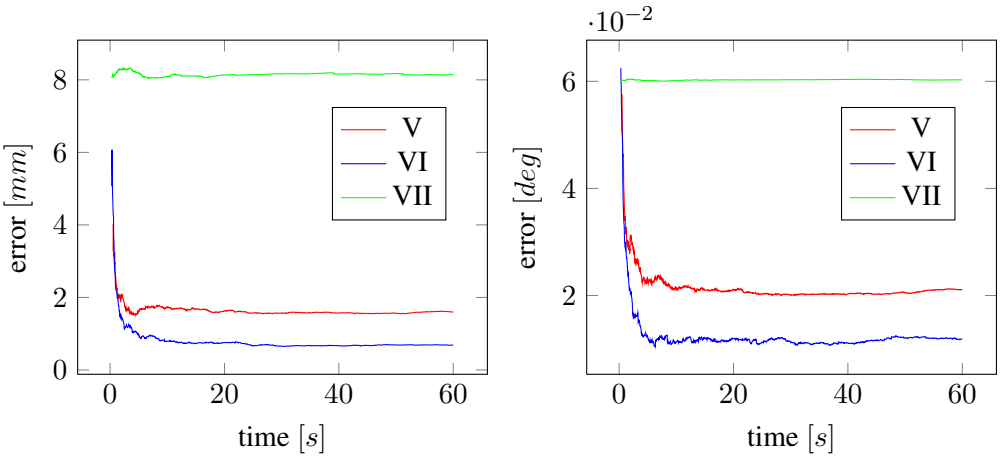
Figure D.7.: Pose measurement position and rotation errors on sequence B. Due to the closeup views of the object the pose errors are very small. However, during the period in which the object is not or barely visible the errors increase. Between 77 and 80 seconds there are no pose measurements at all.



(a) object position errors

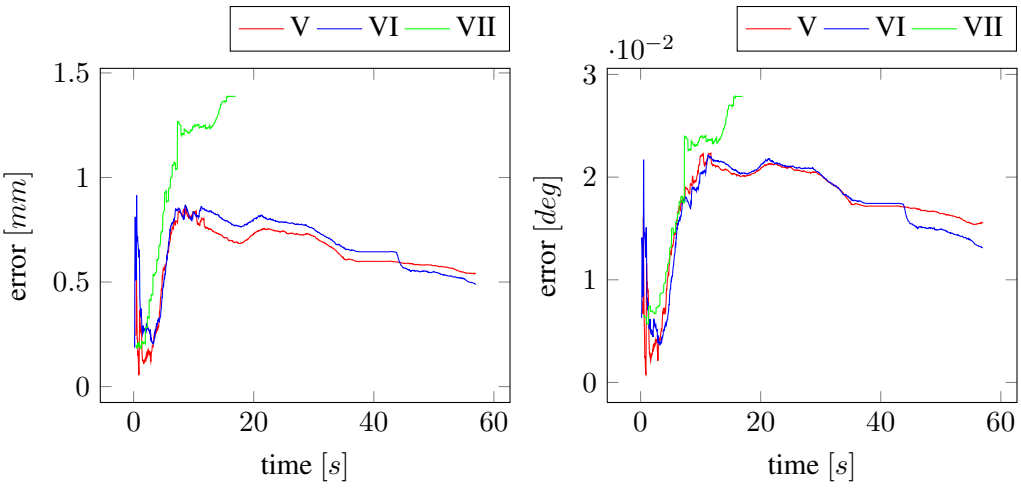
(b) object rotation errors

Figure D.8.: Pose measurement position and rotation errors on sequence C. The increase of the error up to second 18 correlates with the distance of the camera to the object.



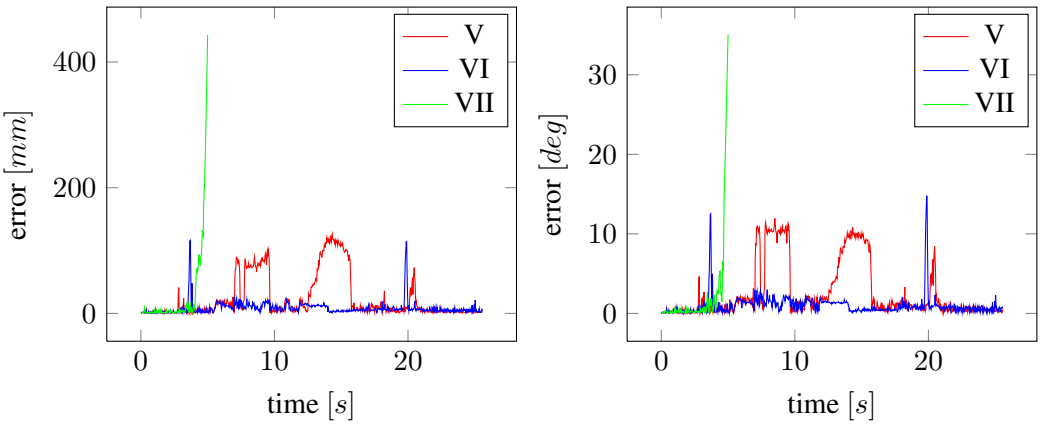
(a) object position errors (b) object rotation errors

Figure D.9.: Object position and rotation errors on sequence A, scenarios V, VI, and VII. It is clearly visible that without feature template measurements (green line) the accuracy is worst. The best result in this experiment was obtained by considering feature templates only (blue line).



(a) object position errors (b) object rotation errors

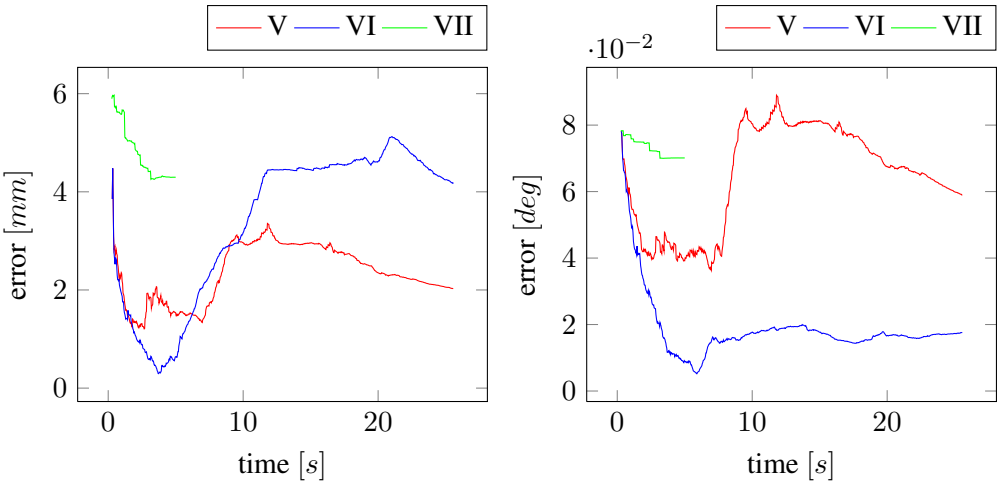
Figure D.10.: Object position and rotation errors on sequence B. Again, we see how the use of feature templates improved the estimation accuracy. In this experiment it did not make a big difference whether we additionally used object recognition (red line) or disregarded them (blue line).



(a) camera position errors

(b) camera rotation errors

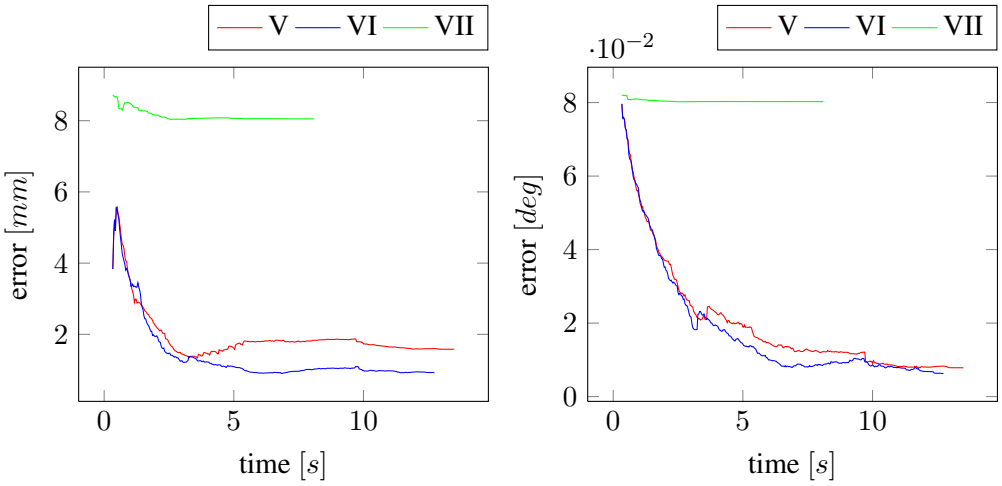
Figure D.11.: Camera position and rotation errors on sequence C, scenarios V, VI, and VII. With just pose measurements (green line) the system loses track soon. Even with template features and pose measurements (red line) we observe some big estimation errors. We got the best results when just considering feature templates (blue line).



(a) object position errors

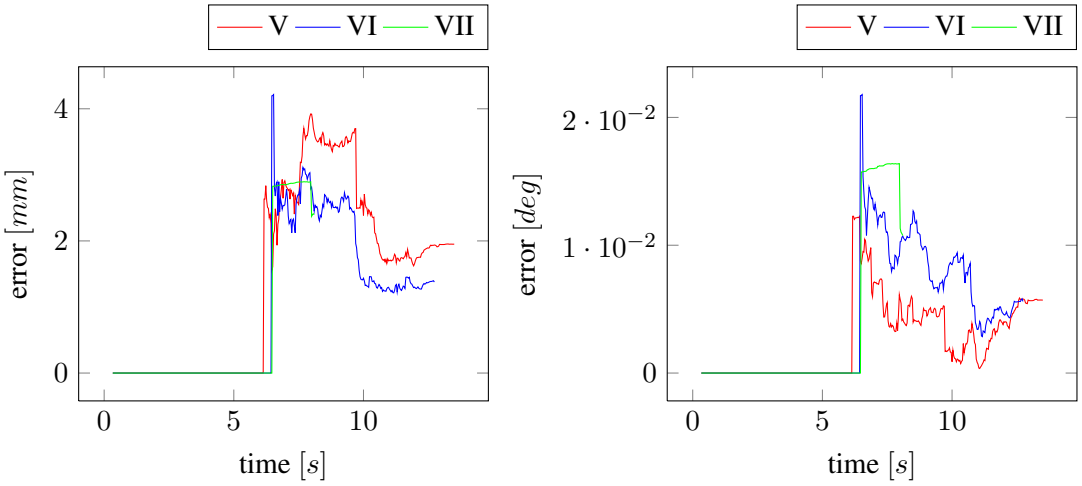
(b) object rotation errors

Figure D.12.: Object position and rotation errors on sequence C, scenarios V, VI, and VII. For the object, the position error is smaller when the pose measurements are considered besides the feature templates (left image, red line). However, that is not the case for the rotation error (right image).



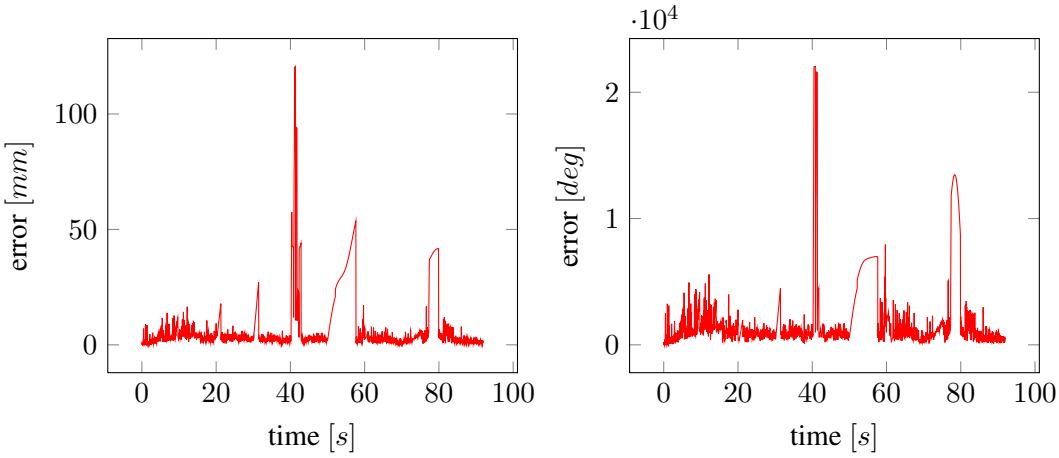
(a) object position errors (b) object rotation errors

Figure D.13.: Caro-tin position and rotation errors on sequence D, scenarios V, VI, and VII. Again, the estimation errors are smallest when ignoring the pose measurements and just measuring the feature templates (blue line).



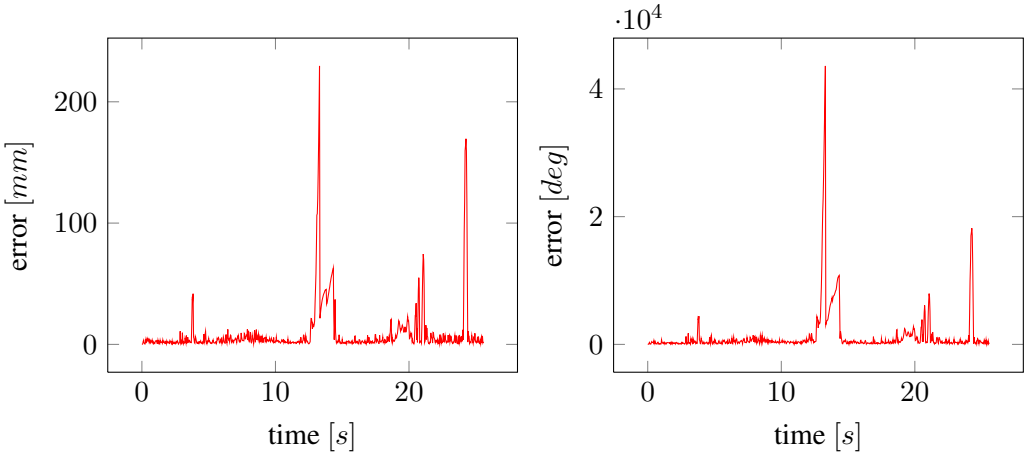
(a) object position errors (b) object rotation errors

Figure D.14.: Packet position and rotation errors on sequence D, scenarios V, VI, and VII. Values of zero until about six seconds of the sequence indicate that the object has not been initialised yet.



(a) camera position errors (b) camera rotation errors

Figure D.15.: Camera position and rotation errors on sequence B, scenario VIII. For this experiment, some frames of sequence B have been blacked out. We can see how the system recovers after periods of large estimation errors.



(a) camera position errors (b) camera rotation errors

Figure D.16.: Camera position and rotation errors on sequence D, scenario VIII. With the help of the relocation strategy, the system recovers from the large estimation error in the middle of the sequence.

Bibliography

[1] *Pov-ray - the persistance of vision raytracer*. http://www.povray.org/.

[2] Howard Anton, *Calculus, a new horizon, 6th edition*, New York: Wiley, 1999.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool, *Surf: Speeded up robust features*, European Conference on Computer Vision (2006), 404–417.

[4] NDF Campbell, G. Vogiatzis, C. Hernandez, and R. Cipolla, *Automatic 3d object segmentation in multiple views using volumetric graph-cuts*, Image and Vision Computing **28** (2010), no. 1, 14–25.

[5] Robert O. Castle, D. J. Gawley, Georg Klein, and David W. Murray, *Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras*, Proc. international conference on robotics and automation, rome, italy, april 10-14, 2007, 2007, pp. 4102–4107.

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

[7] Laura A. Clemente, Andrew J. Davison, Ian D. Reid, José Neira, and Juan D. Tardós, *Mapping large loops with a single hand-held camera.*, Robotics: Science and systems, July 10, 2008.

[8] Daniel Cremers, Mikael Rousson, and Rachid Deriche, *A review of statistical approaches to level set segmentation: integrating color, texture, motion and shape*, International Journal of Computer Vision **72** (2007), no. 2, 195–215.

[9] Daniel Cremers, Florian Tischhäuser, Joachim Weickert, and Christoph Schnörr, *Diffusion snakes: introducing statistical shape knowledge into the mumford-shah functional*, International Journal of Computer Vision **50** (2002), 295–313.

[10] Bogusłlav Cyganek and Jan Borgosz, *A Comparative Study of Performance and Implementation of Some Area-Based Stereo Algorithms* (2001), 709–716.

[11] Andrew J. Davison and David W. Murray, *Simultaneous localization and map-building using active vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 7, 865–880.

[12] W.F. Denham and S. Pines, *Sequential estimation when measurement function nonlinearity is comparable to measurement error*, American Institute of Aeronautics and Astronautics **4** (1966), no. 6, 1071–1076.

[13] Ethan Eade and Tom Drummond, *Unified loop closing and recovery for real time monocular slam*, British machine vision conference, 2008.

[14] Sanja Fiedler, Marko Boben, and Aleš Leonardis, *Optimization framework for learning a hierarchical shape vocabulary for object class detection*, Proceedings of the britith mashine vision conference, 2009.

[15] Martin A. Fischler and Robert C. Bolles, *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM **24** (1981), no. 6, 381–395.

[16] Udo Frese, *Treemap: An o(logn) algorithm for simultaneous localization and mapping*, in spatial cognition iv, c. freksa, ed, 2005, pp. 455–476.

[17] Jan Funke and Tobias Pietzsch, *An evaluation framework for visual slam*, Proceedings of the britith mashine vision conference, 2009.

[18] ———, *VSLAM evaluation framework*, 2009. http://vslam.inf.tu-dresden.de.

[19] Iryna Gordon and David G. Lowe, *What and where: 3D object recognition with accurate pose*, Toward category-level object recognition (2006), 67–82.

[20] Richard I. Hartley and Andrew Zisserman, *Multiple view geometry in computer vision*, Second, Cambridge University Press, ISBN: 0521540518, 2004.

[21] Andrew H. Jazwinski, *Stochastic processes and filtering theory*, Academic Press, 1970.

[22] Emil Rudolf Kalman, *A new approach to linear fitting and prediction problems*, Journal of Basic Engineering (1960).

[23] Autonomous selection, registration, and recognition of objects for visual slam in indoor environments (2007)

[24] Vincent Lepetit and Pascal Fua, *Keypoint recognition using randomized trees*, IEEE Transactions on Pattern Analysis and Machine Intelligence (2006), 1465–1479.

[25] David G. Lowe, *Object recognition from local scale-invariant features*, 1999, pp. 1150–1157.

[26] _____, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision **60** (2004), 91–110.

[27] Chistopher Mei, Gabe Sibley, Mark Cummins, Paul Newman, and Ian Reid, *A constant time efficient stereo slam system*, Bmvc, 2009.

[28] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua, *Accurate non-iterative o(n) solution to the pnp problem*, IEEE International Conference on Computer Vision, 2007October.

[29] J. Tardos Neira, *Data association in stochastic mapping using the joint compatibility test*, IEEE Transactions on Robotics and Automation **17** (2001), 890–897.

[30] Tobias Pietzsch, *Efficient Feature Parameterisation for Visual SLAM Using Inverse Depth Bundles*, Proc. BMVC'08, 2008.

[31] Fabio Ramos, Juan Nieto, and Hugh Durrant-Whyte, *Combining object recognition and slam for extended map representations* **29** (2008), 55–64.

[32] Edward Rosten and Tom Drummond, *Fusing points and lines for high performance tracking.*, IEEE International Conference on Computer Vision, 2005October, pp. 1508–1511.

[33] Axel Rottmann, Óscar M. Mozos, Cyrill Stachniss, and Wolfram Burgard, *Semantic place classification of indoor environments with mobile robots using boosting*, Proceedings of the national conference on artificial intelligence, 2005, pp. 1306.

[34] Eli Shechtman and Michal Irani, *Matching local self-similarities across images and videos*, IEEE Conference on Computer Vision and Pattern Recognition 2007 (CVPR'07), 2007June.

[35] Randall C. Smith and Peter Cheeseman, *On the representation and estimation of spatial uncertainty*, The International Journal of Robotics Research **5** (1986), no. 4, 56.

[36] James Stewart, *Calculus 5th edition*, Brooks Cole, 2002.

[37] Peter Swerling, *A proposed stagewise differential correction procedure for satellite tracking and prediction*, Technical Report P-1292, RAND Corporation (1958).

[38] Simon Taylor and Tom Drummond, *Multiple target localisation at over 100 fps*, British machine vision conference, 2009September.

[39] Sebasian Thrun, Wolfram Burgard, Deepayan Chakrabarti, Rosemary Emery, Yufeng Liu, and Christian Martin, *A real-time algorithm for acquiring multi-planar volumetric models with mobile robots*, Robotics Research (2003), 21–35.

[40] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic robotics*, The MIT Press, 2005.

[41] Phil Torr and Andrew Zisserman, *Mlesac: A new robust estimator with application to estimating image geometry*, Computer Vision and Image Undersatnding **78** (2000), no. 1, 138–156.

[42] Zhuowen Tu, Xiangrong Chen, Alan L. Yuille, and Song-Chun Zhu, *Image parsing: Unifying segmentation, detection, and recognition*, 2005.

[43] S. Wangsiripitak and D. W. Murray, *Avoiding moving outliers in visual slam by tracking moving objects*, Proc. IEEE International Conference on Robotics and Automation ICRA '09, May 2009, pp. 375–380.

[44] Brian Williams, Mark Cummins, José Neira, Paul Newman, Ian Reid, and Juan Tardós, *A comparison of loop closing techniques in monocular slam*, Robotics and Autonomous Systems (2009).

[45] Brian Williams, Georg Klein, and Ian Reid, *Real-time SLAM relocalisation*, Proc. international conference on computer vision, 2007.

[46] Changchang Wu, *SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)*, 2007. `http://cs.unc.edu/~ccwu/siftgpu`.

Selbständigkeitserklärung

Hiermit erkläre ich, Jan Funke, dass die Arbeit “An Integrative Approach to Object Recognition in VSLAM” von mir selbst und ohne andere als die darin angegebenen Quellen angefertigt worden ist.

Dresden, den 15.09.2010

Jan Funke