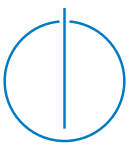
TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Master's Thesis in Robotics, Cognition, Intelligence

Efficient Techniques for Accurate Visual Place Recognition

Tim Stricker



TECHNICAL UNIVERSITY OF MUNICH

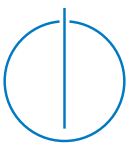
DEPARTMENT OF INFORMATICS

Master's Thesis in Robotics, Cognition, Intelligence

Efficient Techniques for Accurate Visual Place Recognition

Effiziente Verfahren für eine genaue visuelle Ortserkennung

Author: Tim Stricker
Supervisor: Prof. Dr. Daniel Cremers
Advisors: Nikolaus Demmel, M.Sc. and Dr. Vladyslav Usenko
Submission Date: June 15, 2020



I confirm that this master's thesis in robotics, cognition, intelligence is my own work and I have documented all sources and material used.

Munich, June 12, 2020 Tim Stricker

Acknowledgments

At this point, I would like to thank everyone who has supported me throughout my studies and especially in the making of this thesis. First and foremost, a heartfelt thank you to my advisors, Nikolaus Demmel and Dr. Vladyslav Usenko, for offering me the topic and for all the helpful discussions and suggestions during this period.

A special thanks goes to my supervisor, Prof. Dr. Daniel Cremers, for giving me the opportunity to pursue my thesis at the Chair of Computer Vision and Artificial Intelligence. The incredible work he and his team is doing at the chair first inspired me to focus my attention on the subject of computer vision.

Next, I want to express my gratitude to the proofreaders of this work, Nadia Haubner and Robin Petereit. Thank you for all the valuable feedback on the first draft of the thesis.

Last but not least, I would especially like to thank my parents and my girlfriend Tanja for their endless support during the course of my studies. This would never have been possible without you and I am eternally grateful for that.

Abstract

Simultaneous Localization and Mapping (SLAM) is one of the most prolific topics in the wide area of computer vision. It describes the problem of tracking one or multiple cameras in an unknown environment while generating an accurate map of the surroundings at the same time. A key challenge of all SLAM systems is to prevent the accumulation of error over the course of its process. This can be achieved by detecting loops and subsequently performing a global optimization. The task of detecting previously visited locations in images is more generally known as visual place recognition. In this thesis, we evaluate the accuracy and efficiency of multiple place recognition approaches which consider the task as a pure image retrieval problem using local feature descriptors. To that end, we provide an overview of promising state-of-the-art methods and additionally introduce a novel solution based on locality-sensitive hashing. The algorithm uses a hashing procedure to cluster feature descriptors and treats those hashes as terms in a bag-of-words model. The approaches get evaluated in a new flexible and extensible benchmarking suite which was developed as part of this work. The evaluation includes analyses of the possible parameter settings for each method, the impact of the feature extractor choice and a final comparison of all regarded solutions. Based on these findings, we design and implement an open-source library for place recognition containing well documented and highly efficient versions of a relevant subset of the algorithms. The library is easy to use and can be extended to incorporate other approaches with very little effort. The thesis finishes with a final analysis which proves the advantages of our implementation.

Contents

1. Introduction

1.1. Motivation

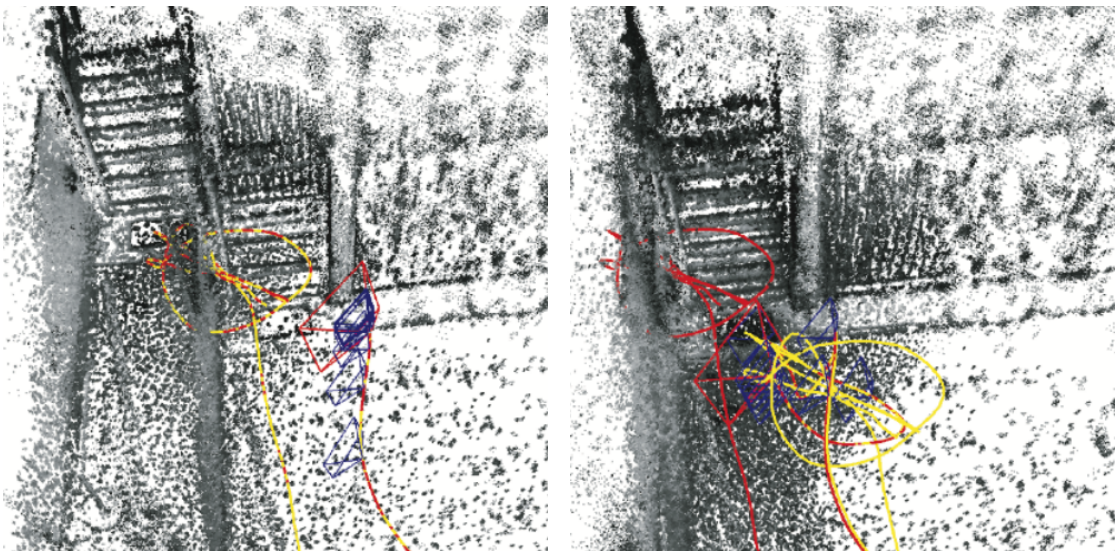
Computer vision has been an important interdisciplinary research field for over five decades now. In all this time, scientists have tried to imitate various aspects of the human visual system. Many things which are very easy to do for a human, are extraordinary hard to recreate in an artificial system. While we have made great progress in many areas and computer vision tasks are now an integral part of many seemingly intelligent systems, enabling computers to interpret visual information on the same level as even a young child still has not been accomplished [108]. Part of the reason for these struggles surely lies in the fact of limited computing power. On the other hand, whereas the growth of microprocessor performance has been exponential for as long as the computer vision field exists (and now even seems to come to an end [114]), its performance improvements did not manage to keep up with this. The main reason for this seems to be that even psychologists or neuroscientists still cannot explain how human vision in its entirety fundamentally works. How can we recreate something we do not fully understand?

Research on computer vision started in the late 1960s, probably with the so called *Summer Vision Project* at the Massachusetts Institute of Technology (MIT) in 1966. Back then, the people involved believed that enabling a computer to describe what it sees in a picture would not take more than a few months. The new direction distinguished itself from the already existing field of image processing by a desire to reconstruct structure from images, a task today called *Structure-from-Motion* (SfM). The foundations for the solution to this problem were laid in the 1970s in the form of edge extraction, optical flow and motion estimation. In addition to improving the previously mentioned tasks, in the 1980s researchers focused on studying the use of various mathematical techniques to perform image analysis. As an example, it was at that time when computer vision problems first got linked to the field of optimization. All of these methods continued to improve in the following decade and based on them, full bundle adjustment approaches were used to solve the SfM problem. At the same time, an increased interest in combining computer vision and computer graphics led to new

ideas like image-based rendering, image morphing and panoramic image stitching. In the 2000s, computer vision researchers made the existing methods more robust, while also developing novel feature-based algorithms for object, scene and location recognition [108]. The past decade showed significant advancements in all of the aforementioned subjects, due to the widespread use of machine-learning techniques. These methods were made possible by the bulk of (partially) labelled data nowadays available on the internet and of course also by the increase in computing power. Especially deep learning has shown very promising results in various categories, oftentimes surpassing the accuracy of traditional algorithms by big margins.

As we have seen, even though computer vision as a way of mimicking the human visual system is far from being solved, various highly productive research directions have been emerged from the once summer-long science project. Today, many of these topics have found their way to industry, creating innovative products and enabling new business models. Object detection and recognition remain very important tasks in computer vision with applications in multiple areas. Object detection can support doctors with image analysis and improve diagnosis. It is also being used in the production industry to quickly and reliably inspect manufactured parts. The gaming industry harnesses gesture recognition to facilitate entirely new forms of interacting with the computer. Another big application area is motion analysis and estimation: The automotive industry uses these techniques to track the eye and head movements of drivers in order to detect fatigue. The Hawk-Eye system [35] tracks the ball in sports like tennis or football to support referees in their decision-making. The problem of reconstructing three-dimensional models from images stays relevant as well, with multiple companies using cameras to accurately measure their environment or even to provide large-scale maps. There are countless more examples, as can be seen in the overview by Lowe [50].

One of the most prolific topics in computer vision is for sure the so called *Simultaneous Localization And Mapping* (SLAM) problem which stems from the original SfM task. With the heightened interest in mobile robotics and especially autonomous driving, the issue of localizing a camera in a previously unknown environment while also mapping it has become more and more important. It may seem like a chicken-and-egg problem at first but it can be solved – at least approximately – using certain initial assumptions and subsequently optimization techniques. SLAM encompasses many of the previously mentioned research directions in computer vision: It is necessary to estimate the camera motion in order to track it across frames. One also has to detect features or even objects to be able to build a meaningful map and correct the estimated camera location. Another important part of many SLAM systems is recognizing previously visited places. This challenge is called *Visual Place Recognition* (VPR). SLAM systems



(a) **(b)**

Figure 1.1.: Estimated trajectory and reconstructed map before (red line, left) and after a loop closure (yellow line, right). Clearly, the point clouds align much better in Figure (b). Source: Gao *et al.* [26, p. 2203].

inevitably introduce drift – meaning they accumulate error – in both the camera position and the map. This is due to the fact that they cannot rely on an external reference like a previously provided map or explicitly known camera poses. On the contrary, every small error in the estimation of the camera location increases the map error and vice versa. However, if the system recognizes that it is currently located at a previously visited place (commonly called *loop closure*), it can correct the drift in the map and therefore also improve the camera localization. Nowadays, most SLAM systems have a visual place recognition module running parallel to their normal operation, kicking in when a loop closure is detected and consequently adjusting the map and previously computed camera locations. An exemplary result of such a correction is shown in Figure 1.1.

At the core, visual place recognition is nothing more than a particular object recognition task. Therefore, it is tightly connected to many applications besides SLAM, for example augmented reality or image search. As diverse as its applications are also the implementations for place recognition. Many aspects of these systems can vary a lot depending on the application area and the specific goals of the overarching framework they are embedded in [53]: Images can be represented by local features, global descriptors or even a collection of high-level object models. Some systems work on single images

while others take advantage of image sequences or full videos. The used map can be represented topologically, metrically or simply be a database containing previously taken images. Sometimes it is necessary to recognize places in real-time, sometimes the task can be performed offline. In addition to the visual information, other sensor data like odometry measurements can be used to improve the results. Finally, there are generic algorithms available as well as data-driven ones specifically tailored to their tasks and environments. For each of these directions, a variety of different methods for solving the problem of place recognition is available. Benchmarking these techniques in order to detect their strengths and weaknesses is therefore an important aspect in choosing a certain system for solving the problem at hand.

1.2. Problem Statement

Based on this motivation we define the problem statement and goals of the thesis. In order to understand why specific decisions on the direction of this work have been made, it is important to know from which application area the task originally stems from. We want to investigate different methods for visually recognizing places in order to speed up the image matching process in a Structure-from-Motion framework. The system uses local features extracted from raw grayscale images to localize the camera poses and build a sparse keypoint-based map. One of the main performance drawbacks of the system is the matching phase: The features from one image must be matched to the features of all other images in order to determine the correspondences. Implementing this using a brute-force matching approach is extremely wasteful since most of the pictures are probably taken at completely different positions and viewpoints. The process can be sped up, however, by using place recognition: Image features are only matched with others if the images have a similar appearance, i.e. the images show the same location with high probability. This can vastly reduce the search space for matching.

Hence, we limit the extent of the work to the following domain: We investigate visual place recognition approaches based on local feature descriptors since most of the SLAM or SfM systems rely on them anyway and we can therefore obtain the input data for free. The next decision we make is that we do not assume the images to be in a certain order, meaning we do not look at image sequences in particular. This makes the investigated methods applicable to a wider range of tasks in contrast to limiting them to consecutive data only. Third, we do not consider any additional steps to the appearance-based place recognition procedure. The map we use should only consist of a database of images or image descriptions with no topological or metrical information. Thus, no geometric

verification on the results can be performed and we evaluate the raw image-retrieval performance. Lastly, we focus on efficient methods which are real-time capable and can be included easily into existing systems.

To summarize, we want to investigate efficient methods for visual place recognition based on pure image-retrieval using local feature descriptors enabling a faster keypoint matching process. It should be noted that the rather generic formulation of the problem statement also allows to use the results for other applications like image similarity searching. The thesis has three main goals: First, give an overview of existing, promising algorithms in this specific area of place recognition. In the wake of this, we also introduce a novel approach for highly efficient image retrieval. Next, we want to benchmark these approaches in order to carve out strengths and weaknesses. Finally, based on these findings, provide an open-source library containing different methods which is straightforward to use and highly extensible.

1.3. Outline

The remainder of this thesis is structured as follows: In Chapter 2, we give a summary of related literature in the wide field of visual place recognition and also look at some applications in research and commercial products. Chapter 3 provides rich background on several topics relevant to this thesis. Here we focus on the various forms of image descriptors, important aspects of information theory and efficient data structures with an emphasis on tree structures and hashing. Chapter 4 gives an introduction to relevant place recognition approaches, focusing on the particular methods which will later be evaluated. In order to gain necessary knowledge required to understand the fundamental ideas behind the approaches, the chapter additionally contains parts about the theory behind visual place recognition in general and a concrete variation called *(visual) Bag of Words* (BoW). Here we also introduce our novel algorithm for place recognition which is based on this bag-of-words model. In Chapter 5 we evaluate the chosen place recognition approaches. For this, we first present our benchmarking suite and define the evaluation domain. Afterwards we show the results and subsequently discuss and interpret them. Based on these findings we developed an efficient open-source library for visual place recognition. The motivation behind it and the structure of the library is explained in Chapter 6 which also contains additional analyses and evaluations. Chapter 7 concludes the thesis with a summary and possible future work.

2. Related Work

The task of visual place recognition is related to a lot of different research areas like image description, information retrieval, localization, 3D reconstruction, machine learning and many more. In order to narrow down the topics which will be considered in this chapter, it is important to reiterate the particular domain of this thesis: visual place recognition as a pure image-retrieval problem based on local features. Many of its foundations and related approaches are addressed explicitly in other parts of this thesis. Local features are introduced in Section 3.1, together with global image descriptions, data-driven and hybrid methods. Notable insights about information retrieval are given in Section 3.2. Theoretical work about visual place recognition and methods relevant to this thesis are presented in Chapter 4. The following section, therefore, does not simply repeat those aspects but rather focuses on interesting related literature which does not receive further attention in this thesis. This includes algorithms based on other features, place recognition using different sensors, content-based image retrieval and the data association task in SLAM. After that, the second part of the chapter gives an overview of popular open-source SfM and SLAM frameworks and their utilization of place recognition methods. Finally, some real-world application areas for place recognition in industry and consumer electronics are presented.

2.1. Related Literature

Image retrieval from local features is not the only way of approaching the challenge of visual place recognition. Another possibility is the use of global image descriptors which will be introduced in more detail in Section 3.1.2. To provide some examples, Ulrich and Nourbakhsh [112] represent images as color histograms and subsequently classify them using a straightforward voting strategy. Lamon *et al.* [46] combine the angular position of vertical edges and color patches in panoramic images to so called fingerprints, simplifying the challenge of place recognition to a string matching problem. Other methods use global image representations built from local features [105] or true whole-image descriptors [68] in order to recognize places. In recent years, data-driven approaches have become increasingly popular due to their success in various other

computer vision tasks. Even before the rise of deep learning, Pronobis *et al.* [81] achieved remarkable place recognition performance by employing support vector machines as a discriminative classifier for global image descriptors based on receptive field histograms. Sünderhauf *et al.* [106] use object proposal techniques and convolutional networks in order to extract expressive and robust landmark features. The current state-of-the-art method, NetVLAD by Arandjelović *et al.* [5], is a convolutional neural network specifically trained for the place recognition task. Besides these various possible input features, place recognition approaches can also include metric information (see Section 4.1) or employ probabilistic techniques [17] to improve their accuracy.

Place recognition is not only a challenge in computer vision but can also be transferred to research directions using other sensors than just cameras. For example, range finders like lidar and radar can be used for localization in depth maps or retrieval of scans in a database. This is commonly achieved by matching point features in 2D [8] or 3D [102], or even complete segments of the scans [20]. Another interesting application is the work by Ribeiro *et al.* [82] who adapt optical image retrieval techniques for acoustic pictures from forward-looking sonars in order to recognize places underwater. Lastly, Lee *et al.* [47] and Nowicki and Wietrzykowski [72] map and classify WiFi fingerprints to enable low-cost place recognition in indoor environments.

Yet another highly related research field is content-based image retrieval (CBIR). Visual place recognition as the problem of detecting previously visited places in an image database can actually be looked upon as a particular type of content-based image retrieval, namely *query by example*. However, there exist many more query schemes besides that. One of the most popular approaches is *query by keyword* which finds application in many modern search engines. An easy way to accomplish this task is by annotating the images, either manually or in some automated fashion. Jing *et al.* [42] propose a more sophisticated system where only a subset of the images is labelled and their visual features are used to propagate keywords to unlabelled images. Image databases can also be queried with sketches drawn by the user. Possible solutions to this challenging task involve classical image retrieval techniques like bag-of-features [21] or edge matching [11]. Other works, like "Sketch Me That Shoe" from Yu *et al.* [118], use deep learning to achieve superior results. Image search can also be performed interactively by users through different interfaces like concept or color maps [115].

As a last point, various aspects of the data association problem in SLAM are strongly connected to methods used in visual place recognition as well. One of these aspects is loop closure detection which typically comprises place recognition with additional verification steps. For this specific task, precision of the detection method is highly important since false positive matches can lead to unrecoverable errors in the created

map. Ho and Newman [36] employ the bag-of-words model and a visual similarity matrix to robustly identify loop closures. Angeli *et al.* [4] use the same image representation but rely on probabilistic techniques – Bayesian filtering in particular – in order to estimate the likelihood of loop closure candidates. DBoW, a place recognition approach described in-depth in Section 4.3.1, was originally developed as one part of a multi-step loop closure detection procedure [25]. The kidnapped robot problem – where a robot must relocalize itself after it was moved to a different part of the map – is another related SLAM challenge. The task can be solved using vision only but can also use other available data sources. For example, Wolf *et al.* [117] combine image retrieval with Monte-Carlo techniques in order to track a robot's pose and recover from localization failures. Chekhlov *et al.* [12] use appearance-based indexing with Haar wavelet coefficients to enable fast and reliable relocalization. More information about image matching and data association in visual SLAM can be found in the survey by Fuentes-Pacheco *et al.* [23].

2.2. Applications

Although SLAM and Structure-from-Motion can be similar in many regards, they distinguish themselves in some important aspects. SLAM systems typically operate on sequential images and aim to perform their task in real-time. Due to the strict runtime constraints, approximations are necessary and bundle adjustment is commonly performed on just a subset of all available frames. On the contrary, SfM systems usually handle unordered image collections and build their map offline with the goal of a higher reconstruction accuracy. Bundler is one of the oldest available frameworks, originating from a 2006 paper by Snavely *et al.* [101]. It enables 3D reconstruction of camera poses and sparse geometry from unordered collections of images. Bundler takes keypoints and feature matches as inputs, although the authors provide code for the matching process as well. The framework does not contain any place recognition procedure but approximates nearest neighbor search via k-d trees in order to speed up image matching. A more recent computer vision library is Theia [107]. Its goal is to provide efficient and reliable SfM algorithms with simple interfaces, few dependencies and extensive documentation. Similar to Bundler, it does not include place recognition but rather uses feature matching with optional geometric verification. Matching can be performed in an exhaustive brute-force fashion or more efficiently by the *Cascade Hashing* approach [13]. OpenMVG, presented by Moulon *et al.* [62] in 2016, is another popular open-source library for multiple view geometry. The library has been developed with the aim of supporting research with easy-to-read yet accurate implementations of common

algorithms. Among many other things, it contains methods for image description, image matching and full bundle adjustment. However, OpenMVG as well does not utilize place recognition for the matching process. Akin to the previously mentioned libraries, it matches images through brute-force or approximate nearest neighbor strategies. Finally, Colmap is a general-purpose SfM pipeline, introduced in 2016 by Schönberger and Frahm [92]. The framework can be used for scene reconstruction from ordered and unordered image collections. In contrast to the other implementations, Colmap contains a custom image retrieval procedure which was published independently [91]. The approach uses a bag-of-words representation of images through hierarchical k-means clustering and additionally employs a vote-and-verify scheme for geometric verification of images which is both accurate and fast.

As can be seen, early open-source SfM systems rarely used place recognition approaches for the image matching procedure, probably due to the fact that run time was not deemed particularly important for their application. Having said this, the developers of the most recent framework Colmap acknowledge the benefits of place recognition and thus recommend to use their vocabulary-tree matching strategy in order to improve performance when processing large image collections. Because SLAM systems have to work in real-time, they rely on efficient algorithms and so image retrieval methods proved themselves valuable very early on. ScaViSLAM is a scalable SLAM framework based on the double-window optimization scheme, introduced by Strasdat *et al.* [103] in 2011. For place recognition, the framework applies a custom bag-of-words scheme built upon the k-means++ clustering algorithm. Two of the most popular open-source SLAM systems are ORB-SLAM [65] and its successor ORB-SLAM2 [67]. As their name suggests, the libraries use ORB features (see Section 3.1.1) in order to track the camera and build a sparse map of the environment. To be able to detect loops, they use the publicly available DBoW2 bag-of-words library. A competing approach is LSD-SLAM, developed by Engel *et al.* [22] in 2014. The approach does not extract keypoints but operates directly on image intensities. Small-baseline loop closures can be determined using a similarity transform to close-by keyframes. If bigger loop closures shall be detected, the framework can optionally use keypoints and the FAB-MAP place recognition method [17]. A successor of this system, LDSO by Gao *et al.* [26], again builds upon direct feature-less techniques. In order to detect loops, additional corner points are extracted and fed into the DBoW3 image retrieval library. Many other recent SLAM frameworks – BAD SLAM by Schöps *et al.* [93], OpenVSLAM by Sumikura *et al.* [104] and Kimera-VIO by Rosinol *et al.* [83] – likewise use DBoW2 or adaptions of it.

Besides these more research-oriented SfM and SLAM frameworks, many real-word applications for consumer products are employing place recognition techniques as part of their system. A popular application area is a specific subfield of robotics, namely

mobile robotics. It is concerned with robots which move inside and interact with their environment. Examples include service robots like robotic vacuum cleaners, semi and fully autonomous cars and even robots used for planetary exploration. All of these robots need to simultaneously localize and map their environment, a task for which visual place recognition can be a corner stone. Another interesting application is augmented reality, where place recognition can be utilized to provide the user with additional information about nearby shops, sights or landmarks. Such systems are nowadays part of many intelligent assistants in smartphones or wearable computers. Lastly, image databases and search engines are frequently built upon image retrieval techniques which are highly related to visual place recognition as well. The previously mentioned query-by-example search scheme is typically used to find similar images in big databases. This is equivalent to the way many place recognition approaches based on the bag-of-words model operate.

3. Background

The previous chapter has already made clear that visual place recognition is a challenge involving a variety of different research directions. In order to understand how such a system works, how its performance can be measured and finally how it can be improved, it is crucial to have some basic knowledge about its algorithmic aspects. This chapter is therefore meant to introduce some core topics which are used by the place recognition methods we consider. First, in Section 3.1, we will show several ways to describe images, focusing on so called local features. Afterwards, we will give an introduction to the area of *information retrieval*, explaining how objects can be retrieved efficiently from a large collection given a user query. Lastly, in Section 3.3, we will talk about data structures and the concept of hashing.

3.1. Image Description

Describing images mathematically has always been a key task in computer vision, dating back to its emergence in the 1960s. After image preprocessing, which is sometimes regarded as a part of signal processing rather than computer vision, image description is the first major step in most applications. For example, visual search engines need to describe images in order to find similar pictures to a given query picture. Image or video databases use descriptors in order to generate high-level annotations for their entries. Systems based on visual SLAM or Structure-from-Motion need to extract and describe features in images to be able to build a map and track these interest points over time.

There are several dimensions which can be used to define the different image description approaches. In Chapter 5 of his book, Krig [44] provides a comprehensive taxonomy of these attributes. In the following, we will focus on the differences in density and distinguish coarsely between local features and global image descriptors. This is also the way image descriptors historically developed: Up until the late 1990s, researchers mainly focused on global image descriptors. Since then – especially because of the increase in computing performance – local features have been used more and

more frequently. These local features were initially described based on gradient orientations. Later, methods derived from intensity comparisons resulted in efficient binary descriptors. Most recently, the rise of deep neural networks in general and convolutional neural networks (CNNs) in particular led to a resurgence of global descriptors.

Because this thesis is focusing on keypoint-based visual place recognition, the following section will provide a short introduction to local image descriptors. We will describe the basic steps of feature extraction, explain how they work, and explicitly present the most important methods which are also used in the thesis. For completeness we will also introduce the essentials of global image descriptors and, finally, discuss some alternative approaches. For an in-depth survey about the mentioned as well as various other detection and description approaches, consult Chapter 6 in [44]. Another resource is the article by Garcia-Fidalgo and Ortiz [27].

3.1.1. Local Features

Feature Detection

In order to extract local features from images, two main processing steps need to be performed: Detecting keypoints and describing them. Keypoint detection aims at finding interesting locations in the image. These so called interest points can vary in their specific form, meaning they can be generated from actual points, corners, edges or bigger regions ("blobs"). There are several desired properties for keypoint detectors: They should be detected robustly, meaning that applying the algorithm to similar images yields a mostly equal set of keypoints. Also, they should be invariant to photometric and geometric changes. Photometric invariance mainly covers illumination changes, geometric invariance includes scale, rotation or even affine transformations in general. At best, they should also be easy to find and cheap to compute. We will now describe the main mathematical concepts used to detect interest points and then look at some algorithms relevant to the thesis which are using these concepts.

The *gradient* (∇f) – the first derivative between pixel intensity values – is the origin of many feature detection algorithms. One can compute its magnitude and direction. As an example, an edge can be described by having a large gradient magnitude in one direction and a small magnitude in the orthogonal direction. On the contrary, a corner corresponds to a high gradient magnitude in both directions. The second derivative of pixel values is called *Laplacian* and is mostly used as the so called Laplace operator (∇^2, Δ) which is defined as the sum of unmixed second partial derivatives. Including

the mixed parts and arranging them in a square matrix yields the so called *Hessian* (**H**). The Hessian describes the local curvature of a function of several variables. One can obtain the Laplace operator from the Hessian by calculating its trace: $\Delta f = \mathrm{tr}(\mathbf{H}(f))$. Most of the times, the Hessian is not used directly but rather some of its properties like the determinant or its eigenvalues.

One of the most well-known feature point detectors is the Harris corner detector [34]. It is based on gradient vectors from which it builds the so called structure tensor. It can detect corners, edges and flat regions based on its eigenvalues. Specifically, it uses a corner response function which is computed from the determinant and trace of the structure tensor. There have been many variations of the Harris method developed, among them the Shi–Tomasi [96] corner detector – which changed the response function in order to produce more stable corners – and the Hessian-Affine [59] corner detector for improved geometrical invariance.

Blob (or region) detectors are often based on the *Laplacian of Gaussians*. It computes the Laplacian over an image which has previously been convolved by a Gaussian kernel. Applying this Gaussian kernel at different scales yields a scale-space representation of the image. Interest points are then defined with a specific radius by strong responses resulting from the Laplace operator. An approximation of this approach can be obtained through the so called *Difference of Gaussians*. This approach detects keypoints at certain scales by simply identifying local extrema in the difference of images filtered by Gaussians. Yet another related blob detector is the *Determinant of Hessian* method which detects interest regions by finding maxima in the determinant of the Hessian matrix at multiple scales.

Finally, there also exist some methods operating on pixel values directly instead of gradients. The SUSAN approach [100] selects a set of pixels from a circular region around a given centroid. Corners and edges are detected by the ratio of pixels with similar brightness to the centroid. The Features from Accelerated Segment Test (FAST) [84] is inspired by SUSAN and detects corners by comparing the centroid brightness to the brightness of 16 pixels in a circle around it. It also includes a high-speed check to quickly exclude non-corners. Another popular variant is called Adaptive and Generic Accelerated Segment Test (AGAST) [54] which improves the performance of the accelerated segment test by combining specialized decision trees.

Feature Description

As mentioned earlier, detecting the keypoints is only the first part in the extraction of local features. In order to associate corresponding features across different images – the main challenge in image registration – it is necessary to describe the previously detected keypoints robustly. Analogous to feature detection methods, feature descriptors should be invariant to both geometric and photometric transformations, highly repeatable and as cheap to compute as possible. An additional important property is their matching performance: The descriptors should be distinctive so that corresponding features can be associated easily. They should also have a representation which allows an efficient matching procedure. Ideally, they are efficient to store as well. There are two main types of feature descriptors: Binary descriptors based on local binary patterns (LBP) and spectra methods based on more sophisticated properties like color, gradients, statistical features or histograms.

One example of spectra descriptors are orientation histograms. These descriptors calculate the gradient magnitudes and directions in a certain neighborhood around the keypoint and insert them into bins, yielding a histogram of oriented gradients. Another possible approach involves using so called *Haar-like features* [113]. Haar-like features approximate Haar wavelets by computing an average pixel value in a rectangular pixel window and comparing it to the values of adjacent regions. Specific regions can then be described by the emerging patterns of these features. Spectra methods are often highly descriptive and were therefore eagerly used in early feature extraction methods. However, they are fairly intensive to compute and match. They also consume a lot of memory because they are typically represented using floating-point values.

On the contrary, binary descriptors were developed with the goal of being efficient to compute, store and match while preserving the accuracy and robustness of spectra methods. These approaches are usually based on so called local binary patterns: binary descriptors based on simple intensity comparisons between certain pixels. One of these approaches is the Binary Robust Independent Elementary Features (BRIEF) descriptor [10]. Its simple approach is to build a 256 bit-sized descriptor out of intensity comparisons between randomly distributed point pairs in a rectangular region around the detected keypoint. This easy procedure already results in an accurate and very fast description of features. Naturally, there are also more sophisticated approaches, incorporating ideas like smoothing, learning or multiple scales to create even better descriptions.

Relevant Feature Extraction Methods

In the following, we will briefly describe some complete feature extraction methods – meaning they both detect and describe features – which find use in this thesis in one way or another. We will present them in the chronological order they were published.

One of the most popular feature extraction algorithms is called the Scale-Invariant Feature Transform (SIFT) and was first published by Lowe [51] in 1999. Interestingly, it did not find any recognition in computer vision journals initially, so the author decided to patent the approach instead (this patent expired in April 2020). However, since it got published, the SIFT paper has become one of the most cited papers in the history of computer vision research [44]. Keypoints are detected as local maxima on multiple scales of the Difference of Gaussians. Afterwards, features are described using 16 gradient orientation histograms with 8 bins each in a certain neighborhood around their position. Additionally, several measures are taken in order to achieve robustness to illumination and rotation changes. The resulting descriptor is an 128-dimensional real-valued vector.

In 2006, Bay *et al.* [7] presented an approach inspired by SIFT which they called Speeded Up Robust Features (SURF). The detection part is based on the Determinant of Hessian matrix which it approximates by exploiting integral images in order to increase efficiency. The characteristical orientation of the feature is detected by a set of Haar-like features. Finally, the descriptor is determined by the distribution of Haar-wavelet responses in a local neighborhood around the feature. By default, the algorithm results in 64-dimensional real-valued descriptors but they can be extended to 128 dimensions in order to generate more distinctive features.

One of the most popular binary descriptors is the Oriented FAST and Rotated BRIEF (ORB) algorithm which was introduced by Rublee *et al.* [85] in 2011 as a more efficient alternative to SIFT and SURF. This method enhances the FAST detector by producing multi-scale features and calculating an orientation for them. ORB then applies this orientation to a modified version of the BRIEF descriptor which makes it rotation invariant and improves the local binary pattern. The method generates 256-bit binary descriptors.

At about the same time, Leutenegger *et al.* [49] presented their feature extraction technique called Binary Robust Invariant Scalable Keypoints, in short BRISK. The detection part is based on the previously mentioned AGAST algorithm. The description of keypoints uses simple brightness comparison tests, similar to BRIEF. However, the method makes some improvements in order to achieve rotation invariance and higher descriptiveness. BRISK yields scale- and rotation-invariant 512-bit binary descriptors.

(a) Local image features **(b)** Global image descriptor

Figure 3.1.: Example images illustrating the difference between local and global descriptors. Figure (a) shows local features extracted by SURF. Figure (b) depicts how global image descriptors like Gist divide the image into individual blocks. Source: Lowry *et al.* [53, p. 4].

One year later, in 2012, Alcantarilla *et al.* [1] developed yet another feature extractor which they called KAZE. KAZE features are detected and described in a non-linear scale space by applying non-linear diffusion filtering. This method improves localization accuracy and distinctiveness at the cost of higher computational effort. The feature description is based on a modified SURF algorithm, resulting in 64-dimensional real-valued vectors. Later, the authors enhanced their approach [2] by speeding up the creation of the non-linear scale space using a novel numerical scheme. Additionally, they exchanged the spectra descriptor for a so called Modified-Local Difference Binary (M-LDB) descriptor akin to BRIEF, resulting in 488-bit binary vectors. They called the new method Accelerated KAZE (AKAZE).

3.1.2. Global Image Descriptors

In contrast to local feature descriptors, global image descriptors do not rely on previously detected keypoints but rather describe the whole image at once. This has the advantage that these descriptors can be computed very efficiently compared to feature-based approaches. At the same time, due to their holistic nature, they are more invariant to photometric transformations like illumination changes. On the other hand, local features have several advantages compared to global descriptors as well: First, features can be used for additional tasks besides simply describing images or locations. These possibilities include map making, image stitching or defining new locations in a place recognition task. They can also be combined easily with metric data in order to improve localization systems. Finally, keypoint-based techniques are less dependent on the camera pose because of their invariance property to many geometric

transformations. Global descriptors, by contrast, typically depend strongly on the viewpoint. Applying global descriptors to image segments rather than the complete image can provide a good trade-off, resulting in decent photometric and geometric invariance. An illustration of the fundamental difference between local features and global image descriptors is shown in Figure 3.1. Next, we will give a short overview of the different methods used to produce global descriptors.

One popular description technique are histograms. A histogram is a simple – sometimes approximate – representation of the distribution of numerical data. It consists of bins defining a certain range of values. Data points then fall into the bins corresponding to their value. The whole dataset is therefore described by the number of data points inside each of the bins. One example of a global image descriptor based on histograms are color histograms. Here, the bins represent the dimensions of a color space and each pixel contributes to the histogram according to its value inside this space. Another more frequently used type are gradient histograms which represent the distribution of gradient orientations in an image. A very famous example is the Histograms of Oriented Gradients (HOG) descriptor [18] which has been successfully applied for detection of people in images.

An additional widely used descriptor is called Gist, initially conceived by Oliva and Torralba [74], [75]. It is based on the fact that humans are able to capture the gist of a scene in a single glance based on a concept they call *spatial envelope*. Mathematically, a corresponding whole-image descriptor can be built using Gabor filters and principal component analysis (PCA) to reduce the dimensionality. The Gist descriptor has been extended and refined by various papers for different application areas.

One more possible method is to generate a global descriptor out of a collection of image features. The possible features can include edges, corners and color patches. This collection of features is commonly referred to as the *fingerprint* of a picture [46]. Ordering these features in a sequence (angle-based in the case of omnidirectional cameras) creates a characteristic description of an image or location and reduces the matching process to a simple string comparison.

Finally, generic mathematical tools like the discrete Fourier transform have been used to describe images. The Fourier transform can reduce the dimensionality of other global descriptors and put focus on the most expressive parts. For example, one can apply this transformation to histograms of gradients or local binary patterns and subsequently select the lower-frequency components of the power spectrum to create a short yet distinctive global descriptor. Related methods include the popular Hough transform and the previously mentioned Haar transform and Gabor filters.

3.1.3. Alternative Approaches

Besides the prototypical local keypoint-based and global image descriptors, there exists a wide variety of alternative approaches. For example, we have mentioned the trade-off of applying global descriptors to image segments before. In addition to these grid-based techniques, there have been several other hybrid methods developed. Examples include the BRIEF-Gist descriptor [105] and Whole Image SURF (WI-SURF) [6]. Bag-of-words approaches can also be considered as hybrid methods since they combine local descriptors into one global image descriptor.

In addition to hybrid methods, various other forms of image description have emerged. Milford *et al.* [61] developed RatSLAM where the description of places is based on models of the hippocampus in rodents. Other works incorporate additional sensory information like geometric data from depth sensors or 3D object models [86]. For the particular challenge of loop closure detection, sequences of images have been used to improve the description of locations [60]. Lastly, in recent years more and more data-driven image description approaches have emerged. For example, Philbin *et al.* [80] learn a non-linear transformation function for descriptors which greatly improves matching performance. Other authors propose to use CNNs for image recognition problems [97]. The NetVLAD architecture by Arandjelović *et al.* [5] produces a powerful image descriptor for the specific task of visual place recognition.

3.2. Information Retrieval

Information theory is a research field originating from the seminal paper "A Mathematical Theory of Communication" by Shannon [95], published in 1948. It studies the quantification, encoding, processing and transmission of information, where information can be defined as a mathematical quantity expressing the probability of the content of a message. It is an interdisciplinary research direction connected to mathematics, computer science, physics, engineering, and others. It has found applications in a wide variety of areas including cryptography, neurobiology, pattern recognition, signal processing, and many more.

One particular application which builds upon a lot of fundamental concepts in information theory, is information retrieval. According to Manning *et al.* [56, p. 1], information retrieval can be defined as "finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections", although noting that it "can also cover other kinds of data and information problems".

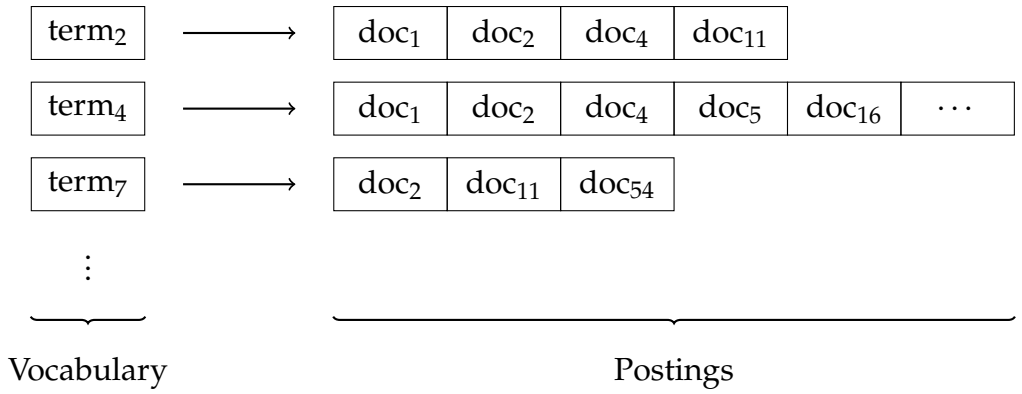


Figure 3.2.: Visualization of the concept of an inverted index. The terms on the left side each map to a list of documents (also called postings lists) in which they occur. Drawing adapted from Manning *et al.* [56, p. 6].

Recently, with the massively increasing amount of data available on the internet, information retrieval has become one of the main topics in computer science, especially as a core concept in its trending subfield *data mining*. In this thesis, we are addressing the visual place recognition task as an image retrieval problem which in turn is a specialized form of information retrieval. Especially the previously mentioned visual bag-of-words model – which will be discussed in more detail in Section 4.2 – derives its origin from the classical task of text retrieval.

In the following, we will first explain some of the terminology used in information retrieval in general and image retrieval in particular. Next, we will talk about so called weighting schemes which are used to rank certain features in the considered source of data. We will also explore some strategies for computing scores measuring the similarity of retrieved objects. Finally, we will introduce a key quantity of information theory called *entropy*, which is highly relevant to certain parts of this thesis. The book by Manning *et al.* [56] is a good reference for more detailed information. Leskovec *et al.* [48] provide a standard work on data mining which is relevant for some of the following sections as well.

3.2.1. Terminology

In order to understand how information retrieval fundamentally works, it is important to know the basic terminology used in the field, especially because many image retrieval systems borrow these terms for their own application. We will not attempt to give a complete overview but rather limit ourselves to the most important concepts necessary to understand the problem at hand:

- **Index:** The index can be defined as the key data structure used to retrieve objects based on the value of a single element or multiple of them. Objects are the matter a user of an information retrieval system is interested in. These objects are indexed with some characteristic units they contain.
- **Document:** Documents are the basic objects we aim to find in the information retrieval process. Thinking back to the definition of information, these were declared as material of unstructured nature. When talking about text, documents could be complete books, single chapters or also just individual notes. In our application the documents are images.
- **Term:** A term can now be defined generically as being the characteristic unit from which the index is built. In the case of document retrieval, these are usually words. In image retrieval a term could be a single feature or a collection thereof.
- **Collection:** The collection is the group of documents on which the image retrieval is performed. It is also commonly referred to as the *corpus*, stemming from the Latin word for body.
- **Vocabulary:** Similar to how the group of documents is called collection, the set of all terms is named vocabulary. Synonymous expressions include *dictionary* (typically referring more to the explicit data structure) and *lexicon*.
- **Query:** A query is a sequence of terms whereby the user of an information retrieval system conveys his information need. Applying this to document retrieval, a user would input a list of words and wants to receive relevant documents matching this query. In image retrieval the query could be some descriptive words or another image from which a sequence of terms is extracted.
- **Inverted Index:** The inverted index (sometimes referred to as *inverted file*) is a central concept in information retrieval. It states for each term in the vocabulary, which documents of the collection contain this term. Manning *et al.* [56] notes that the name is actually redundant because an index always maps the dictionary

to documents in which the terms occur. Nevertheless, it has become standard since it portrays the difference to a database where each document is mapped to all terms it includes. The concept is visualized in Figure 3.2.

3.2.2. Weighting Schemes

With the terminology and the ideas established in the previous section, it is now possible to construct a first simple image retrieval process: Given a user query, which we defined as a set of terms, find relevant documents containing these terms. This can be achieved easily using the inverted index by returning all documents which are mapped to any of the given query terms. The retrieved documents could then for example be ranked by the number of query terms occurring in each document. One could also combine the query terms using boolean operators (we effectively assumed an `OR` relationship before) and therefore confine the results. Accordingly, this simple procedure is called *boolean retrieval*.

This leads directly to the first of four possible weighting schemes we want to present in this section, called binary weighting. In this case, the inverted index simply stores a list of documents which can be thought of as a binary weight for each term given the document: If a term occurs in the document, its corresponding weight is 1 and it is therefore added to the inverted index. If the term does not occur in a document, we do not add it to the inverted index which implies a weight of 0.

Naturally, this binary view of terms occurring in a document or not, is not very meaningful. Let's imagine the query words being "car", "tire" and "speed", indicating that the user is interested in documents dealing with vehicles. However, when we employ a binary weighting scheme, the retrieved results would probably include a lot of unrelated documents because they contain these words as well, even if only one single time. In contrast, documents directly related to vehicles, would have a lot of occurrences of each individual term. Consequently, it makes sense to assign a weight to each term in a document, indicating its number of occurrences therein. This new weighting scheme is commonly referred to as *term frequency* and we will denote it as $\mathrm{TF}_{t,d}$, where t indicates the term and d indicates the document.

Another way to improve the expressiveness of information retrieval systems is to assign weights to each term independently of the document they occur in. As an example, if we would only be using the term frequency weighting, extremely common words like "the" or "of" would consistently get very high weights compared to other – maybe way more descriptive – words. As a result, it is common to use the *document frequency* to scale down the weight of terms based on their frequency of appearance. The document

frequency specifies the number of documents in a collection which contain a certain term. It should not be confused with the collection frequency (the total number of occurrences of a term in a collection) which is a rather poor metric in quantifying the importance of terms. The introduced weighting scheme is called *inverse document frequency* and it is typically defined as $\text{IDF}_t = \log \frac{N}{\text{DF}_t}$, where N is the number of documents in the collection and DF_t is the document frequency as defined above. A related concept are *stop words*: extremely common words which are excluded from the vocabulary entirely because they are not useful for information retrieval purposes.

The most popular weighting scheme therefore combines the term and the inverse document frequency to the *TF-IDF* weighting scheme:

$$\text{TF-IDF}_{t,d} = \text{TF}_{t,d} \times \text{IDF}_t \tag{3.1}$$

Besides this basic formulation, there exist various alternative ways of computing the term frequency as well as the inverse document frequency. There are also many different possibilities of normalizing the measures. A summary is shown in [56], Figure 6.15.

3.2.3. Similarity Measures

Up until now, we have mainly dealt with how to describe documents based on the terms they contain. We can do this by assembling each document as a vector, where each entry corresponds to the weight of the term in the document. If a term does not occur in a document, the vector entry is simply zero. In actual information retrieval implementations, these entries are obviously omitted in order to reduce memory consumption. This simple model is the basic idea of bag-of-words representations for documents.

The next question is how we can quantify the relevance of the retrieved results given a user query. A first simple idea is to add up the TF-IDF weights of all terms in the query q for each document:

$$\text{score}\,(q,d) = \sum_{t \in q} \text{TF-IDF}_{t,d} \tag{3.2}$$

This is called the *overlap score measure*. However, note that this query can be viewed analogously to the documents as a vector containing a 1 for every desired term and a 0 for all others. We can make the scoring more generic by allowing weights in the query vector as well which allows the user to rate the query terms based on importance. This

leads to a very common scoring strategy in information retrieval, namely *dot product scoring*:

$$\text{score}_{\text{DP}}(\mathbf{q}, \mathbf{d}) = \mathbf{q} \cdot \mathbf{d} \tag{3.3}$$

In the particular case of a binary query vector, this is equivalent to the overlap score.

This measure works well for collections containing documents with similar length. Unfortunately, if the length of the documents differ substantially, the dot product is usually not a good strategy for scoring. In that case, documents with a similar distribution of terms could get widely different scores simply because of their difference in length. In order to get rid of this typically undesirable property, most information retrieval systems use a different strategy called *cosine similarity*:

$$\text{score}_{\text{CS}}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{|\mathbf{q}|\,|\mathbf{d}|} \tag{3.4}$$

The denominator of this equation is the product of the Euclidean norms (L_2-norms) of the two vectors. Instead of using this equation, it is also possible to previously normalize the vectors and then calculate the dot product directly. This similarity measure is called cosine similarity because it specifies the cosine of the angle between the two vectors. It has the additional benefit of resulting in a value between 0 and 1, which makes it highly suitable for scoring.

Note that these scoring techniques allow to retrieve ranked results from queries by processing one term at a time. This is a big advantage compared to so called *document-at-a-time* approaches since the processing time grows linearly with the number of documents in the collection. In contrast, *term-at-a-time* strategies do not need to check every document explicitly but rather make direct use of the inverted index. Therefore, documents without related terms are not considered in the scoring procedure at all. Besides the dot product and cosine similarity, there are other scoring strategies which allow the term-at-a-time processing. As an example, Nister and Stewenius [71] have derived efficient scoring methods based on the L_1- and L_2-distances between vectors.

3.2.4. Entropy

Entropy is an important measure in many different science disciplines, including thermodynamics, statistical mechanics and especially information theory. It can roughly be described as the level of uncertainty or randomness in a system. In his seminal work, Shannon [95] defined information entropy as the expected value of self-information of a random variable, where the self-information is in turn the level of surprise of a particular outcome of the random variable. Information entropy can therefore be

interpreted as a measure for the average level of information of a random variable, specifically a message. Mathematically it is defined as follows:

$$\mathrm{H}(X) = \mathrm{E}[\mathrm{I}(X)] = -\sum_i \mathrm{P}(x_i) \log_b \mathrm{P}(x_i) \tag{3.5}$$

In this equation X denotes the random variable, $\mathrm{I}(X)$ its self-information and x_i the possible outcomes or events. Its unit depends on the base b of the logarithm. For the popular case of $b = 2$, the unit is called *bits*.

To give some intuition, consider a coin flip: If the coin is fair, so $\mathrm{P}(x_i) = 0.5$ for both possible outcomes, the equation results in the maximum entropy of $\mathrm{H}(X) = 1$. On the other hand, if the coin is very unfair (e.g. $\mathrm{P}(X = \text{"Heads"}) \to 1$, $\mathrm{P}(X = \text{"Tails"}) \to 0$), the entropy goes against zero. This means that the level of information you gain from one flip of a fair coin is very high because you cannot tell beforehand what the result will be. In contrast, if the coin is maximally unfair, you gain no information at all from the flip because the result is already predetermined.

Information entropy is important for many applications because it conveys the information content of a specific data source. As an example, this can be used in data compression. Think about our coin flip example before: To transmit the results of a sequence of fair coin flips, you need to send one bit (heads or tails) per throw. When using a highly unfair coin, this is not necessary because the result will always be the same. Another application area is cryptography, where it can measure the unpredictability of passwords. Finally, it can also be used in information retrieval to determine the information content of terms. We will therefore later use entropy as a measure to train our novel place recognition method.

3.3. Efficient Data Structures

Data structures are a core concept in computer science. Cormen *et al.* [14, p. 9] defines a data structure as "a way to store and organize data in order to facilitate access and modifications", while noting that there is not a single best type for all applications. There exists a wide variety of different operations on data structures – for example searching, insertion, deleting, sorting, merging, and more – and specific types of data structures work better or worse for each of them.

Data structures can coarsely be divided into two categories: Primitive data structures are the most basic form of data types and can be regarded as a building block in any programming language, directly supported at the machine level. Examples include

booleans, integers, floating-point numbers and pointers or references. Non-primitive data structures (also called composite types) are constructed from the primitive types and other structures. These can further be divided into linear and non-linear data structures. The elements in linear structures form a clear sequence, examples are fixed-size arrays or variably sized lists. Non-linear types like trees or graphs store their elements in various different manners. In addition, there are so called hash-based structures which use hash functions in order to efficiently locate stored elements.

The chosen data structures have a big influence on the performance of computer science tasks. Depending on the expected number of elements and the most frequent operations, using different structures can lead to widely different algorithmic run times. In our application of visual place recognition as an image retrieval task, for example, it is very important to choose the right type for implementing the inverted index or the vector representing documents and queries. There are also different possibilities in how to efficiently implement the transformation of local features into terms which can be derived from various data structures.

In this section we first give a brief introduction to tree structures since they are highly relevant to the place recognition approaches we consider. Then we look at hashing as an efficient way of data storing, clustering and similarity search. Finally, we provide a short overview of data structures available in the C++ standard library (STL). For extensive information about algorithms and data structures, see [14]. Additionally, the works by Knuth [43] and Leskovec *et al.* [48] are excellent resources as well.

3.3.1. Tree Structures

Tree structures can generally be defined as graphical representations of hierarchical structures. They are named after trees in nature because they bear some kind of resemblance to trees, starting from a single initial item and then progressively branching out. The concept of tree structures is not limited to a single area but rather applies to a lot of different fields. In biology, phylogenetic or evolutionary trees visualize the evolution of species. In management, organizational structures frequently have a tree-like appearance. In the following, we will consider the specific application of tree data structures in computer science.

As a first important step, we will introduce the basic terminology which is typically used when talking about trees:

- **Node:** The basic unit in a tree which could be a value, condition or a complete data structure.

- **Edge:** A connection of two nodes.
- **Root:** The initial node of a tree from which all other nodes originate from.
- **Child:** A node which is directly connected to another node while moving away from the root.
- **Parent:** The opposite of a child.
- **Leaf:** A node without children.
- **Branching factor:** The number of children of a node, also called *degree*. This factor can be constant across the tree or differ for every node.
- **Depth:** The distance (number of edges) between a node and the root.
- **Level:** The depth or the depth incremented by one, depending on whether the root is considered as level 0 or level 1.
- **Height:** Longest distance from a node to a leaf. The height of a tree is, therefore, defined as the depth of its deepest node.
- **Width:** Number of nodes in a level. The total number of leaves in a tree is called *breadth*.

In our particular application a tree is usually defined recursively, where each node is a container storing the relevant data and references to their children. There exist different kinds of trees used to efficiently store and operate on data. *Binary trees* – where each node has a maximum branching factor of two – are typically used for efficient searching and sorting. As an example, binary search trees are ordered binary trees satisfying the binary search property: The key stored in each node must be greater than or equal to any of the keys of its left sub-tree and less than or equal to any of the keys of its right sub-tree. The *B-trees* are a generalization of the binary search trees, allowing a branching factor bigger than two. They are also self-balancing, meaning they keep their height as small as possible. Another popular tree structure is called *heap*. A heap is defined as a tree satisfying the heap property (children have always either bigger or smaller values than their parent) and is therefore an efficient implementation for a priority queue. Besides those, there are a myriad of other popular tree structures for a variety of different applications like *syntax trees* for computational linguistics and *k-d trees* for space partitioning.

3.3.2. Hashing

Generally speaking, hashing denotes the process of transforming arbitrarily sized data into fixed-size values. The transformed data is usually called *key*, while the resulting value is named *hash code* or just *hash*. The function transforming the key into a hash is labelled *hash function* accordingly. The term "hashing" originates from the standard meaning of chopping food into small pieces, since a hashing function analogously slices and mixes up a key in order to create a smaller output. The use of the term dates back even to the early 1950s. A very simple first example for a hash function is the modulo operator which maps integers to a fixed range of values (depending on the divisor).

The use of hash functions is extremely common in different areas of computer science. For example, cryptography applications frequently use one-way hash functions. These are hash functions which cannot be inverted, so once a hash is generated, the original data cannot be restored. This can be used to verify the integrity of data by comparing the resulting hash code to a reference hash. Another application is password storage, where instead of storing passwords in clear-text, only their hashes are used for comparison. Hash functions are also used for hashing-based data structures, a typical example being the so called hash tables for associative arrays. Here, the hash function is used to map keys to the index of a certain bucket storing the desired value. This is especially interesting for large amounts of data, since they allow lookup (and other operations) of entries in constant time. For these applications it is desirable for hash functions to be *uniform*, meaning each value in the range of possible outputs is generated with equal probability. If certain values are generated more frequently than others, collisions – different keys resulting in the same hash code – are more likely to occur. These collisions deteriorate the efficiency of hash-based data structures because they require a linear search in addition to the constant lookup time for the bucket. Consider the extreme case, where each key maps to the exact same value: All entries would then fall into a single bucket and the lookup time would be the same as if no hashing scheme would have been used at all. On the contrary, if every key maps to a different value, no collisions occur and every lookup can indeed be performed in constant time. We then talk about a *perfect hash function*.

However, for some applications, collisions are actually a desired property. This is the case for a technique called *locality-sensitive hashing*, in short LSH. Locality-sensitive hashing was initially conceived in 1998 by Indyk and Motwani [38] as a way of approximating nearest neighbor search. The main idea is to hash the input data "in such a way that similar items are more likely to be hashed to the same bucket than

dissimilar items are" [48, p. 100]. A related approach is called *locality-preserving hashing* (LPH), which differs from LSH in being dependent on the input data. We define locality-sensitive hashing in the following way:

Definition 1 *Let $\mathcal{M} = (M, d)$ be a metric space. Next, let d_1 and d_2 be two distances according to the distance measure d with $d_1 < d_2$. Finally, let $\mathcal{F}$ be a family of functions $f : M \to S$ mapping elements from the metric space to a bucket $s \in S$. This family is called locality sensitive if for any two points $p, q \in M$ and a randomly chosen function $f \in \mathcal{F}$ the following conditions hold:*

- *If $d(p, q) \leq d_1$, then the probability of $f(p) = f(q)$ is at least p_1.*
- *If $d(p, q) \geq d_2$, then the probability of $f(p) = f(q)$ is at most p_2.*

Such a family is called (d_1, d_2, p_1, p_2)-sensitive. In order for the LSH family to be useful, it needs to satisfy $p_1 > p_2$.

LSH can be used to speed up or approximate nearest neighbor search. For example, if hashing similar items results in the same values, the nearest neighbor of a certain item can be determined quickly by only examining items which fall into the same bucket as itself. It can also be viewed as a technique for reducing the dimensionality of data. We will later apply the method as a way of data clustering. Specifically, the resulting hashes will be treated as terms in a bag-of-words model.

3.3.3. Data Structures in C++

In this last section, we want to provide a very brief overview of available data structures in C++ which are relevant for the thesis at hand. The C++ standard library provides many different containers suitable for a variety of different tasks. We will not introduce every available structure here but rather limit ourselves to containers which are finding use in one or more of the implementations of this work:

- `std::array`: Sequence container whose size is fixed at compile-time and, therefore, cannot be resized. It is the equivalent to a C-style array with added benefits of a standard container.
- `std::list`: Sequence container implementing a doubly linked list. It enables element insertion and removal in constant time. However, it does not provide fast random access of elements.

- `std::vector`: Another sequence container, this time as a dynamic array implementation. It contrast to a list, it does not allow insertion or removal in constant time but supports fast random access. It also utilizes cache well and has a low memory usage.
- `std::map`: Sorted associative container implemented as a binary search tree. Searching, removing or inserting items have logarithmic time complexity.
- `std::unordered_map`: Unordered associative container. It is implemented as a hash table which enables operations in (on average) constant time.
- `std::bitset`: Specialized container for bit sequences. The size of the bit array must be known at compile-time. It allows very efficient binary operations.

For a complete description of all available containers in C++ (we use C++17), consult the standard [39].

4. Introduction to Visual Place Recognition

Visual place recognition is a very diverse research direction with connections to many different science and application areas. The variety of available techniques which can be applied to this task therefore gives rise to a great number of methods, each with its own set of advantages and disadvantages. In this chapter, we first give a generic theoretical introduction to the challenge of visual place recognition. Afterwards, in Section 4.2, we focus the attention to a particular solution strategy called visual bag of words. In the final section, we describe two specific methods – those which are evaluated in Chapter 5 – in detail and also present a novel place recognition method inspired by the former.

4.1. Theory of Visual Place Recognition

In order to understand the concepts behind the various available place recognition approaches, it is necessary to gain some theoretical knowledge about the problem in general. The following section, therefore, aims at providing a definition of the task, describing the parts of a typical visual place recognition system and showing challenges as well as different requirements for these systems. This overview of the theory of visual place recognition loosely follows the survey by Lowry *et al.* [53], which also serves as a reference for further information.

As a first vital step in this section, we want to develop a definition of the problem at hand. We will do this by examining each part of the term ("visual", "place" and "recognition") separately. Starting from the back, *recognition* can generically be defined as the act of recognizing, where – according to [58] – recognizing means to perceive something which is previously known. In computer vision, recognition (as in *object recognition* or *scene recognition*) is typically differentiated from detection and identification tasks. Whereas detection only detects and perhaps localizes a certain thing in an image (*if* and *where*), recognition actually classifies the detection (*what*). Identification

goes another step further and determines the particular instance of an object. Place recognition, therefore, denotes the challenge of classifying places in order to realize that they have previously been perceived.

Next, we need to define what a place actually is. We can look at this term from a linguistic, a natural or a domain-specific perspective. Linguistically the word "place" cannot be defined uniquely. The Merriam-Webster dictionary, as an example, lists 12 distinct meanings for it [57]. Relevant for our application are the characterizations of a place as a "physical environment", "indefinite region" and "locality used for a special purpose", or "a particular region, center of population, or location". Still, these definitions are rather vague and so we can try to transfer a more meaningful answer from the natural world. The concept of a place has been studied in various disciplines like neuroscience or psychology for many years. Experiments on rats in mazes, dating back as far as 1948 [110], led researchers to hypothesize the existence of mental representations of the world. This so called *cognitive map* consists of places with relational edges between them. While this has been a purely theoretical construct, recording animal brain activity later resulted in the concrete detection of *place cells* and therefore solidified the previous hypotheses. These place cells fire when an animal is situated at a certain place in the environment. Similarly, researchers later discovered *head direction cells*, firing when the animal's head is oriented in a particular direction, and *grid cells*, firing in regular patterns across the environment. It has been observed that place recognition in nature arises from a complex combination of the mentioned cells, triggered by sensory and motoric stimuli. Inspired by nature, all of these concepts eventually found their way into our specific application area of robotics as well: Topological maps are frequently modeled on the cognitive map. Occupancy grid maps form a natural analogy to grid cells. Motion information can be obtained by internal odometry sensors and later combined with external sensor information to localize a robot in its environment.

The definition of a place, however, still depends highly on the navigational context of the application. In some cases, a place can be considered as a specific point or pose in space. On the other hand, a place can also be a complete region, a subset of the map. As an example, in a particular context a whole city can be specified as a place, in others as the individual buildings or as the separate rooms inside of the buildings. Therefore, each application should state their definition of a place explicitly. Visual place recognition usually defines places qualitatively: Different places should have a widely different appearance, in whatever way this may be measured.

This already constitutes the modality of our considered application. *Visual* place recognition tries to detect previously encountered places by comparing their visual appearance which is typically captured by images. Here it is important to note that vision is not the only possible source of data for recognizing places. Using absolute position and orientation sensors – global navigation satellite systems (GNSS), gyroscopes and compasses – allows to determine the exact location of an object. Place recognition can also be performed using range finders, for example lidar or radar sensors. This can be achieved by aligning point clouds with a map of the environment, a technique known as *scan matching* or *point cloud registration*. Besides these specific methods, there are countless other possibilities using a variety of different sensors to recognize places more or less accurately. We will now again get back to our original problem of visual place recognition and look at the usual structure of such a system.

Visual place recognition systems are traditionally composed of three distinct modules: image description, mapping and belief generation. Image can be described by local features, global descriptors and hybrid methods. This part has already been presented extensively in Section 3.1. The mapping module deals with the aspect of remembering previously visited places. This can be achieved using different levels of abstraction. In a pure image retrieval model the map simply consists of the stored images or image representations and is therefore strictly based on appearance. This allows highly efficient place recognition but can be inaccurate compared to other map models. Topological map models add a so called *location prior* to pure image retrieval systems. In topological maps, places are represented by nodes connected with edges which describe their relative position. This approach increases accuracy and also allows a smaller memory footprint because the location prior relaxes the accuracy requirements for the image description. Going one step further, topological-metric maps enrich the relational edges with metric data which means that edges now specify concrete distances and angles between nodes. In this case, the nodes are still defined solely based on their appearance. It is also possible to add metric information – in the form of sparse landmark maps or dense occupancy grid maps – to the nodes as well. However, this approach can hardly be considered a mere visual place recognition system anymore, rather going into the direction of full SLAM systems. Finally, the belief generation module deals with the actual decision of whether a perceived place has been visited before. When thinking about visual place recognition as an image retrieval problem, this can be achieved by the weighting and scoring schemes investigated in Section 3.2. Other methods make use of probabilistic techniques like Monte-Carlo simulation or Kalman filtering. There also exist hybrid and biologically inspired approaches, adapted from the previously mentioned place cells [61].

(a) (b) (c)

Figure 4.1.: Image pairs illustrating the problem of perceptual aliasing. Each of the three image pairs shows two distinct places despite having very similar appearance. Source: Cummins and Newman [16, p. 2045].

While the task of visual place recognition has been studied thoroughly and a wide variety of different solutions have been developed, the problem still remains challenging. The reasons for this are manifold. A main difficulty is so called *perceptual aliasing*. This term describes the effect that distinct places in the environment can have a very similar appearance, like the example images in Figure 4.1. This is a key problem which cannot be solved completely by purely appearance-based solutions, although its impact can be diminished using probabilistic techniques [16]. Conversely, identical places can look widely different due to different factors. Appearance deviations based on viewpoint or illumination changes can be solved by using invariant image descriptors. This has already been explained in Chapter 3. For other challenges like seasonal differences or occlusions, learning-based approaches are usually employed. Examples include the works by Naseer *et al.* [69] and Arandjelović *et al.* [5]. It is also important to note that these challenges can affect the whole place recognition pipeline: Image descriptions have to be largely invariant to natural appearance changes. Next, the mapping module potentially has to allow for places changing their appearance. Finally, the belief generation needs to deal with identical places looking differently and different places looking similarly.

As a last note, visual place recognition solutions can have very different requirements based on the task they are meant to achieve. For example, in an image search application, the accuracy of the retrieved results is very important. On the other hand, when place recognition is used as way to obtain loop closure candidates for a SLAM system, the results may not have to be as accurate due to additional consistency checks in later stages. Another variable requirement is the performance of the solution. Reusing our examples, image search allows longer processing times, while SLAM systems need to be real-time capable. For SLAM, knowledge about the environment can also change its requirements. If the environment is known before, image description and mapping can be tailored specifically to the operating domain. On the contrary, if the system is employed in an unknown environment, parameters of the description and mapping techniques may need to adapt or be learned.

4.2. Visual Bag of Words

In the last section, we have given an introduction to the visual place recognition problem. We have provided a generic definition of the task, described the typical structure of such systems and finally noted the main challenges they face. In this section, we now want to take a closer look at a specific implementation strategy for visual place recognition derived from text retrieval. This model is called visual bag of words or bag of visual words. We will first explain the main ideas of the model and how it got adopted from text retrieval. Next, we will describe some different ways in which particular parts of this model can be implemented. Finally, we will address some advantages and disadvantages of the model.

The bag-of-words model was originally developed for text retrieval, where the goal is to find relevant documents in a large database given a user query. The query could either just be a sequence of words or another document. The core idea of the model stems from the assumption that similar documents contain the same words or even a similar distribution of those words. A document can accordingly be represented by a vector with each element of the vector corresponding to a specific word. In case of a binary vector, the values then represent whether a word occurs at least once in the document. In case of a real-valued vector, the values count the (absolute or relative) occurrences of words in the document. Similar documents can therefore be retrieved by using binary string or histogram comparison techniques on the query and database vectors. This has already been explained in more detail in Section 3.2.

In 2003, Sivic and Zisserman [98] transferred this concept for the first time to visual data, enabling object retrieval in a movie database. The approach – later dubbed visual bag of words – closely follows the ideas from text retrieval: First, meaningful descriptors (visual words) are extracted from the images. An image can then be described and compared by the distribution of these words in the image. All of the words together build the so called *codebook* or *visual vocabulary*. Similar to information retrieval from text, words can be weighted differently based on the frequency of their occurrence in the image collection (remember the inverse document frequency from Section 3.2.2). The visual words can be different things depending on the particular implementation, although usually they are created from clustering similar local features which have previously been extracted. In this case, visual words can be thought of as characteristic structures found in images. Note how this model also maps nicely to the modules in a classical visual place recognition system: The image description is some form of hybrid approach of local and global descriptors, where the global descriptor is created from the set of local features. The mapping module is strictly appearance-based and consists of the stored image representations. The belief generation is a pure image retrieval task, using the vector space model for scoring as a probability measure.

The visual bag-of-words model can be implemented in a variety of different ways. One discerning factor of the available methods is the way in which local features are clustered into visual words. In literature, two main strategies for clustering can be found: Hierarchical (or agglomerative) clustering and algorithms based on point assignments. Hierarchical clustering methods start by assigning a separate cluster to each data point. Then they iteratively merge clusters depending on a chosen closeness measure up until a certain stopping criteria is met. This criterion can, for example, be the desired number of resulting clusters. If the number is not known beforehand, the algorithm can be stopped when inadequate clusters are generated. The inadequacy of a cluster could for example be determined by a fixed distance limit of each data point to the center of a cluster. Yet another approach would be to cluster the dataset completely – such that only one cluster containing all data points remains – and then store the tree describing the clustering process. Hierarchical clustering techniques can also differentiate themselves by the way they are measuring the closeness of clusters.

Algorithms based on point assignments basically take the other way around and start from an initial set of clusters. They then successively assign each point to the best fitting cluster. These techniques usually assume that the desired number of clusters is previously known, although there are also methods available which can deduce this number (one example being a simple trial-and-error procedure). One of the most well-known point assignment algorithms is called *k-means*. K-means first initializes a set of k clusters from a random subset of data points and then assigns the other points based

on their distance to the clusters. The cluster centroids are subsequently recalculated and all data points reassigned. This procedure continues until the algorithm converges. Since the method's success is highly dependent on the initial choice of clusters, different initialization strategies have been developed. As an example, *k-means++* only picks the first cluster centroid randomly and initializes the other clusters with a probability proportional to their squared distance to the nearest already existing cluster. Variations of this technique include *k-medians* or *k-modes* which turn away from the use of the mean as the characteristic cluster center.

We have now explained how the visual vocabulary can be built using different clustering techniques. However, this is not the only way in which visual bag-of-words implementations differ. One challenge, which many visual bag-of-words methods face, is the number of words necessary to build an expressive visual vocabulary. Local descriptors are rather high-dimensional and so a large vocabulary – ranging from thousands to hundreds of thousands of terms – is usually needed in order to generate distinctive image representations. This fact exacerbates the generation of the vocabulary via clustering and also the transformation of features into words. To increase efficiency, many methods use a hierarchical structure for the vocabulary. For example, the vocabulary can be created as a tree structure where each node represents a cluster. The complete set of terms is then defined by the leaves of the tree. One particular method employing such a hierarchical bag-of-words scheme is the popular Fast Appearance-Based Mapping (FAB-MAP) approach by Cummins and Newman [17]. Another distinguishing factor is the weighting of terms in the vocabulary. Apart from the strategies we have already investigated – binary, term frequency, inverse document frequency and their combinations – words can also be weighted by their importance for certain applications like loop closure detection [55]. Finally, instead of the default way of building a vocabulary from training data in advance, there are also methods available to incrementally create the vocabulary while the system is running. As an example, Nicosevici and Garcia [70] are using a hierarchical clustering technique in order to generate terms step-by-step for their so called Online Visual Vocabulary (OVV).

Bag-of-words models have many advantages compared to other visual place recognition approaches. First, the fact that they ignore the geometrical distribution of their input data (the local features) makes them implicitly invariant to the camera pose from which images where taken. This is a desired property for every place recognition task. Next, thanks to the use of an inverted index and potentially hierarchical structures, they are highly efficient. This is especially true in comparison to methods based on raw feature matching and more complicated topological-metric maps. Finally, their simple architecture allows them to be incorporated into other complex frameworks (for example SLAM systems) rather easily. On the other hand, they also have some

significant weak spots. While their dismissal of geometrical structure makes them pose-invariant, it can also decrease the accuracy of place recognition. This can be illustrated by an example from text retrieval: For bag-of-words approaches, the sentence "Lena likes cats more than dogs" is identical to "Lena likes dogs more than cats", despite the obvious semantic contradiction. Analogously, ignoring the geometrical structure of image features can lead to incorrectly recognized places. Another disadvantage is the usually rather expensive initial vocabulary generation and the danger of introducing a bias if the training data is not generic enough for the task. These concerns can be reduced by building the vocabulary online, although this can also decrease the performance and limit the accuracy due to the required vocabulary modifications.

As a last point, it is important to note that the usage of the bag-of-words model is not limited to text or image retrieval tasks. On the contrary, the technique has already been employed in a variety of different fields. Pancoast and Akbacak [77] have used a bag-of-audio-words approach for multimedia event classification. Similarly, Jin *et al.* [41] recognize emotion from speech using acoustic and lexical words. González *et al.* [29] extract words in accelerometer readings from smartphones in order to classify roadway surface disruptions. Ofli *et al.* [73] even adapt the model for multiple different sensor modalities including motion capturing systems, RGB-D cameras and more.

4.3. Methods for Visual Place Recognition

This as well as the previous chapter have shown that the task of visual place recognition cannot be reduced to one particular implementation or even implementation strategy. We have already seen that methods can be based on global image descriptors like BRIEF-Gist [105] or WI-SIFT [6]. Besides these, a large quantity of VPR systems uses local features or bag-of-words approaches built from these features [17], [55], [70]. Yet other methods combine these different techniques, see for example the work by Wang and Yagi [116]. Finally, there are many SLAM frameworks embedding various different place recognition methods. Examples include the previously mentioned RatSLAM [61] and SeqSLAM by Milford and Wyeth [60].

In this next section, we want to direct our attention to a set of place recognition methods with the very specific properties we have already carved out in the problem statement in Section 1.2. We want to examine approaches which can be used to speed up loop closure detection and matching in a SLAM or Structure-from-Motion system. Therefore, they shall be pure image-retrieval techniques with no additionally required mapping or consistency checking features. Such capabilities can optionally be

provided by the framework they are embedded in. The methods shall also be based on previously detected local features, since these are often implicitly available in such systems. Naturally, they also have to be able to run in real-time.

Stemming from these requirements, we have chosen two state-of-the-art methods which are publicly available. The first one, originally developed by Gálvez-López and Tardós [25], is called *DBoW* and follows a hierarchical bag-of-words scheme. The second one, *HBST* by Schlegel and Grisetti [89], constructs a binary search tree in order to speed up descriptor matching and enable place recognition. In the following, we will give a detailed introduction to both of them. Partially inspired by ideas from these methods, we will also present a novel visual place recognition approach, a hashing-based bag-of-words strategy we call HashBoW. The methods will be evaluated thoroughly in Chapter 5.

4.3.1. DBoW: Hierarchical Bag of Words

DBoW was officially presented as part of a bigger loop closure detection system in the paper "Bags of Binary Words for Fast Place Recognition in Image Sequences" by Gálvez-López and Tardós in 2012. However, they first described the method in another paper one year earlier [24]. The main novelty of DBoW is the use of binary descriptors in a bag-of-words approach. In order to achieve this, they generate a "vocabulary tree that discretizes a binary descriptor space" [25, p. 1188]. For the particular use case of loop closure detection in mobile robotics, their complete system additionally includes a grouping strategy for images close in time as well as temporal and geometrical consistency checks.

The use of binary features for place recognition is motivated by their advantages compared to real-valued spectra descriptors. Real-valued features like SIFT or SURF are rather expensive to extract and match. Binary features on the other hand, can be detected and described quickly by using simple intensity comparisons. They can also be matched more efficiently by computing the Hamming distance instead of the Euclidean distance. This has already been explained in more detail in Section 3.1. In their paper, Gálvez-López and Tardós therefore chose to use FAST keypoints combined with the BRIEF descriptor, although the general approach is independent of the specific feature type.

Visual bag-of-words approaches build a vocabulary by discretizing the descriptor space into a set of characteristic visual words. In the case of DBoW, the vocabulary is structured in a hierarchical way as a tree. It is generated offline by a large amount of training images which should be independent of the ones used in operation in

order to prevent overfitting. The vocabulary generation proceeds as follows: First, the descriptors are clustered into k distinct groups using k-medians clustering in combination with the k-means++ initialization strategy. The algorithm is then repeated for each of the groups separately, resulting in another level of clusters. This iterative clustering approach continues until the tree reaches a predetermined depth level L or until there are no descriptors left. The resulting vocabulary has a maximum size of $W = k^L$ terms which subsequently get weighted according to their frequency of appearance in the training data. DBoW employs the previously discussed inverse document frequency (see Section 3.2) for this. Each node in the final vocabulary tree stores a characteristic descriptor, representing the centroid of the cluster. A new descriptor can then be transformed into a visual word by traversing down the tree: At each level, the descriptor simply chooses the node with the minimum Hamming distance to its centroid. Finally, an image can be represented by the bag-of-words vector, where each entry states the TF-IDF value of the corresponding word. This TF-IDF value is the combined value of the word weight and occurrence frequency of the term in the image.

The database of DBoW consists of two separate indices: the well-known inverse index and a novel direct index. For each word, the inverted index stores a list of images augmented with the TF-IDF value. The most similar image to a given query can then be retrieved rapidly by calculating a score based on the L_1 distance. Additionally, DBoW stores the features of each image in a direct index which can later be used to speed up the geometric consistency check. For each image, the method determines which features correspond to the same word or the same node at a certain level. These feature lists, together with the image they originate from, are stored in the direct index. Later, when a loop closure candidate is getting verified geometrically, only the descriptors belonging to the same word or node have to be considered as potential correspondences. This type of approximate nearest neighbor search increases the efficiency of the matching process.

The complete loop closure detection procedure described in the paper works in the following way: Loop closure candidates are acquired by querying the database using the inverted index. Since the calculated score can vary a lot due to different word distributions of the query images, it gets normalized by an expected score. This best possible score is determined by the similarity score of the query and the image taken right before. Next, a match grouping is performed. Returned candidates close in time are grouped into so called islands and treated as a single match. An island's score is the combined score of all matches it contains which means that longer sequences are favored. The best matching island undergoes a temporal consistency check, verifying that the matches are consistent with previous queries. Lastly, the best candidate in

the island is verified by its geometrical consistency. Therefore, a fundamental matrix between the loop closure candidates is computed with RANSAC. In order to make this procedure efficient, the previously mentioned direct index is used to approximate and thus speed up the search for corresponding features.

Gálvez-López and Tardós evaluate their whole pipeline on five different indoor and outdoor datasets. They manually create the ground truth information on loop closures for those datasets which do not contain such information. As a correctness measure they apply the well-known precision-recall metric. They do not tune the parameter settings to each dataset but rather use a subset as training data and the others for final evaluation. The vocabulary is generated from yet another collection of images. The results show that BRIEF descriptors detect loops almost as reliable as SURF features while being at least an order of magnitude faster and more memory efficient. Using more invariant binary descriptors like ORB or BRISK could improve the results even more. They also showed that the temporal as well as the geometrical consistency checks are valuable additions when tuned properly. Measurements of the execution time confirm that the approach is faster than several other state-of-the-art methods for loop closure detection. Comparing it to the popular FAB-MAP 2.0 method proved an order of magnitude smaller execution time with only slightly worse results.

The authors open-sourced their full loop closure detection system, calling it DLoopDetector. They also released the image retrieval part of the pipeline – DBoW – separately as a stand-alone library. Some time later they presented an improved version, DBoW2, which allows the use of arbitrary descriptors (not just binary ones). In 2016, Muñoz-Salinas released DBoW3 [63]. This version simplifies the interface, provides out-of-the-box compatibility to any type of descriptors extracted by OpenCV [9] and introduces some speed optimizations. In addition, a highly optimized version of just the vocabulary tree (FBOW) was later published as part of the UcoSLAM library [64]. Today, depending on the use of OpenCV, DBoW2 and DBoW3 are the most commonly used implementations.

4.3.2. HBST: Hamming Distance Embedding Binary Search Tree

Stemming from a very similar background as this thesis – Visual SLAM using local features – Schlegel and Grisetti developed a novel place recognition approach called HBST. They officially presented the method in their 2018 paper "HBST: A Hamming Distance Embedding Binary Search Tree for Feature-Based Visual Place Recognition," although the core concept was already established two years prior [88]. The goal of their work is to provide a robust yet efficient solution for visual place recognition by enabling

fast image retrieval with feature correspondences for consecutive descriptor matching. They limit the approach to binary descriptors due to their high performance in the extraction and matching process. The main idea of HBST is to approximate brute-force descriptor matching between images based on Hamming distance by constructing a binary search tree. An additional voting scheme "enables fast and robust image retrieval" [89, p. 3743].

The key component of HBST is the tree construction. The tree is a binary search tree, i.e. every node has exactly two children. Each node stores a certain descriptor index representing the splitting criterion: An input descriptor is stored in the left or the right subtree depending on its bit value at the given index. The index values are therefore in the range of the descriptor dimension. An additional requirement is that a given index may only appear once in any given path of the tree. This also limits the maximum depth of the tree to the dimension of the descriptors. The leaves of the tree thus partition the set of all descriptors and each feature corresponds to exactly one leaf.

This tree structure allows for efficient descriptor matching by approximating the nearest neighbor search. A query descriptor traverses down the tree based on the splitting criterion at each node. Once the descriptor reaches a leaf, brute-force search can be performed on this subset of all available descriptors. In terms of complexity, given that we have a total set of N descriptors, a simple brute-force search for the nearest neighbor would require $\mathcal{O}(N)$ operations. Arranging the descriptors in a balanced binary tree of depth h results in a decreased search time complexity of $\mathcal{O}(h + \frac{N}{2^h})$. Because of the approximative nature of the approach, the *correctness* of the result (finding the most similar descriptor) can only be guaranteed if the exact same descriptor is already part of the tree. Also, the search is not guaranteed to be *complete* if the goal is to find all descriptors inside a given distance threshold.

It is important to note that if the descriptor search inside a threshold would be complete, it would also be correct due to the following brute-force search. Schlegel and Grisetti therefore analyzed the mean completeness of the search for different descriptors, bit indices, distance thresholds and tree depths. The results are as follows: The choice of the specific bit index at a given node hardly influences the completeness of the search. On the other hand, greater distance thresholds result in lower mean completeness. This makes intuitive sense because of the lower probability that all of the corresponding descriptors fall to the exactly same leaf. An even bigger impact has the increase in tree depth which results in an exponential decrease of the mean completeness.

These analyses result in a trade-off between search completeness (and thus correctness) and search time, depending on the chosen tree depth. However, this is only true for balanced trees where each leaf stores approximately the same number of descriptors.

HBST therefore applies a simple metric in order to generate a balanced tree: At each node, choose the bit index which splits the set of input descriptors in half as good as possible, i.e. whose mean bit value among all descriptors is the closest to 0.5. This simple approach leads to a well-balanced tree. Yet, with a growing number of images and thus descriptors to insert, the construction of such a balanced tree from scratch quickly becomes infeasible to do in real-time. Because of this, the authors developed a procedure to build the tree in an incremental fashion. The idea is rather simple: The tree starts with just a single split and thus two leaves. Descriptors are accumulated in the leafs until they reach a certain number of descriptors. At that time, the leaf splits according to the previously described metric and therefore becomes a node. This algorithm does not guarantee a perfectly balanced tree but experiments show that it successfully limits the tree depth. As a last step of the method, each stored descriptor is augmented with the image index it originates from. This enables image retrieval by using a simple voting scheme at no additional cost.

The paper concludes with a rich evaluation about the image retrieval accuracy in a loop closure application with sequentially acquired images. They compare six different approaches: Brute-force, LSH-based approximate nearest neighbors (FLANN-LSH), DBoW2 using the image score, DBoW2 using a score based on the direct index and HBST in two different parameterizations. The evaluation is performed on four publicly available datasets with manually computed ground truth. They employ the F_1-score – representing the best trade-off between precision and recall – and the run time as evaluation measures. The reported results suggest that DBoW using direct indexing results in the best accuracy after brute-force matching, with the HBST approaches as close runner-ups, while FLANN-LSH and score-only DBoW largely fail. Regarding run time, the HBST approaches are considerably faster than any other method, with score-only DBoW2 being the runner-up. The evaluation concludes that HBST achieves comparable accuracy to state-of-the-art methods at significantly higher computational speed.

The authors provide their implementation as a C++ header-only library with additional OpenCV wrapper functions. Furthermore, they open-sourced the ground-truth calculation toolkit as well as the benchmarking application used for the evaluation.

4.3.3. HashBoW: Hashing-Based Bag of Words

As we have seen from the theory and some specific methods, bag-of-words schemes are a popular and efficient way for retrieving images and consequently visual place recognition. Their biggest expense lies in the transformation of local features into

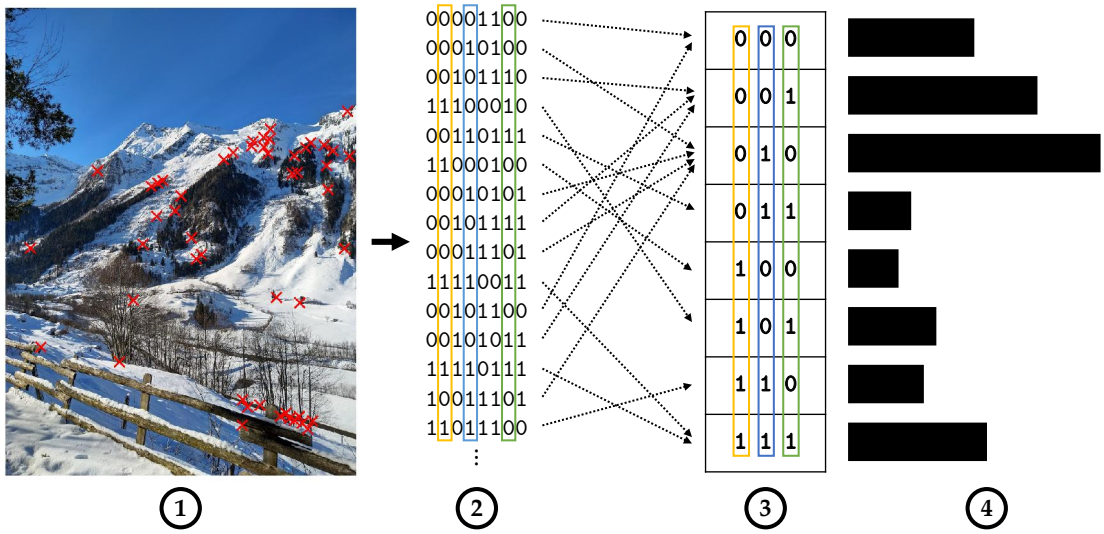


Figure 4.2.: Generation of the vector space representation of an image using HashBoW. After local features got extracted (1), the descriptors (2) get hashed based on their bit values at defined indices. The resulting hash codes (3) represent the terms of the bag-of-words model. An image can then be characterized by its distribution of these words (4).

visual words, for example by clustering. On the other hand, we have also seen that simply arranging binary descriptors in a tree structure based on their bit index can lead to good image retrieval results as well. Finally, Section 3.3.2 among other things introduced the concept of locality-sensitive hashing which can be interpreted as a form of dimensionality reduction or clustering. Inspired by these insights, we now present HashBoW, a novel image retrieval method based on the bag-of-words approach and locality-sensitive hashing.

The main idea of HashBoW is rather straightforward: We want to to discretize the descriptor space using hashing and subsequently treat those hashes as terms in a bag-of-words scheme. As a first step, we will focus on binary descriptors only. This is enough to understand the core concept which can easily be extended to real-valued features as well. Locality-sensitive hashing is always defined given a metric space, that is to say a set with a distance metric on it. In the case of binary descriptors, the metric space is the set of d-dimensional binary vectors together with the hamming distance. For this metric space, a simple LSH family can be defined by a process called *bit sampling*:

Hash	Number of descriptors
00	5
01	25
10	25
11	5

Table 4.1.: Exemplary distribution of 60 descriptors over 2-bit hash codes. Although each index of the hash code independently gives rise to an equal number of descriptors (30), the overall entropy is not maximized because of the uneven distribution.

Definition 2 *Let $p \in \{0,1\}^d$ be a d-dimensional binary vector in the metric hamming space. Then, for any distance threshold $r \geq 0$ and approximation factor $c \geq 1$, the family*

$$\mathcal{H} = \left\{ h\colon \{0,1\}^d \to \{0,1\} \mid h(p) = p_i,\ i = 1 \dots d \right\} \tag{4.1}$$

is $\left(r, cr, 1 - \frac{r}{d}, 1 - \frac{cr}{d}\right)$-sensitive (compare Indyk and Motwani [38]).

As long as the approximation factor c is greater than 1, this also implies $p_1 > p_2$ and therefore the family $\mathcal{H}$ is useful according to Definition 1 from Section 3.3.2. This means that choosing a random bit from a binary descriptor is a valid locality-sensitive hash function.

In order to make the hashing function more expressive, we use multiple of these functions and concatenate them. So, if we want to hash a binary descriptor into a n-dimensional vector, the hashing function $g\colon \{0,1\}^d \to \{0,1\}^n$ is defined as follows:

$$g(p) = (h_1(p), h_2(p), \dots, h_i(p), \dots, h_n(p)) \tag{4.2}$$

Here, each h_i is a locality-sensitive hash function according to the above Equation 4.1. In our approach, the set of all possible concatenated hash codes is a term in the vocabulary. The size of this vocabulary, N_{voc}, is therefore limited by the dimensionality of the hashes: $N_{voc} = 2^n$.

The place recognition procedure then works as follows: First, extract local features from images. In the next step, transform the descriptors into visual words using the hash function. Each image can then be described by a vector representing the distribution of these words (going by the term frequency). This strategy is visualized in Figure 4.2. The database consists of an inverted index, enabling fast similarity search. The method can be parameterized by changing the hash code dimensionality n, and the choice and order of the locality-sensitive functions h_i.

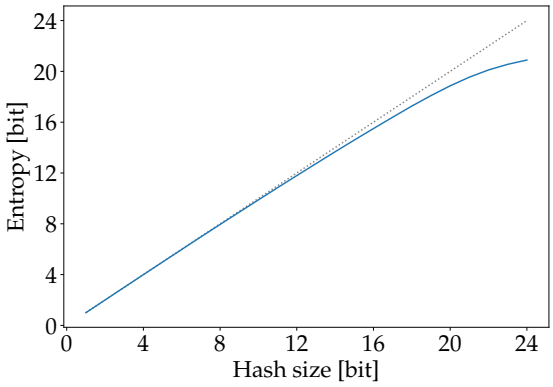


Figure 4.3.: Entropy of the hash codes after training HashBoW on a dataset with 1491 images. The dotted gray line denotes the theoretical ideal behavior. It can be seen that even with this relatively small dataset, the training procedure results in near-optimal hash codes up until their size gets very big.

In order to improve the image retrieval performance or enable it in the first place, bag-of-words schemes usually incorporate a training process. For example, DBoW uses a collection of training data to cluster the descriptors and build up its vocabulary. In our case, choosing random locality-sensitive functions already provides an initial vocabulary. Still there are many conceivable strategies to improve the performance of the method, including the use of the terms' inverse document frequency or locality-preserving hashing. However, in this work we decided to take a slightly different approach, using the fundamental measure in information theory – entropy – in order to create a meaningful vocabulary. For the vocabulary to be expressive, it should discretize the descriptor space as evenly as possible. If, on the contrary, many different descriptors correspond to the same words, the image retrieval performance diminishes since the vocabulary is not discriminative enough. As an example, think about the case of a single bit index: In the extreme case that the bit value of this index is always 0, using this bit for the hashing function would provide no information about the descriptor at all. This fact can be measured by calculating the entropy of the resulting hash codes.

We, therefore, propose an entropy-based training process in order to choose a combination of bit indices, maximizing the entropy on a given training set. For each dimension of the locality-sensitive hashing function g, determine the bit index which distributes the set of descriptors as equal as possible, that is to say it maximizes the entropy. Note that it is not possible to do this for each dimension independently by simply choosing the n best-dividing indices. This is due to the fact that these indices could potentially be correlated and thus diminish the entropy of the resulting hashes. An example of

such a situation is given in Table 4.1. Hence, after a certain bit index is chosen, it needs to be fixed and the entropy must be maximized in correspondence to the already determined indices. Although this greedy approach is not guaranteed to result in the optimal solution as well, experiments show that the resulting entropy is very close to the maximum possible value (see Figure 4.3).

HashBoW has been implemented in C++ and we provide it as part of the open-source library which will be introduced in detail in Chapter 6. So far, the implementation is limited to binary descriptors. However, it can easily be extended to real-valued descriptors by providing a fitting locality-sensitive hash function or embedding the real-valued vectors into hamming space. The approach will be thoroughly evaluated in the following chapter, together with DBoW and HBST.

5. Evaluation of Visual Place Recognition Approaches

So far, we have given a literature overview and provided extensive background information necessary to understand the different aspects of visual place recognition based on local features using pure image retrieval. In the last chapter we explained the essential theory behind this particular task and described three promising state-of-the-art implementations: DBoW, HBST and HashBoW.

In this next chapter we now want to evaluate the performance of those methods and investigate their robustness to various changes like feature extractor types or parameter settings. First, we will introduce a new, easy to use and highly extensible benchmarking suite we have developed. The second section describes the contents of the evaluation, specifically looking at the considered methods and their parameters, feature extractors, datasets and employed metrics. At third and last, we will present and interpret the results of our performed evaluation.

5.1. A Benchmarking Suite for Visual Place Recognition

In this section, we will present our newly developed benchmarking suite for the evaluation of visual place recognition methods. We will first explain the importance of benchmarks and discuss weak points of currently available evaluation frameworks. Then we will explain the structure of the library and finally show its typical setup and workflow. The project is available in an online repository, as specified in Appendix A.

5.1.1. Importance and Availability of Benchmarks

Evaluation is one of the most important parts in scientific research, especially when developing new solutions to existing problems. Without thorough and impartial evaluation, the value of new contributions cannot properly be assessed. It is also important to illustrate advantages and disadvantages given certain conditions in order

to gauge the applicability of a method to particular tasks. Consequently, it is common practice for scientific publications to include a section where the newly presented approach is evaluated. The quality of these analyzes, however, can vary substantially.

This is not any different in the field of visual place recognition. In the course of this thesis we have already portrayed the wide variety of methods suitable for different application areas a number of times. These methods usually get evaluated as part of the publications by their respective authors. Typically, these separate evaluations are performed rigorously and with best intentions. Nevertheless, they frequently have major problems which greatly diminish their value. First, evaluation metrics – the measurements used to judge methods – can differ between author groups which makes it hard to compare various techniques. This is especially true if the methods of interest were not even included in the separate evaluations. Evaluations can also be performed on different datasets which complicates the assessment of applicability for differing situations. Next, while the test methodology is regularly described in detail, crucial information like exact parameter choices of the compared approaches is often missing. In combination with the unavailability of the utilized evaluation framework, this fact impedes reproducibility of the reported results. Lastly, the generated findings are usually only valid for the specific application area the authors decided to consider. It can therefore be difficult to choose which particular technique to use for a given problem.

Publicly available benchmarking suites can be a possible solution to these problems, especially if their implementation is open-source. Reporting results based on such benchmarks provides a fair comparison of different approaches and makes it easy to reproduce the findings. They allow researchers to quickly evaluate and tune newly developed techniques. When users have access to the concrete implementation, they can also add new datasets or metrics they are interested in. This simplifies the decision-making process for choosing a certain solution. Unfortunately, at the moment there are only very few such benchmarks available for the task of visual place recognition. One current example is the Segway DRIVE Benchmark, published by Huai *et al.* [37] in 2019. They present a new dataset consisting of images, odometry data and pose information, captured by a fleet of mobile delivery robots. The authors also provide several metrics for evaluating visual-inertial odometry, SLAM and place recognition algorithms. However, their focus certainly lies on the rich dataset and the extensive theoretical part about the proposed metrics. In contrast, their released tool box (a set of python scripts) is rather limited and tied to a specific mapping framework [90]. A more comprehensive benchmarking suite was released by Schlegel and Grisetti as part of their HBST publication. They provide a full-fledged application for evaluating visual place recognition methods which is completely open-source. Out of the box,

the application is compatible with twelve popular datasets, eight feature descriptor types and six different place recognition methods. For performance comparisons, they employ the precision-recall metric and additionally measure the run time of each method. However, despite all of these positives, the benchmark also has some major shortcomings. First of all, their application is explicitly tailored to loop closure detection and therefore assumes image sequences, which makes it impractical for generic image retrieval tasks. Another weak point is that the whole pipeline – dataset preparation, image processing and evaluation – is combined into one single application written in C++. This has multiple disadvantages: Because each dataset has a different format, the program needs to handle each of them separately. This makes its code rather complex and, therefore, difficult to read and adapt. Additionally, every time something changes – even if it is just the parameterization of some method – the whole pipeline needs to run again. This wastes time because many steps are repeated unnecessarily. As a last point of criticism, while the usage of the benchmark is documented very well, the code for the most part is not. Hence it is rather hard to add new datasets or evaluation metrics.

Motivated by the need for thorough, unbiased evaluation and inspired by available solutions and their shortcomings, we thus decided to develop a new open-source benchmarking suite for visual place recognition. Our goal is to provide a stand-alone tool box which is not tied to a certain framework or application. Its usage as well as the code is well documented in order to make it easily usable and adaptable. Its structure is highly modular such that individual parts can be modified or exchanged without altering the remaining pipeline. This also allows to utilize different programming languages for different tasks, exploiting their strengths in specific domains. The resulting benchmarking suite is fast and efficient as well as easy to use, maintain, adapt and extend. In the next section we will describe its structure and justify our design decisions in detail.

5.1.2. Library Structure

The benchmarking library is structured into three main components: data preparation, data processing and evaluation. Each component can be used separately from the others which allows quick adaptions and rapid prototyping. Only the interfaces for the whole pipeline – expected inputs and outputs for each part – are defined such that new implementations can easily be included in the complete pipeline. In accordance with our considered task, the implemented pipeline is meant to evaluate pure image retrieval systems. In case that someone is interested in a slightly different problem, each component was written in such a way that many of its functions can be reused.

An example could be the evaluation of more complete loop closure detection systems like the previously mentioned DLoopDetector. However, in light of the specific task at hand, the following paragraphs will focus solely on the description of the pipeline which is currently available.

Data Preparation

The data preparation component transforms datasets into a unified format such that the processing part can make use of it. This enables the addition of a wide variety of datasets without the need to handle each of the different formats separately in later stages of the pipeline. New sets of data can either be transformed by hand in the case of prototyping, or by providing a conversion script for other users who do not need to gain deeper insight into the particular format. The defined output format is a directory with the following contents:

- A flat directory called `images` containing the images of the dataset.
- A file called `query_list.yaml` specifying the subset of images which are used to query the database. The subset is defined by a list of the file name stems of the image files.
- A file called `ground_truth.yaml` specifying the ground truth for the image retrieval process. The ground truth is defined by an associative array with the query names as keys and lists of corresponding image names as values.

As a way to exchange information between the different components, the data serialization language YAML (YAML Ain't Markup Language) was chosen. It has the advantage of being lightweight, human-readable and supported by libraries in various programming languages. The currently available conversion scripts are written in Python 3, together with the PyYAML library. Python offers a rich standard library, is easy to read and write and there is no compilation necessary which simplifies the usage of the scripts. Note again that it is not a strict requirement to use Python for preparing the input data. As mentioned before, only the resulting output format is fixed in order to make it compatible with the remaining pipeline. How to generate this data is up to the developer.

The released scripts are able to download the currently supported datasets and convert them into the specified format. Generic functions like downloading data, extracting archives or generating the YAML files are outsourced into helper files such that they can be reused when adding new datasets. For those datasets which rate the quality of the matches, an optional command-line option can be used to declare the minimum

quality requirement. At the time of publication, there are setup scripts for three datasets available. We additionally provide a routine which can merge multiple data directories in order to create bigger, more diverse datasets.

Data Processing

The actual place recognition methods are part of the data processing component. The procedure is implemented as a program written in C++ – using the latest standard C++17 – which is a flexible and highly efficient programming language. Because of its popularity for performance-critical applications (including computer vision), there exists a rich landscape of third-party libraries for a wide variety of tasks. Besides the previously presented place recognition algorithms (DBoW, HBST and HashBoW), compiling the code requires three external dependencies. First, we use the header-only library CLI11 [94] to offer a descriptive and easy-to-use command-line interface for parameterizing the process. Second, we employ a subset of the OpenCV library [9] – the components `core`, `features2d`, `imgcodecs` and `imgproc` – for local feature extraction and YAML file processing. Finally, the serialization library Cereal [31] is used to store intermediate results. For example, when comparing different place recognition methods, local features do not have to be extracted every time. We can use Cereal to store the descriptors after the first run of the program and subsequently reuse them. This speeds up the data processing.

The command-line interface requires the user to specify a valid (following the previously defined format) input directory and the desired place recognition algorithm. Optionally, the user can define a path to a serialized feature database, the number of retrieved results per query, a path to a file stating the methods' parameters, and the desired output directory. Once again, the method parameters are set using a YAML file such that the program does not have to be recompiled for every settings change. A recompilation is only necessary when switching to a different feature extractor because some place recognition methods need to know the descriptor dimension at compile-time. The feature type can be specified via CMake which is also used to manage the complete build process.

For implementing the concrete functionality, we again employ a modular structure and encapsulate it into two libraries that can be reused in case a new pipeline shall be added. The first library contains helper functions regarding input and output processing (YAML parsing and writing), image processing (feature extraction), place recognition (method initialization and usage), serialization and additional utilities. The second library implements the available place recognition methods. For that we

created an abstract base class which serves as an interface definition and contains some low-level functionalities like run-time measurement. Every method needs to inherit from this base class and has to implement the pure virtual declared functions (`addImage`, `trainMethod` and `queryDatabase`). This design concept establishes a fixed interface such that new methods can be added to the place recognition process with only minimal changes to the existing code base. Furthermore, the library handles the parameterization of the methods using YAML.

Finally, the output of the place recognition process is again defined as a directory containing the following files:

- A file called `settings.yaml` which contains information about the used dataset, the number of retrieved results and the place recognition method including its parameters.
- A file called `timeResults.yaml` containing time measurements of each phase (add, train, query) of the procedure.
- A file called `vprResults.yaml` which reports the retrieved results for each query including their scores.

Thanks to the settings file, each result directory specifies precisely which data, methods and parameters were used in this processing instance. This helps to prevent accidental mix-ups when the pipeline is running multiple times.

Evaluation

The last component, the evaluation of the computed results, is once more using the Python programming language. However, to that end we are not simply using scripts but so called Jupyter Notebooks. These notebooks are web-based environments consisting of cells which can contain (and run) code, text, media and more. The cells can be processed separately or automatically one after another. For developers these notebooks can be a powerful tool to implement analyses and quickly see the results. For users the notebooks are interactive and intuitive to handle. In addition, they can be exported to a variety of different formats: They can be shared in their own format in case they should remain editable. They can be saved as an HTML document in order to remain interactive. For strict documentation purposes, they can be stored in markup language or as PDFs.

We use notebooks as a way to evaluate the previously processed data and visualize our findings. For our defined visual place recognition pipeline, we created a first notebook as a comparison tool for different methods and parameterizations. The user can define multiple result directories (compliant to the previously described format), run the evaluation notebook and subsequently obtain multiple plots about the accuracy and run-time performance of the specified methods. Yet again we follow our modular approach and outsource helper functions such that they can be included into multiple notebooks.

In summary, we want to emphasize the advantages of the modular design concept of our benchmarking suite with some examples. If one wants to evaluate an existing method on a different dataset, it is only necessary to provide the data in the specified input format. The remaining pipeline stays unaffected. If an additional method shall be used in the benchmark, implementing a class which inherits from the abstract method base class is enough. Again, the other components do not have to change. Even if the method is not available in C++ (imagine some deep learning frameworks using Python), one simply has to take care of the defined interfaces and can then still use the data preparation and evaluation parts. More evaluation metrics can easily be added by changing the provided Jupyter notebook or by creating a new one. This flexibility combined with the exhaustive documentation makes the benchmark very easy and fast to work with.

5.1.3. Setup and Workflow

In this last part of the section, we want to give a short outline of the setup and typical work flow of our currently implemented benchmarking pipeline for place recognition based on pure image retrieval. Above all, the project has some dependencies which are strictly required in order to be able to use it:

- The version control system Git for cloning the project and third-party repositories
- Python 3 for the data preparation and evaluation
- The Python library PyYAML for YAML handling in the Python scripts
- Jupyter and Matplotlib for the evaluation notebook
- CMake with a minimum version of 3.10 to set up the place recognition application
- The GNU Compiler Collection (GCC) with a minimum version of 8.4 for building the application

These packages have to be provided by the user. The project is currently hosted as a GitLab repository, so as a first step users have to clone it onto their local machine. For a quick setup we provide a Bash script in the repository which takes care of the required third-party libraries for the data processing component. Since these libraries are included in the project as so called submodules, the script first synchronizes and updates these external repositories. Afterwards, it automatically builds the required OpenCV components and the DBoW library. The script also applies a small patch to the DBoW header files in order to make it compatible with C++17. The user can then build the place recognition application according to the provided CMake file. After that, the project is ready to be used.

The next step is usually to get the desired datasets for the evaluation. At this time, we provide setup scripts for three different datasets (which are introduced in the next section). The scripts handle download, extraction and conversion of the data into the defined input format for the subsequent processing part. Users can specify the output location and – if applicable – the minimum ground truth quality. Additionally, we provide a script for merging multiple sets of data.

Next, we can continue with the data processing part, specifically the place recognition application. After the user starts the program through the command line, the validity of the inputs is verified and a message containing information about the settings is printed to the terminal. The application then loads the images from the data directory and extracts keypoints and descriptors using the defined feature type. Afterwards, the benchmarked place recognition approach is initialized based on the parameter file or its default settings. Next, the images – or rather the extracted features – which are not specified as queries are added to the image database. If applicable and necessary, the place recognition method then proceeds with training. As an example, if HBST is used without incremental tree construction, it will build the binary tree in this phase of the application. In the next step, the database is queried by the defined images and the retrieved results including their calculated scores are stored internally. Finally, the settings, image retrieval results and run-time measurements are saved in a directory labelled with the current time stamp at the previously specified location or at the current working directory.

After one or multiple runs of the application, the provided Jupyter notebook can be used to evaluate the results. To this end, the user starts a local Jupyter server and opens the notebook in any web browser. In a code block, the user can specify the desired result directories and some descriptive names. When running the notebook, the code blocks are hidden for easier readability. The notebook then verifies the validity of the

input directories and computes the included performance metrics. Finally, the results are visualized and can be inspected by the user. In order to share the findings, the user can download the notebook in multiple available formats.

5.2. Evaluation Contents

Using our new benchmarking suite, we can now investigate the performance of place recognition methods under varying conditions. There are different levers available which can influence their accuracy and run time: First, the parameterization of the methods can change and therefore alter performance and efficiency. Second, the choice of the feature extractor and its invariance properties can have a significant impact on the results as well. Finally, some datasets may be more challenging than others, or certain algorithms work better for a particular environment. It is therefore important to use multiple datasets for evaluating place recognition approaches. Hence, in this section we want to outline our chosen areas of interest. We will describe the methods and their parameter options, the available feature extractors and the chosen datasets. As a last point, different metrics can be used to quantify the results, so we will give a short rationale about our choices there as well.

5.2.1. Methods

The selection of methods for the evaluation has already been justified in Chapter 4. We presented three place recognition approaches – DBoW, HBST and HashBoW – which are all pure image retrieval techniques based on extracted local features. Each method has a different set of adjustable parameters, possibly influencing their accuracy and efficiency. In this part, we thus describe these parameters and how we are evaluating them.

As noted previously in Section 4.3.1, there are multiple implementations for the hierarchical bag-of-words approach available. For this evaluation we chose to include DBoW3, an improved version of DBoW2. This particular implementation has the big advantage of being directly compatible with all descriptors provided by OpenCV which we are using for feature extraction. Furthermore, the author rewrote certain parts to optimize the run time of the algorithm. The set of available parameters is equivalent to that of the original publication: The vocabulary size can be determined by the branching factor k and tree depth L. The bag-of-words vectors can be weighted binary, using term frequency, inverse document frequency or the combined TF-IDF

weighting scheme. There are also multiple scoring types available including L_1, L_2, χ^2, *Kullback–Leibler*, *Bhattacharyya* and *Dot-Product* scoring. For more information about these scoring functions, refer to the DBoW3 code [63] or to the extensive work by Deza and Deza [19]. The parameter baseline we will use is defined by the choices of the original authors in their own evaluation. They decided to use a comparatively large vocabulary with one million words ($k = 10$, $L = 6$), resulting in higher bag-of-words computation times but faster queries due to the sparse inverted index. They also chose the popular TF-IDF weighting scheme and the L_1 scoring type. In the evaluation we will first benchmark the approach with this default parameter set and then successively change a particular parameter to see the effects.

For HBST, we are using the original implementation provided by the authors of the paper. The approach can be adjusted by the splitting decision threshold δ_{max}, the maximum leaf size N_{max} and the maximum tree depth h. Additionally, the binary tree can be constructed incrementally or at once, using even, uneven or random uniform splitting strategies. The original paper uses an incremental tree construction using even splitting. They do not limit the maximum tree depth and set the splitting decision threshold δ_{max} at 0.1. The authors evaluate their approach with smaller ($N_{max} = 10$) and bigger ($N_{max} = 50$) leaf sizes in order to emphasize the trade-off between performance and run time. Again, for our baseline we will use these default parameters, setting the maximum leaf size to the mean value 30. We will then try different settings for each parameter to see how they influence the results.

Lastly, our novel approach HashBoW has two adjustable parameters. The first one is the dimensionality of the hash codes, that is to say the number of sampled bits, n. This choice also defines the theoretical vocabulary size, as previously noted in Section 4.3.3. Second, the scoring type can be set. We provide three different scoring types: L_1, L_2 and the cosine similarity. Since HashBoW is a new technique, there exist no empirical default values for this approach so far. We will therefore try various parameter combinations in order to determine the optimal settings. Obviously, the performance of the approach is also impacted by the choice of the hashing function. For the parameter search, we simply use random bit sampling which can be justified by its locality-sensitive property.

5.2.2. Feature Extractors

In the evaluation we limit the range of feature extraction methods to the ones available through the OpenCV library. We also do not combine different keypoint detectors and feature descriptors, but rather use complete feature extractors. Finally, we only consider binary descriptors since HBST is not defined for real-valued descriptors and we did not appoint a suitable hashing function for HashBoW yet.

The remaining feature extraction methods are AKAZE, BRISK and ORB which have already been presented in Section 3.1.1. For setting up the algorithms, we largely use the default parameters provided by OpenCV. We only adjusted the maximum number of extracted features from ORB. We noticed that AKAZE and BRISK extract a very high number of features per image on average, ranging from 2 000 to 8 000 depending on the dataset. The default limit for ORB, however, is set to only 500. On the other hand, setting no limitation results in a much higher number of extracted descriptors, often in more than 20 000. In order to achieve comparable expressiveness while maintaining efficiency, we thus increased this artificial limit to 2 500. Additional information about the full set of parameters for each feature extractor can be found at the official OpenCV documentation [76].

5.2.3. Datasets

We evaluate the place recognition approaches on three different datasets, all created for the particular purpose of testing image retrieval systems. We provide a setup script for each dataset which handles the download and conversion into our defined format. In addition, we are using a fourth, independent set of data for training DBoW and HashBoW generically.

The first dataset is the Oxford Buildings Dataset which has been presented in 2007 as part of a publication by Philbin *et al.* [78]. It consists of 5 063 images retrieved from the image hosting service Flickr by searching for famous landmarks of Oxford in the UK. The dataset has been annotated manually, providing a ground truth of image matches for eleven landmarks with 5 queries each (resulting in 55 queries in total). The ground truth matches are assigned one of four possible labels – *good*, *ok*, *bad* and *junk* – representing the quality of the match. This quality is defined by the visibility of the specific object in the image. The authors additionally provide some evaluation tools written in C++, extracted SIFT descriptors, a trained vocabulary and a set of 100 000 distractor images. In our evaluation we only use the original images and the ground

truth labels. For this dataset we define a ground truth match only if it is labelled as *good*, meaning it is a clear picture of the building. The minimum quality can however be specified as a command-line argument of the script.

As part of a paper published one year later [79], the same author group provided another set of data, called the Paris Dataset. As the name suggests, it contains 6 412 (of which 20 are corrupt and cannot be used) images of popular landmarks of Paris, France. It is again collected through the Flickr image search and includes manually annotated labels for 55 query images. The format of the dataset is equivalent to Oxford Buildings, however it does not include the quality label *good* such that we use *ok* as our minimum quality requirement.

The third and final dataset used in the evaluation is the INRIA Holidays Dataset, released by Jegou *et al.* [40] in 2008. This set is more diverse than the previous two because it contains a wider range of objects: Besides buildings, it includes scenes like natural environments, groups of people, fireworks and more. The dataset provides 1 491 images in 500 groups where each group represents a particular scene. The first image in every group is treated as the query with the other images being the ground truth matches. In contrast to the other datasets, the authors do not annotate the matches with a quality label. Similar to Oxford, the dataset includes evaluation tools, extracted features and vocabularies, which we do not use. This is the only dataset which we slightly modify from the original for the evaluation: Because the pictures are in high resolution, we scale the images such that the longer side has a length of 1 024 pixels. That way, the images are sized equally to the Oxford Buildings and Paris Buildings datasets, and we prevent skewed results based on the difference in resolution.

Lastly, we employ the Places365 dataset from Zhou *et al.* [119] to train DBoW and HashBoW independently from the evaluation data. The whole dataset contains about 2 million images from 365 distinct scene categories. However, the DBoW approach is typically trained on a set of around 10 000 images [24], [25], [66], suggesting that a diverse image collection of that size is enough to generate meaningful vocabularies. We therefore randomly choose 10 000 images from the validation set of Places365 as our training collection. Again, we scale the images to 1 024 pixels on the long side. Our experiments show that the created vocabulary is equally expressive to, for example, the one used in ORB-SLAM2 [67] and therefore suitable for generic place recognition tasks.

5.2.4. Metrics

In order to assess the performance of the considered methods, we evaluate their accuracy and run-time efficiency. The provided notebook implements a number of metrics for this task. For measuring the image retrieval accuracy, we adapt a metric commonly used in literature about place recognition [5], [87], [111]. There, an image is said to be correctly localized if at least one of the top-k results is within 25 meters of the query image position. Since we do not consider the location of the camera, we are adapting the metric such that a place is said to be correctly recognized when at least one of the results is among all possible ground truth images. The percentage of correctly recognized queries can then be plotted over the number of retrieved results k. In the ideal case, each query is correctly localized within a small number of k which is visualized by a rapidly increasing curve in the plot. Since we know the full set of ground truth matches, we can additionally calculate the *recall* – also called true positive rate or sensitivity – of the image retrieval process. The recall is defined as the fraction of true positive results (N_{TP}) among all possible ground truth matches (N_{GT}):

$$\text{recall} = \frac{N_{TP}}{N_{GT}} \tag{5.1}$$

The recall can again be plotted over various values of k. In most cases, the conclusions drawn from both metrics are largely the same, so in our evaluation we usually only consider the first one.

We are evaluating the efficiency of the approaches by simply measuring their run time in each step of place recognition: adding images, training the algorithm and querying the database. We save the exact run time for each individual call and report the accumulated sum for the complete run of the given dataset. The measurements are ensured to only include the actual processing necessary for place recognition and no additional conversions which may be needed to use the approach, for example regarding the data structure of the descriptors. In case no training is necessary, we omit this part of the evaluation. This is the case when HBST is building its tree in an incremental fashion, or when DBoW and HashBoW are using predetermined vocabularies.

5.3. Evaluation and Discussion of Results

In the final section of this chapter, we will now use the introduced benchmarking suite to evaluate the performance of the previously defined methods. The first subsection deals with the influence of the parameters of each of the three considered methods. For DBoW and HBST, starting from the default values suggested in the original paper, we alter each parameter independently and see how it impacts the accuracy and efficiency of the approaches. Since we do not have default values for HashBoW yet, we check multiple combinations of the available parameters. Next, we will investigate possible improvements of the performance through the particular training procedures of each method. After that, we will look into the different feature extraction methods – AKAZE, BRISK, and ORB – and find out which one works best for a specific method. Lastly, using the determined optimal parameter set of each approach, we perform a summarizing comparison of the methods and carve out their strengths and weaknesses.

It is important to emphasize that we use a rather greedy approach to determine the settings for each of the methods. By default, we use the ORB feature extractor and evaluate the parameters one after another, fixing the best performing parameter after each step. This means that in theory we could miss some superior combinations. However, the amount of variables in this evaluation is simply too high to allow an exhaustive test of all possible parameter, dataset and feature extractor combinations. Additionally, experiments with random combinations have shown that this greedy approach works good enough to generate reasonable insights. In the general case, we report our findings on the INRIA Holidays dataset only, since the conclusions are valid for Oxford and Paris as well.

The benchmarking was performed on a gaming notebook running Linux Mint 19.3 (Ubuntu 18.04). The machine is powered by an Intel i7-6700HQ quad-core CPU clocked at 2.6 GHz and equipped with 16 GB of RAM. The complete set of results together with all raw data files is provided in Appendix C.

5.3.1. Parameter Analysis

DBoW

The first and most important parameter of DBoW is the size of the vocabulary, respectively its structure. The vocabulary size is determined by the branching factor k and tree depth L. The higher the branching factor and depth level, the wider and deeper the vocabulary tree gets. These two settings influence the accuracy and run time in each

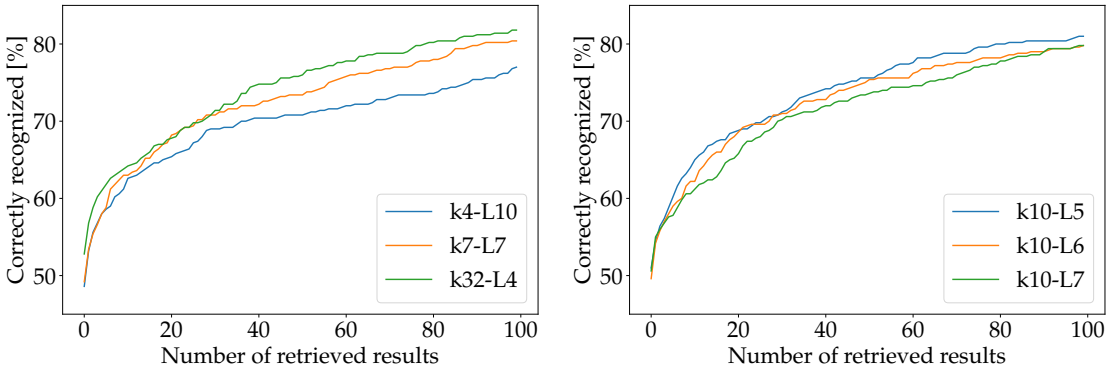
Parameters	Training time
$k = 4$, $L = 10$	74 min 23 s
$k = 7$, $L = 7$	65 min 35 s
$k = 32$, $L = 4$	83 min 6 s

(a) Deep, balanced, wide

Parameters	Training time
$k = 10$, $L = 5$	64 min 41 s
$k = 10$, $L = 6$	74 min 12 s
$k = 10$, $L = 7$	78 min 35 s

(b) Small, medium, large

Table 5.1.: Influence of vocabulary size and structure on the time spent for training DBoW. Results are reported for training the DBoW3 library using around 23 million descriptors extracted from a subset of the Places365 dataset as described in Section 5.2.3.



(a) Deep, balanced, wide

(b) Small, medium, large

Figure 5.1.: Accuracy of DBoW depending on the vocabulary size and structure

place recognition phase significantly. The original paper suggests a rather balanced structure with a branching factor of 6 and a maximum depth level of 10, resulting in a vocabulary containing 1 000 000 terms. Starting from this baseline, we have changed both the size and the structure of the vocabulary, to investigate their impact. Regarding the structure, we trained and used three different models with similar size: one deep and narrow ($k = 4$, $L = 10$), one balanced ($k = 7$, $L = 7$), one wide and shallow ($k = 32$, $L = 4$). Regarding the vocabulary size, we increased and decreased the default setting by an order of magnitude using the depth level.

The time spent training the models is summarized in Table 5.1. We can see that training a balanced tree takes less time than very deep or very wide ones, with the wide tree being the slowest overall. Also, bigger vocabularies need more time to train which intuitively makes sense. When using the models, wider trees performed better than deeper ones, as visualized in Figure 5.1a. From that we can assume that a more detailed clustering at higher levels is preferable to a more distinct clustering within the clusters

Parameters	Add	Query
$k = 4$, $L = 10$	12.88 s	10.40 s
$k = 7$, $L = 7$	13.04 s	10.39 s
$k = 32$, $L = 4$	20.88 s	13.35 s

(a) Deep, balanced, wide

Parameters	Add	Query
$k = 10$, $L = 5$	9.95 s	19.00 s
$k = 10$, $L = 6$	14.33 s	10.42 s
$k = 10$, $L = 7$	17.74 s	10.80 s

(b) Small, medium, large

Table 5.2.: Processing times of DBoW depending on the vocabulary size and structure

themselves. The vocabulary size has negligible impact on the accuracy, at least in the range we have investigated. The run time of the different models is shown in Table 5.2. We measured the time it takes to add the images (bag-of-words conversion and insertion) and to query the final data base. In general, wider vocabulary trees seem to be slower, especially in the addition phase. The bigger the vocabulary, the longer it takes to convert images into bag-of-words representations and add them to the data base. On the other hand, small vocabularies result in considerably slower query responses. This behavior has already been explained in the original paper with the sparsity of the inverted index when using bigger sized vocabularies. In summary, given a sufficiently large vocabulary, the DBoW approach is not overly sensitive to the specific size and structure of the tree. A rather large, balanced vocabulary tree seems to be a reasonable compromise between accuracy and all relevant processing times. In the following sections, we therefore use the default suggestion of the original paper ($k = 10$, $L = 6$).

Next, we looked into the different available weighting and scoring types. The image representations can be calculated using binary, TF, IDF and TF-IDF weighting. Our evaluation has shown that the particular strategies have no significant impact on the performance or run time of the method. In our tests, the TF-IDF and IDF weighting schemes resulted in a slightly better accuracy, however the amount is so small that we do not want to convey this as a general conclusion. Finally, we evaluated the influence of the available scoring types. Comparing the accuracy, dot product scoring is significantly worse than the other strategies, probably due to the fact that it does not normalize the vectors. Kullback–Leibler scoring has decent accuracy but is prohibitively slow in the query phase. Besides those two, all other scoring schemes perform very similar both in accuracy and run-time efficiency. Hence, we decided to use the default parameters (TF-IDF with L_1 scoring) here as well.

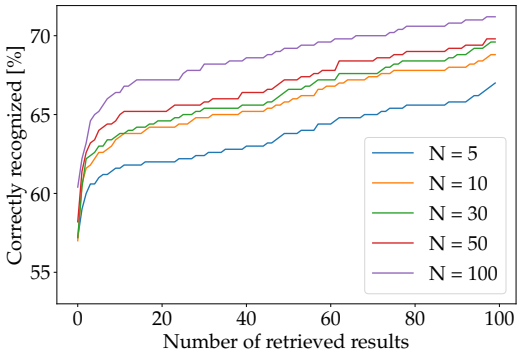


Figure 5.2.: Accuracy of HBST depending on the maximum leaf size

Maximum leaf size	Add	Query
5	9.69 s	2.10 s
10	4.58 s	2.25 s
30	5.15 s	2.71 s
50	6.20 s	3.22 s
100	8.85 s	4.61 s

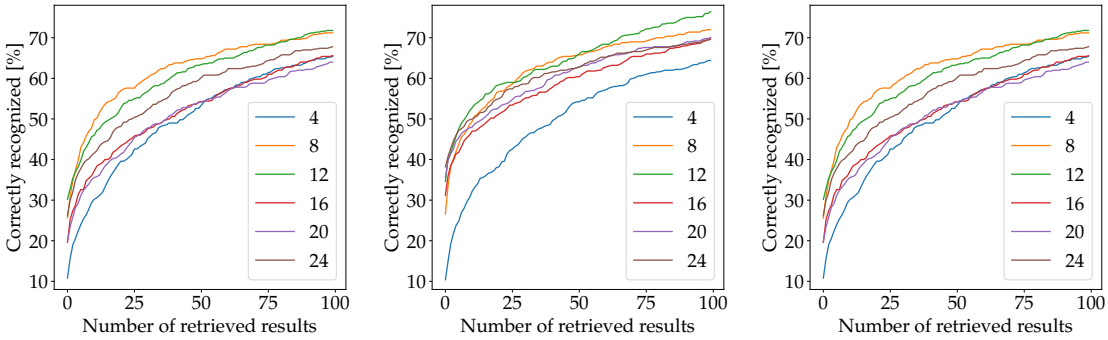
Table 5.3.: Processing times of HBST depending on the maximum leaf size

HBST

The main parameter for HBST is the maximum leaf size N_{max}. In the original paper, the authors evaluate their approach with the parameter set to 10 and 50. We go one step further and check the influence for a range of 5 to 100 maximum descriptors per leaf. In theory, a higher value for this parameter should result in better accuracy but slower processing times because each descriptor corresponds to a larger amount of nearest neighbor candidates. The results of our benchmarking procedure shown in Figure 5.2 and Table 5.3 confirm this hypothesis. The only outlier is the run time in the image addition phase for a very small leaf size, which is exceptionally high. We assume that this effect occurs due to the higher frequency of necessary splits but have not confirmed this theory. We figure that a maximum leaf size of 30 represents a good trade-off between accuracy and efficiency and subsequently take this as our default value.

It is also possible to limit the depth of the constructed tree. By default, the tree depth is not limited, that is to say the maximum depth level is equal to the dimension of the descriptor type. Since in our experiments the resulting depth never comes close to this theoretical value even for the biggest datasets, we refrained from evaluating this parameter further.

Lastly, we investigated the impact of the splitting decision threshold δ_{max} and the splitting strategy. The approach allows even, uneven or random splitting to construct the tree. Our results show that uneven splitting – which means generating a very unbalanced tree – actually achieves the best accuracy. However, the run time gets prohibitively slow: in comparison to the other splitting strategies, both addition and



(a) Cosine Similarity **(b)** L_1 **(c)** L_2

Figure 5.3.: Accuracy of HashBoW depending on the number of bits for different scoring types

query times increase by a factor of 5000 to 10 000. This is not feasible for real-time applications. On the contrary, the even and random splitting strategies result in similar accuracy and run time. This is probably also the reason why the choice of the splitting decision threshold has only minor impact on the results as well. We therefore follow the suggestions of the original paper and use even splitting with a threshold of 0.1. For a more in-depth report, we refer to the appendix.

HashBoW

As previously mentioned, our novel HashBoW approach can be adapted using two parameters: the number of bits of the generated hash codes and the scoring type used for ranking the query results. In contrast to DBoW, the weighting type is currently fixed to a simple term-frequency scheme. However, a subset of meaningful bit indices can be calculated using the entropy-based training strategy which is evaluated in Section 5.3.2. Our implementation includes three available scoring types – cosine similarity, L_1 and L_2 – and allows a maximum hash size of 32 bits.

Since a baseline of default parameter settings does not exist yet, we performed the evaluation over a wider range of parameters compared to DBoW or HBST. For the number of bits, we decided to test values in a range from 4 to 24 bits, covering both very small and very big vocabulary sizes (16 to 16 777 216 possible terms). In each run we increased the number by 4 bits, such that we actually tested six distinct values for this parameter. We evaluate each bit number in combination with each implemented scoring type, resulting in 18 different parameter sets.

Bits	Add	Query
4	0.13 s	0.19 s
8	0.15 s	1.18 s
12	0.28 s	3.00 s
16	0.58 s	1.98 s
20	0.85 s	1.07 s
24	1.17 s	0.65 s

(a) Cosine Similarity

Bits	Add	Query
4	0.12 s	0.19 s
8	0.15 s	1.23 s
12	0.27 s	3.19 s
16	0.56 s	2.09 s
20	0.84 s	1.11 s
24	1.19 s	0.71 s

(b) L_1

Bits	Add	Query
4	0.13 s	0.18 s
8	0.15 s	1.19 s
12	0.27 s	2.98 s
16	0.64 s	2.02 s
20	0.85 s	1.08 s
24	1.14 s	0.64 s

(c) L_2

Table 5.4.: Processing times of HashBoW depending on the number of bits for different scoring types

The results of the accuracy evaluation is visualized in Figure 5.3. The plots clearly show that a very small vocabulary, resulting from the 4-bit hash codes, is not expressive enough for the task and therefore performs poorly. Depending on the employed scoring type, the 8-bit and 12-bit hash codes seem to work best. For bigger vocabularies, the accuracy of the method actually declines. Similar to DBoW, the choice of the scoring type has only minor influence on the results. The run time for each set of parameters is reported in Table 5.4. Naturally, the run time for adding images to the database grows as the number of bits increases. This intuitively makes sense, since more bits need to be processed in order to generate the hash code. The processing time for querying the database starts small, then grows until it hits a peak at the 12-bit version, and declines afterwards. We assume this is due to the fact that the inverted index becomes increasingly sparse with a growing vocabulary size, a behavior previously noted in the DBoW evaluation as well. Again, the particular scoring type has little effect on the overall run time. With these insights in mind, we decide to pick a 8-bit hash code as our default setting, since it combines high accuracy with good run-time efficiency. Because it has no significant impact on the results, we choose the L_1 scoring type to achieve comparability with the DBoW settings.

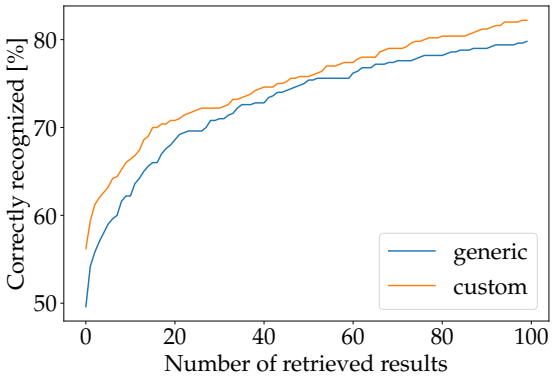


Figure 5.4.: Accuracy of DBoW trained generically and custom

5.3.2. Training Differences

DBoW: Custom Training

Alongside the specific parameter settings, the choice of the dataset used for training has significant impact on the performance of DBoW. The method is usually trained on a large, diverse and independent set of data in order to create a generic vocabulary usable for different image retrieval applications. Naturally, a generic vocabulary can perform worse than a custom vocabulary which has been trained for a specific dataset.

For our evaluation we want to measure this potential increase of performance. Therefore, we trained custom vocabularies for our three available benchmarking datasets and compared their accuracy to the generic vocabulary trained from the Places365 dataset. The results for the Holidays dataset are visualized in Figure 5.4. As expected, the custom trained vocabulary performs better than the generic one. In this particular instance the custom vocabulary increased the percentage of correctly recognized places by about 2.5 percentage points. Across all evaluated datasets we saw a maximum increase of 5 percentage points in this metric and also in recall.

Hence we suggest that if the application environment is known a-priori, a custom trained vocabulary should be used to achieve maximum performance. If, however, the environment is not explicitly known, it is preferable to use a generic one since the actual drop-off in accuracy is rather small.

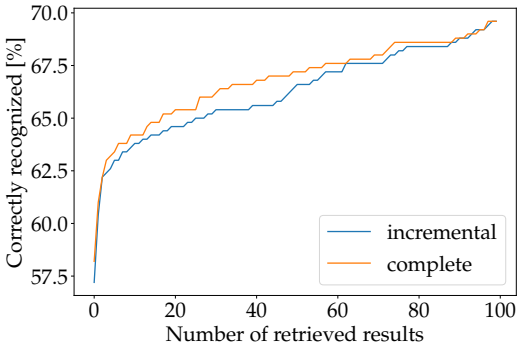


Figure 5.5.: Accuracy of HBST depending on the tree construction strategy

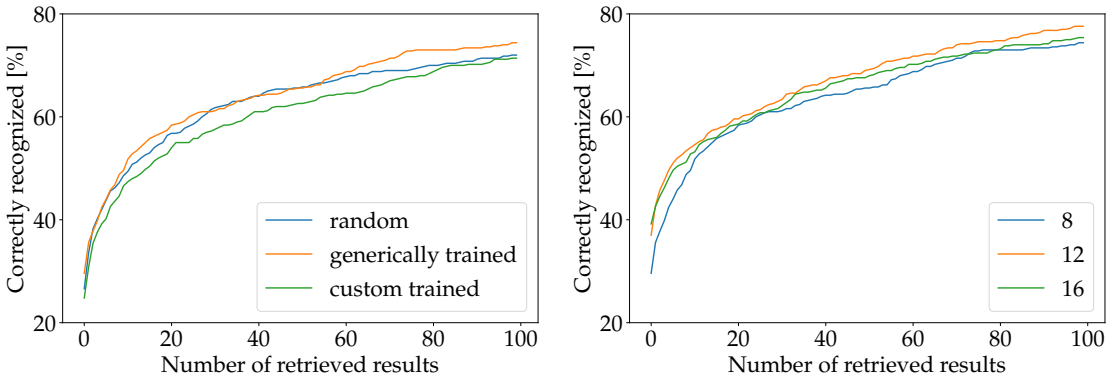
Construction	Add	Train
Incremental	5.17 s	-
Complete	0.02 s	103.27 s

Table 5.5.: Image addition and training times of HBST depending on the tree construction strategy

HBST: Tree Construction Strategy

As already described in Section 4.3.2, HBST offers two possible strategies to construct its search tree. Their initial strategy was to build the tree from scratch using all available descriptors. In order to do this, for every tree level the method needs to iterate over the full set of descriptors in order to find a suitable bit index for splitting the nodes. This algorithm has a complexity of $\mathcal{O}(N \cdot h)$, where N is the number of descriptors and h is the tree depth. This quickly becomes infeasible in real-world applications because the number of descriptors grows significantly with each new image. Thus, the authors propose to construct the tree incrementally by splitting leafs once they contain a defined amount of descriptors. This incremental construction is the default behavior of the approach.

In order to verify that this algorithm performs sufficiently well for place recognition tasks, we measured the accuracy and run time of both construction schemes. Figure 5.5 shows the percentage of successfully recognized places using the two strategies for the Holidays dataset. Here, the accuracy of the tree constructed from scratch is slightly better than the incrementally constructed one. This also holds true for the other datasets in the benchmarking suite, although the amount of improvement varies considerably. When looking at the run-time differences in Table 5.5 though, it quickly becomes clear that the complete tree construction is not a viable approach for real-time insertion of new images. For this rather small dataset of about 1000 reference images, the tree construction already takes over 100 seconds. On the contrary, the incremental construction strategy takes just over 5 seconds to add the image descriptors and simultaneously generate the tree.



(a) Training Overview **(b)** Bit Number Influence

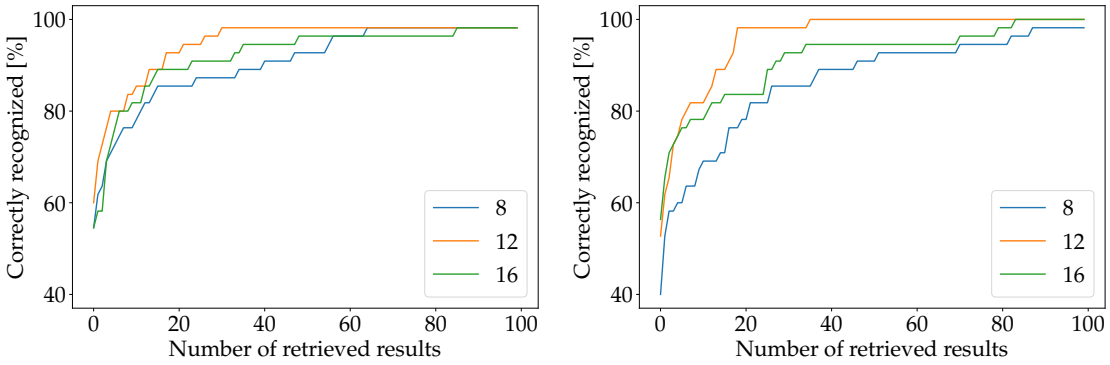
Figure 5.6.: Accuracy of HashBoW for the Holidays dataset depending on the (a) type of training and (b) number of bits when trained generically

To sum up, the incremental tree construction is an efficient approach to create a binary search tree suitable for accurate place recognition. However, if images do not have to be inserted in real time, generating the tree from scratch can yield better performance.

HashBoW: Entropy-Maximization of Hash Codes

Lastly, we want to evaluate the proposed entropy-based training strategy for HashBoW. Remember, the training procedure aims to find the subset of bit indices which maximizes the entropy of the generated hash codes. In theory, this should make the vocabulary more meaningful and result in better image retrieval performance. In order to confirm this hypothesis, we compared the performance of HashBoW using three different sets of bit indices: the default random set of bits, the entropy-maximizing set of bits from Places365 (“generic”) and the entropy-maximizing set of bit indices from the specific dataset which is being evaluated (“custom”).

The results, using the previously defined parameter settings, are shown in Figure 5.6a. Unfortunately, with the default settings, the training does not impact the accuracy in any positive way. One possible reason could be the diversity of the Holidays dataset. However, the data for Oxford and Paris did not show significantly different results. Another hypothesis is that the vocabulary defined by the 8 bits cannot get more expressive because it is simply too small to benefit from the training procedure. In order to confirm this hypothesis, we evaluated the training using different numbers of bits for the hash code. Figure 5.6b shows the percentage of successfully recognized



(a) Generic **(b)** Custom

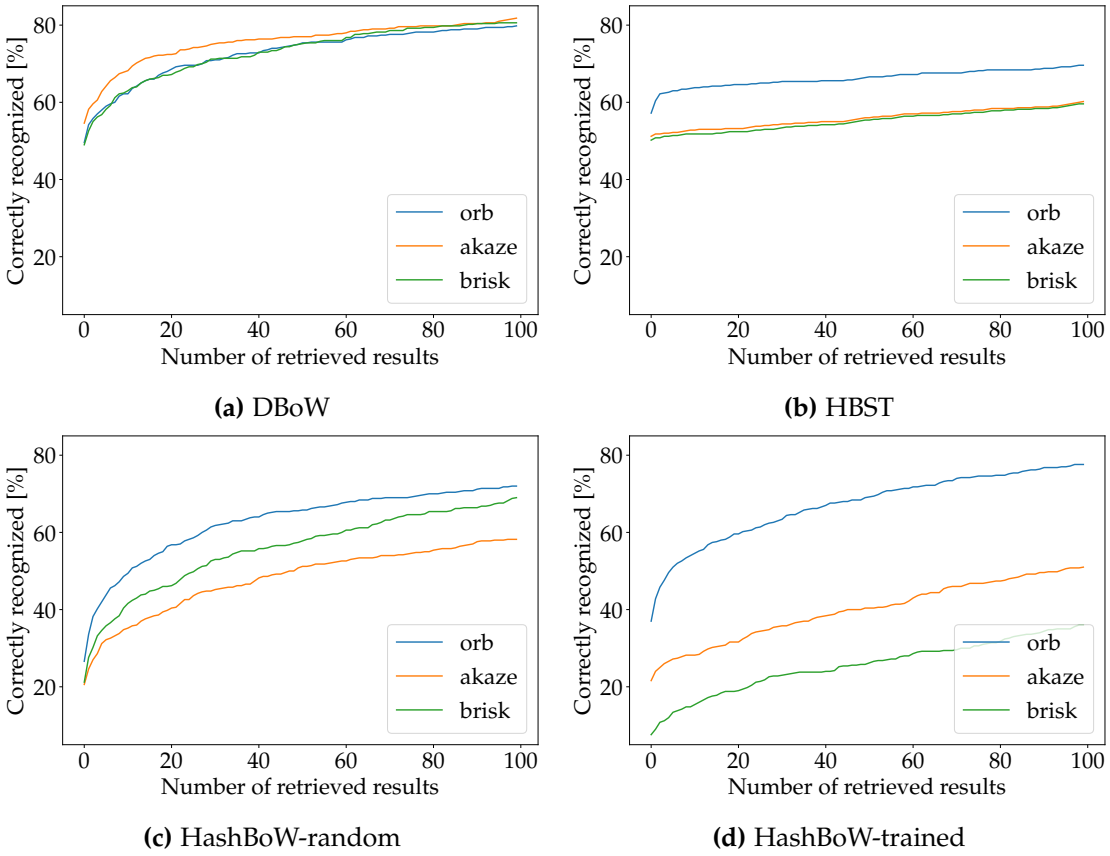
Figure 5.7.: Accuracy of HashBoW for the Paris dataset depending on the number of bits and the type of training (generic or custom)

images for a generically trained HashBoW using 8, 12 and 16 bits. Indeed, in contrast to the untrained version (compare Figure 5.3), the accuracy increases for longer hash codes. The improvement is even bigger for other datasets like Paris which is visualized in Figure 5.7.

Overall, the results suggest that a 12-bit HashBoW using an entropy-maximizing set of bits achieves the highest possible accuracy. However, we have already shown that the 12-bit hash codes also impacts the query efficiency negatively. In the following sections, we therefore evaluate both versions of HashBoW: "HashBoW-random" is the 8-bit version with random sampling, "HashBoW-trained" is the mentioned entropy-maximizing 12-bit version. The entropy maximization is performed on the Places365 dataset in order to avoid overfitting to the particular datasets.

5.3.3. Influence of Feature Extractors

Next, we evaluate the sensitivity of the methods to the employed feature extractor. As already noted, the benchmarking suite includes AKAZE, BRISK and ORB. So far, all evaluations were based on ORB features, due to the fact that it is the most popular binary feature extraction method in literature and relevant applications. Compared to the other two, it is also considerably faster. In this section, we only report results for the accuracy, not the run time. Since the amount of extracted descriptors heavily impacts the run time, we cannot guarantee a fair comparison of the processing times. For example, BRISK consistently detects a larger number of keypoints than AKAZE or ORB. Nevertheless, in general it can be said that descriptors with lower dimensionality



(a) DBoW **(b)** HBST

(c) HashBoW-random **(d)** HashBoW-trained

Figure 5.8.: Accuracy of place recognition methods depending on the feature extraction algorithm

can be processed faster. ORB (256 bits) is therefore more efficient than AKAZE (488 bits) or BRISK (512 bits). The concrete impact of this depends on the actual place recognition method.

Figure 5.8 shows the accuracy of our considered methods depending on the employed feature extractor. DBoW seems to work best when using the AKAZE algorithm, which holds true for each of the available datasets. Apart from DBoW, the other methods work better with ORB features. For these approaches the accuracy increase can be as high as 10 percentage points which is quite significant. One possible explanation for these results could be the length of the descriptors: HBST and HashBoW are focused on single bits which are less informative for longer descriptors. DBoW, on the other hand, takes the whole descriptor into account and can therefore make use of the more expressive AKAZE. However, this is only an assumption which we have not explicitly verified and

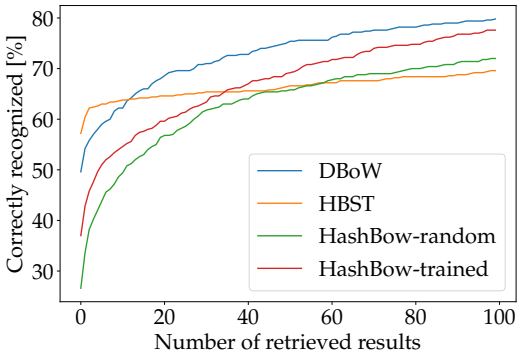


Figure 5.9.: Accuracy comparison of VPR methods

Method	Add	Query
DBoW	14.15 s	10.27 s
HBST	5.09 s	2.67 s
HashBoW-random	0.15 s	1.23 s
HashBoW-trained	0.32 s	3.74 s

Table 5.6.: Processing times comparison of VPR methods

thus more research in this direction is necessary. Another interesting insight is that while the ORB version of HashBoW benefits from the entropy-maximization training, AKAZE and especially BRISK suffer a considerable accuracy decrease. Because our evaluation focuses on ORB features, we have not investigated this behavior in-depth, though it could be a worthwhile question for future work on HashBoW. In the following final section, we again limit our evaluation to the ORB extractor.

5.3.4. Method Comparison

In this last section of the chapter, we want to compare the performance of all methods using the best-performing parameter sets. As a summary, those are the following four methods:

- DBoW trained on the Places365 dataset with a branching factor of 10, tree depth of 6, TF-IDF weighting and L_1 scoring
- HBST whose tree is incrementally constructed using even splitting, a maximum leaf size of 30 and a splitting threshold of 0.1
- HashBoW-random using 8-bit hash codes with randomly sampled bits and L_1 scoring
- HashBoW-trained using 12-bit entropy-maximizing hash codes trained on the Places365 dataset and L_1 scoring

The evaluation was performed using the ORB feature extractor. The results for the Holidays dataset are shown in Figure 5.9 and Table 5.6. The following discussion of the results is valid for the other datasets as well.

Regarding the accuracy, HBST is exceptionally good at the very beginning, that is to say it correctly recognizes more scenes within the first 5 to 10 reported results compared to the other methods. Afterwards, however, its performance quickly drops off and it struggles with the remaining, unrecognized queries. DBoW has a slightly worse start but quickly supersedes the accuracy of HBST after about 10 retrieved results. Overall, also considering the results for Oxford and Paris, DBoW achieves the highest accuracy of all methods. Both HashBoW versions are slightly worse than the other methods, although they catch up to the performance of HBST after about 30 to 40 results.

In contrast, HashBoW is the clear winner when considering run-time efficiency. Looking at the image addition phase, HashBoW is one to two orders of magnitude faster than both HBST and DBoW. The gap is smaller in the query phase, where HBST can at least keep pace with the trained version of HashBoW due to its usage of 12-bit hash codes. Still, also in this phase, HashBoW-random is clearly the fastest approach. DBoW, on the contrary, is significantly slower than the other methods.

All in all, DBoW seems to be the method of choice if the user is interested in the highest possible accuracy. When run-time efficiency is the limiting factor, HashBoW offers the fastest place recognition performance. HBST is a very good compromise between those two, achieving high accuracy at low processing times. Also, if only very few results are retrieved, HBST has the highest accuracy of all considered approaches. Note that these statements – especially the ones concerning run time – only hold for the particular implementations we have chosen. In the next chapter we will show that DBoW can be made much more efficient with some rather simple adjustments.

6. An Efficient Library for Visual Place Recognition

Driven by the results of the previous chapter, we decided to develop a new library containing a collection of efficiently implemented algorithms for visual place recognition. In this chapter, we will first explain our motivation and state goals for the library. Next, we will outline its structure and contents. Lastly, we will present and evaluate our improvements of the included methods. The full source code of the library is available online (see Appendix B).

6.1. Motivation

The idea of setting up an open-source library for visual place recognition approaches initially came up while working on a first prototypical working version of HashBoW and studying various implementations of related methods. We realized that many approaches based on the bag-of-words scheme actually follow a very similar structure. Typically, binary or real-valued descriptors are received as inputs. The algorithms then employ a conversion step where the individual features from an image are clustered into a set of words, producing a single vector representation for each image. The set of words – the vocabulary – is usually trained prior to the actual place recognition task from a large, diverse set of local features. After converting the features of an image into the bag-of-words representation, the image is added to the database which in most approaches is implemented as an inverted index. The database can afterwards be queried by an image, a set of descriptors or an already transformed bag-of-words vector. The results are ordered by a scoring function which can usually be specified by the user. Although the actual method implementations can look vastly different, this principal structure is mostly identical. An additional observation is that there are many individual algorithms, like FAB-MAP or DBoW, publicly available. However, there exists no reference collection like OpenCV. Therefore, the performance, code and documentation quality of each method can vary greatly. Also, the methods can use

different data structures for the input descriptors. This can complicate their usage in existing projects, especially when comparing multiple approaches, and makes it harder to compare their performance.

The results from the evaluation in Chapter 5 ultimately triggered the decision to actually publish such a generic library as a collection of place recognition approaches. One key insight of this evaluation was that the DBoW method achieves the highest accuracy overall. Unfortunately, the rather high run times in each step of the procedure diminish its applicability for real-time place recognition. Thus, it seems like a valuable challenge to find the efficiency bottlenecks and get rid of them in order to provide the best possible accuracy with low processing times. As noted previously, there actually already exists a highly efficient version of DBoW called FBOW which is part of the UcoSLAM library [64]. However, the implementation does not contain the full DBoW functionality but only provides the vocabulary part for converting images into bag-of-words representations. Furthermore, it makes heavy use of SIMD vector processing in order to speed up the approach. The applied instruction sets are not available for every processor, so the performance of the implementation can differ considerably. Finally, like DBoW3, it requires OpenCV which is a quite heavy dependency.

Motivated by the previously mentioned problems, we therefore decided to develop an open-source library for image retrieval, focusing on methods based on the bag-of-words approach. The goal is to provide a well documented library such that it is easy to use and understand. Similar to our benchmarking suite, the library shall be written in a highly extensible fashion such that new methods can be added with minimum effort. The initially provided approaches are HashBoW and DBoW which was rewritten to increase its processing speed while maintaining the original accuracy. Additionally, the readability of the code has been improved. Due to its high code and documentation quality, the finished library makes it easy for people new to the field to gain knowledge about the implemented algorithms. It can also support research by providing a reference implementation for multiple bag-of-words approaches. Lastly, since it is designed as a lightweight header-only library, it can easily be incorporated into already existing frameworks.

The remainder of this chapter is organized as follows: After this motivational introduction, we describe the structure and contents of the finalized library. We state its dependencies, justify the design decisions for the different components, and explain how the initially available methods have been implemented. Subsequently, we analyze shortcomings of the original approaches and discuss our introduced improvements. Finally we will evaluate the accuracy and run time of the methods in order to prove the functionality of the new library.

6.2. Library Structure and Contents

As a first point, we want to describe the main design choices we took for the implementation. Fundamentally, the library is designed in a header-only fashion, meaning other projects simply need to include its header files into their application source in order to use the functionality. Header-only libraries certainly have some disadvantages: they tend to be brittle, that is to say even minor changes to the library typically require recompilation of all affected compilation units. They also result in longer compilation times compared to installed dependencies which only need to provide their interfaces. However, its advantages usually clearly outweigh these drawbacks. As mentioned before, header-only libraries are very easy to use. They do not need separate compilation, packaging and installation. In addition, they allow better compile-time optimization because the complete source code is available to the compiler. Finally, header-only libraries are often the only possible way to use template programming. This is also the case in our place recognition library where templates are used to specify the type of descriptor at compile-time. Bag-of-words approaches mostly do not need to know the exact feature extraction method but only if they produce binary or real-valued descriptors together with their dimensionality. Therefore, we provide templated interfaces for these descriptor types, on which the implemented methods can depend on. Thus, it is not important how the descriptors have been computed, they simply have to be mapped to the defined interface in order to use them for place recognition. Since the bag-of-words approaches are templated on the descriptor type and size, they can use generic or specialized functions depending on their particular needs. We also chose to keep the database and the bag-of-words methods independent, since they usually do not have to share any global information. The methods can simply produce bag-of-words representations from feature descriptors which the database then uses for image addition and querying. The following paragraphs will provide more in-depth information about each of the separate components.

Before that, it is important to give a brief overview of the required dependencies, all of which are header-only libraries as well. Similar to the benchmarking suite, the library uses Cereal in order to serialize data. It is specifically used to save and load trained vocabularies or databases to and from disk. The interface for real-valued descriptors employs the well-known Eigen library [32] in order to store and manipulate the values. In addition, a specialized hash table implementation by Skarupke [99] finds use in HashBoW's training procedure, which will be discussed in more detail in Section 6.3. Finally, if unit testing is enabled, the library also depends on the GoogleTest testing framework [30].

Next, we provide details about the input and output interfaces of the library. As already mentioned, the inputs of the library are either binary or real-valued descriptors of a given size. This descriptor type has to be known at compile-time in order to initialize the bag-of-words approaches. The interface for binary descriptors – simply called `BinaryDescriptor` – is an alias template for the `bitset` container, available in C++'s standard library, previously mentioned in Section 3.3.3. It is a specialized container for bit sequences and directly allows logic operations and conversions to string sequences or integers. Bitsets are optimized for run-time and space efficiency which makes them highly appropriate for our application. Analogously, the real-valued descriptor interface – named `RealValuedDescriptor` – is an alias template to Eigen's matrix class. We have chosen Eigen because it is lightweight and extremely popular within the robotics and computer vision community. The library is heavily optimized for linear algebra tasks which the algorithms can exploit to their advantage. Similar to FBOW, Eigen uses SIMD vector processing for a lot of its operations, if it is available on the machine. Using Eigen therefore increases the processing speed without the need to add rather complex parallelization code. On top of these two main input interfaces, a wrapper structure is provided which allows generic programming at places where the specific type of descriptor does not need to be known. Using template metaprogramming, this structure can return information like the descriptor's dimension or its numeric type at compile-time without the need to define the descriptor type explicitly. This can greatly reduce code duplication.

On the output side, the implemented methods generate bag-of-words representations for images which they return as a class called `BowVector`. This class is fundamentally a wrapper around a STL vector, mimicking its interface while adding some domain-specific functionality like vector normalization. The elements of the vector are type aliases named `BowEntry`, pairs of integer and floats representing the words and their histogram values. The database implementation takes this class as its input interface to add images or query the index. The results of a query are returned with another type alias called `QueryResults` which is a vector of pairs of an image specifier and the correspondingly computed score. The image specifier is simply defined as an unsigned integer.

The two main components of the library are the database and the available bag-of-words approaches. For the database, a very generic implementation was developed which is completely independent of the employed method and its vocabulary. It uses a classical inverted index to store and query the image representations, managing a vector of image specifiers for every encountered term. As already explained in Section 3.2, this approach is memory efficient and allows fast image retrieval. Currently, there are three common scoring types available: L_1, L_2 and Cosine Similarity. If desired, adding new

scoring functions should be rather easy. However, our evaluation in Section 5.3.1 has already shown that different scoring types do not influence the results significantly. Because of that we decided to keep the number of available scoring types small. Lastly, the database can be serialized and stored to disk if necessary.

For the bag-of-words methods, we define a fixed interface in the form of an abstract base class called `BowGenerator`. The interface is designed using the template method pattern: The abstract base class defines public methods and additionally declares corresponding purely virtual private methods. The public methods specify the minimum interface each implemented approach provides. The purely virtual private methods force the approaches – child classes inheriting from the base class – to implement the actual functionality. These private implementation methods get called by the base class using the public methods. That way, the interface is very consistent because it is completely separated from the implementation which is up to the particular methods. The base class can also provide wrapping functionality like enforcing certain conditions, valid for all child classes. The interface defines the following functions:

- `train`: Train the approach given a collection of descriptors
- `generateBowVector`: Transform a set of descriptors into a bag-of-words vector
- `saveToFile`: Serialize and save the vocabulary or model to a given file path
- `loadFromFile`: Load and deserialize a vocabulary or model from a given file
- `name`: Return the name of the method as a string

Obviously, derived classes can furthermore implement additional functionality if they see fit. At its current state, the two methods provided by the library are HashBoW and DBoW. They are implemented as child classes inheriting from the `BowGenerator` base class described above. In order to maintain high readability and usability of the library, these classes merely implement the interface. The actual bag-of-words algorithms are encapsulated in implementation classes which are marked by the `detail` namespace to emphasize that they are not meant to be used externally. In the next paragraphs, implementation details of the two available approaches are given.

The HashBoW method is largely implemented as described in Section 4.3.3. The approach is able to generate hash codes up to a length of 32 bits, allowing both very small and very large vocabularies. Additionally, it can be specified whether the generated bag-of-words vectors shall be normalized and if so, which normalization type – L_1 or L_2 – shall be used. Similar to the binary descriptor interface, the hashes are implemented using the STL `bitset` container. A set of randomly sampled bit indices is provided by default such that the approach can be used out-of-the-box without

any prior training. However, the training procedure using entropy maximization of hash codes is implemented as well and can adapt these bit indices. So far, only binary descriptors are supported by the method, although it can be extended rather easily by providing a hashing function for real-valued vectors.

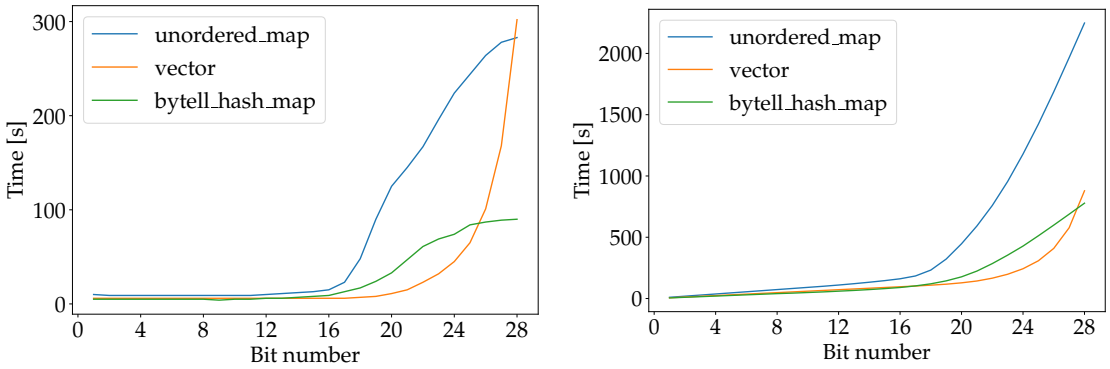
With the publicly available DBoW2 and DBoW3 libraries serving as blueprints, our implementation of DBoW has been rewritten for improved performance and clarity. One of the main differences is the strict separation of vocabulary and database which can now work completely independent from each other. One disadvantage of this design choice is that it removed the possibility to use direct indexing for fast geometrical verification. A possible way to alleviate this issue will be discussed in our outlook of future work in Section 7.2. Apart from the direct index, the full functionality of the original implementation is available, including the k-means++ training procedure and all weighting and normalization schemes. The new DBoW code utilizes more modern C++ features like the `auto` keyword or range-based loops in order to increase readability and intelligibility. The approach supports real-valued as well as binary descriptors. Thanks to our provided wrapper struct for descriptors, the majority of the code could be written generically for both types. The algorithm only needs to differentiate at a few distinct places like mean or distance calculation which the code does automatically by means of template metaprogramming.

6.3. Improvements to Implemented Approaches

In this section, we will analyze weak points of the original method implementations and present our introduced improvements. Obviously, since HashBoW is a new method, there exists no original implementation yet. Nevertheless, while evaluating the prototype implementation of HashBoW, we came across an interesting problem which will therefore be discussed in this section as well. The largest part, however, will involve enhancements of our DBoW rewrite compared to the DBoW2 and DBoW3 libraries.

6.3.1. HashBoW

The mentioned HashBoW problem is a performance issue which came up while implementing the entropy maximization of hash codes. In order to understand it, it is important to know how this training procedure is implemented. The input to the training procedure is a set of training descriptors and the desired number of bit indices



(a) Processing time per bit

(b) Cumulated processing time

Figure 6.1.: Processing times of the HashBoW training procedure using different containers

to calculate. In order to find the optimal bit index – that is to say the bit index which results in hash codes with the highest possible entropy – the algorithm evaluates the distribution of descriptors over all generated hash codes. This can be done by counting the number of descriptors corresponding to the hash codes, storing them in a container and then calculate the entropy from it. It has already been noted previously that the training procedure follows a greedy approach: once an entropy-maximizing bit index is found, it gets fixed and the remaining indices are being processed. This continues until the desired bit number is acquired. Section 4.3.3 also described that the bits cannot be evaluated independently because of potential correlations between indices. Because of this, the number of possible hash codes grows with each fixed bit index, up until it reaches the maximum vocabulary size.

This means that the container storing the counters grows in size as well. The maximum size of the container is however limited by the number of possible hash codes or available descriptors, whatever may be smaller. In most applications, the number of hashes will be smaller at first and then overtake the descriptor count. We have initially chosen an unordered map for this task because it is memory efficient and allows access in constant time. With this implementation, we noticed that the time to compute a new bit index rose dramatically after around 16 bits which is probably due to the fact that the hash table does not fit into cache anymore. Subsequently, we tried to use a simple vector as an alternative. While it holds fast performance for a longer time, it gets very memory inefficient at higher bit counts and the time to calculate the entropy grows exponentially. Finally, we came across some fast hash table implementations by Skarupke [99] which are noticeably faster than STL's unordered map, especially when they do not fit into cache.

Some exemplary processing times for the training procedure depending on the employed container are visualized in Figure 6.1. The compared data structures are STL's unordered map, vector and Skarupke's `bytell_hash_map`, a chaining hash table in a flat array which the author recommends by default. Until a bit number of around 16, all containers perform similar. After that point, the unordered map becomes very slow. The vector implementation is very fast for a longer amount of time, although its processing time explodes after around 20 bits. Skarupke's hash map performs best overall with only being slightly slower than the vector in a range of around 20 to 28 bits. At the same time, it is much more memory efficient than the vector version. We therefore chose to use the `bytell_hash_map` as the best compromise of fast performance over the full range of bits combined with high memory efficiency. For additional information about the particular design choices for this hash table, we refer to the author's blog post [99].

6.3.2. DBoW

Thinking back to the previous evaluation chapter, we have shown that DBoW achieves the highest accuracy of the compared methods. On the other hand, it is rather slow in all phases – training, image addition and database query – of the place recognition task. For our own implementation we focused on accelerating its processing speed while maintaining high accuracy. The main contribution to this goal is the analysis and choice of better suitable data structure in a variety of different places of the implementation.

A first major difference is our use of bitsets and Eigen matrices as containers for the descriptors. Since DBoW3 aims to be directly compatible to all OpenCV descriptors, it exclusively uses its matrix class `cv::Mat`. While this structure is very flexible and can be utilized for a lot of different tasks, our analyses have shown that it is rather slow compared to specialized containers. Specifically, bitsets are considerably faster for operations on bit sequences. Similarly, the Eigen library is optimized for linear algebra, i.e. real-valued matrix or vector manipulation. In the training procedure, calculating the mean and distances between descriptors are two of the most common operations. The distance also needs to be calculated multiple times while transforming features into words. Small run-time improvements for these operations can, therefore, lead to huge performance gains.

In DBoW2 and DBoW3, the bag-of-words representation of images is implemented as a custom class called `BowVector`. The class publicly inherits from a `std::map`, mapping term identifiers to weights. This is a bad design choice in general because STL containers are not meant to be used polymorphically and doing so can lead to

undefined behavior in certain situations (for example when deleting an object through a pointer). Furthermore, as already mentioned in Section 3.3.3, maps are implemented as binary search trees. When constructing a new bag-of-words vector, the original DBoW implementation uses `lower_bound` to find keys and `insert` to add new ones. Both of these operations are logarithmic in complexity. Our version, on the contrary, uses the hash table implementation `std::unordered_map` which allows constant-time search and insertion of elements. After constructing the bag-of-words representation, we push the results to our vector-based class for even faster access and manipulation.

Next, the inverted index in DBoW2/3 is implemented as a vector where each element represents a row of the inverted file. The rows are designed as lists of image identifiers and corresponding weights. Their location in the vector defines the term to which they belong to. The identifiers and weights are stored jointly in a wrapper struct. There are two issues with this design: First, while using a vector for the inverse index enables fast random access of the rows, it is memory inefficient when using a large vocabulary. In this case, the inverted index is typically very sparse, so using a vector with indices corresponding to term identifiers unnecessarily allocates a lot of memory. Second, using a `std::list` for the rows is not a good choice for the task at hand. Doubly-linked lists are especially good for fast random insertion and erasing of elements, operations which are not needed in the DBoW algorithm. The two main operations on the inverted file rows are iteration and pushing elements to the back of the sequence. Vectors are generally significantly faster for these tasks. Because of that, our new implementation again uses an unordered map, mapping terms to vectors of pairs which contain the image identifiers and weights. Using a hash table over a vector as the main container leads to slightly slower access times but greatly reduces memory consumption for large vocabularies. Using vectors for the rows results in considerably better performance than the original version with lists. We also replace the custom struct with a `std::pair` for consistency.

Finally, when calculating the scores of retrieved results, the original DBoW implementations again use a `std::map` with logarithmic complexity for the relevant operations. The run time can easily be improved by exchanging it with an unordered map where the operations have constant complexity. On top of this, the results are again stored in a class which is publicly inheriting from `std::vector`. This should be avoided due to the reasons given above. In addition to all described differences in the employed data structures, we rewrote certain parts of the code in order to improve the clarity and run time of the algorithm.

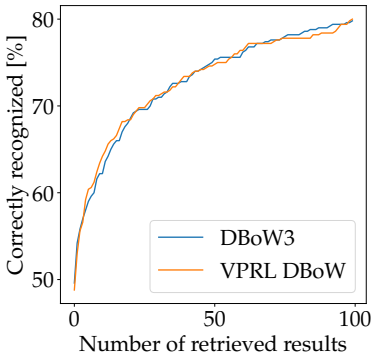


Figure 6.2.: Accuracy comparison of DBoW implementations

Method	Training	Add	Query
DBoW3	74 min 12 s	14.22 s	10.42 s
VPRL DBoW	14 min 41 s	3.12 s	1.98 s

Table 6.1.: Processing times comparison of DBoW implementations

The accuracy and run time of the resulting implementation is evaluated using the benchmarking suite introduced in the previous chapter. It is compared to DBoW3 with the exact same parameter set: a branching factor of 10, a maximum depth of 6, TF-IDF weighting and L_1 scoring. The vocabularies were trained on ORB descriptors extracted from 10 000 validation images of the Places365 dataset. We expect our implementation to have near-identical accuracy – some noise cannot be avoided due to the randomness of the k-means++ algorithm – with significantly improved run times. The results for the Holidays dataset are shown in Figure 6.2 and Table 6.1, our approach being called "VPRL DBoW". Reports for the other available datasets are given in the appendix.

As expected, the accuracy is largely equivalent between the two implementations, apart from some random noise. This result is consistent across all datasets. Regarding the run time, our implementation performs better in all phases of the benchmark. Training speed is improved by a factor of 5. The main reason for this is the better performance of bitsets compared to OpenCV matrices, since the vocabulary parameters are the same. We assume a similar result for real-valued descriptors, although we did not test it explicitly. In the processing phase, our implementation outperforms DBoW3 by a factor of 4 to 5 in the addition of images and 5 to 7 in querying the database. Again, with some minor variations, these results are consistent across datasets. Here, all previously mentioned improvements apply.

In summary, it can be stated that we achieved our goal of developing the fastest standalone DBoW implementation. As a final note, for the sake of transparency, we also want to draw the reader's attention to some arguments which could be made against this statement. First, direct indexing is a main functionality of DBoW which has not been implemented in our version yet. However, our focus lies on place recognition

as a pure image-retrieval task where the direct index as a way to geometrically verify results is not considered. It also had no influence on the run-time evaluation because we disabled the functionality for DBoW3. Another conceivable point of criticism is that a conversion of descriptors is necessary when OpenCV is used for feature extraction. Because DBoW3 is designed with OpenCV descriptors in mind, additional processing time is needed. While this is true, our tests have shown that our implementation achieves significantly faster run time even with this conversion procedure considered. In all fairness, one should also note that DBoW2 allows the use of different containers for storing and manipulating descriptors which can result in improved performance. Nevertheless, their descriptor implementation is not unified and the other previously mentioned issues still apply.

7. Conclusion

We conclude the thesis with a summary and possible future work items. At first, we will reiterate the problem statement and give a quick overview of the contents of each chapter. Next, we will summarize the results of our three main contributions, namely HashBoW, the benchmarking suite and the new place recognition library. Finally, we will suggest potential improvements for each of these contributions.

7.1. Summary

In this thesis we investigated the task of visual place recognition as a way of improving efficiency and robustness of SLAM or Structure-from-Motion systems. The regarded methods work on local feature descriptors due to their natural connection to sparse keypoint-based 3D reconstruction. We considered the place recognition challenge as a pure retrieval problem from unordered collections of images without any additional verification steps. This highly generic formulation allows to transfer the obtained results to other related application areas like image similarity search. The thesis yielded three main contributions: First, we gave an overview of state-of-the-art solutions for the task and subsequently introduced a novel highly-efficient approach inspired by those. Second, we developed a powerful benchmarking suite which was used to analyze, evaluate and compare the presented methods. Third and last, we provided an efficient and easy-to-use open-source library for visual place recognition which contains multiple algorithms based on the bag-of-words model.

The thesis started with a motivational introduction to the topic, leading to the problem statement and structure of the thesis. In Chapter 2 we first looked at research areas which are related to visual place recognition but do not receive further attention in our work. After that, we investigated the usage of place recognition approaches in some popular open-source SLAM and SfM frameworks. Additionally, we gave a brief overview of commercial applications. Chapter 3 then provided rich background information on multiple topics which are essential to understand the thesis at hand. We first presented the various ways of describing images with a particular focus on local

feature extraction. We explained theoretical aspects and introduced specific methods relevant to this work. Furthermore, we presented the foundations of global image descriptors and noted some alternative description techniques as well. Afterwards, we covered the subject of information retrieval because it forms the basis for many place recognition approaches. We described its terminology and showed several weighting and similarity scoring strategies. In addition, we gave a quick introduction to entropy, a pivotal measure in the field of information theory. At last, we talked about efficient data structures and their impact in computer science. Here we set the focus on tree structures and hashing, since these topics are highly relevant to efficient place recognition implementations. We also listed some particular containers from the C++ standard library which found use in this thesis. Subsequently, in Chapter 4 we presented theoretical aspects of visual place recognition, including a definition of the task, the typical structure and major challenges. Then we introduced a specific implementation scheme for the task, namely the bag-of-words model. Finally, we explicated three state-of-the-art place recognition approaches: the hierarchical bag-of-words implementation DBoW, the binary search tree HBST and our novel hashing-based HashBoW method. In Chapter 5, these solutions were subsequently evaluated in a newly developed benchmarking suite. There, we first outlined the structure and workflow of the benchmarking library, narrowed down the evaluation domain and lastly reported the actual results of the evaluation. For the evaluation, we performed an analysis about each method's parameters, their training procedures and the influence of the choice of the employed feature extractor. At last, we provided a comparison of the optimally parameterized approaches. Based on the findings of the previous chapter, we created an open-source library containing promising place recognition approaches. Last of all, this library was introduced in Chapter 6 where we also presented some implementation improvements to the included solutions.

With our new benchmarking suite, we achieved the goal of providing an easy-to-use and highly extensible tool box for the evaluation of visual place recognition methods. At its current state, it contains three different datasets, three place recognition methods and a basic evaluation script with accuracy and run-time metrics. Our performed evaluation reveals multiple interesting findings: First, it affirms the parameter choices for DBoW and HBST suggested in the original publications. Besides that, it provides a comprehensive analysis about the impact of various possible changes to these parameters. The results also helped to define a default parameter set for the new HashBoW method. The evaluation proved that the feature descriptor choice can have a big influence on the performance of the place recognition methods and that ORB works best for most approaches. The final method comparison showed that DBoW is the most accurate

solution but suffers from rather inefficient implementations. HashBoW on the contrary is a very fast algorithm but it is not as accurate as the other approaches. Finally, HBST represents a very good compromise of accuracy and efficiency.

The novel place recognition library has been written with similar goals in mind as the benchmarking suite. It is well-documented and has a high code quality so that it can easily be used by others. Fixed interfaces make it possible to add new place recognition methods without needing to adapt other parts of the codebase. Furthermore, it is very lightweight due to its header-only design. With HashBoW and DBoW, we provide two solutions out of the box. The DBoW implementation has been completely rewritten in order to significantly increase its efficiency. Our final evaluation shows that our new version has approximately 5 times higher processing speed than the current most popular implementation.

7.2. Future Work

In this final section of the thesis we want to state some possible future works for our different contributions, the first being our benchmarking suite. A straightforward extension of the library would be to add more datasets, place recognition methods and evaluation metrics. The currently available datasets are all comparatively small or focused on one particular scene type like buildings. In order to make the evaluation more expressive and challenging, other datasets or even distractor sets – containing images not relevant to the available queries – could be added. In addition, the included list of place recognition methods is by no means exhaustive. There exist many more promising solutions like FAB-MAP [17], MILD [33] or NetVLAD [5], all with their own potential benefits and drawbacks. Furthermore, the evaluation script could contain many more interesting metrics, for example precision-recall curves. The run time could also be measured regularly in order to provide information about how the processing time grows over the course of the application. Lastly, completely different workflows could be implemented. So far, the benchmarking suite only evaluates place recognition as image retrieval from unordered collections of images. However, other pipelines like for example loop closure detection with additional verification steps could also be an interesting direction to evaluate.

The introduced place recognition library could be extended in many ways as well. An obvious idea would be to add new bag-of-words generators. Here, FAB-MAP is again a prime example which has been omitted for time reasons. Similar to these different bag-of-words approaches, one could also think about multiple database

implementations. The current solution is very generic and therefore not particularly efficient or accurate. More specific versions with a fixed vocabulary size could for example improve the efficiency of the retrieval process. Another possibility would be to provide a specialized database for DBoW which can make use of the direct-indexing functionality in order to provide the full feature set of the original implementation. Moreover, a database allowing adaptable vocabularies would enable online learning of the approaches. Finally, the current database can potentially be enhanced as well by evaluating the efficiency of the employed data structures. As we have already shown in previous chapters, minor adjustments – like exchanging the default containers to more specialized ones – can lead to big run-time improvements.

As a final point, we want to talk about our novel hashing-based place recognition solution HashBoW. In its current form, it is a highly efficient method in both run time and memory consumption but suffers from its subpar accuracy. One possible reason for this is our choice of a simple one-step bit sampling as the hashing function. While this hash function is in fact locality-sensitive, the gap between the relevant probabilities given in Definition 1 of Section 3.3.2 may be quite small and therefore impair the expressiveness of the approach. Because of this issue, a common solution to improve performance – proposed in a popular publication by Gionis *et al.* [28] – is to apply multiple hash functions and combine the results in a second hash step. Other possibilities include the choice of a completely different hash function or the data-driven locality-preserving hashing. Last of all, HashBoW is currently compatible to binary descriptors only, so extending it to real-valued descriptors could be a valuable future task as well.

Appendices

A. Benchmarking Suite

A.1. Code

The code of the benchmarking suite is provided in a repository, hosted on the chair's GitLab server at `https://gitlab.vision.in.tum.de/vpr/vpr_benchmark`. The final version for the thesis is tagged as `thesis-final-version` (commit hash: `da65f26e`).

A.2. Repository Structure

For reasons of clarity and comprehensibility, we provide a short overview of the repository and its most important files:

```
vpr_benchmark
├── cmake/ ..... Additional CMake files
├── data_preparation/ ..... Scripts for setting up the datasets
├── evaluation/
│   └── vpr_performance_evaluation.ipynb ..... Jupyter notebook for evaluation
├── include/vpr_benchmark/ ..... C++ header files
├── scripts/
│   ├── DBoW3.patch ..... Compatibility patch
│   ├── clang-format-all.sh ..... Code formatting
│   └── setup-benchmark.sh ..... Benchmark initialization
├── src/ ..... C++ source files
├── third_party/ ..... Third-party libraries
├── CMakeLists.txt ..... C++ setup
├── README.md ..... Documentation
└── method-parameters.yaml ..... Method parameter settings
```

Note that the list of files shown in the tree visualization above is not exhaustive.

B. Visual Place Recognition Library

B.1. Code

Like the benchmarking suite, the code of our place recognition library is provided in a repository at `https://gitlab.vision.in.tum.de/vpr/vpr_library`. The final version for the submission of the thesis is tagged as `thesis-final-version` (commit hash: `083ac48c`) as well.

B.2. Repository Structure

Again, we give a short overview of the repository structure and a list of the most important files (which is not exhaustive):

```
vpr_library
├── cmake/ ..... Additional CMake files
├── include/vpr_library/
│   ├── internal/ ..... Implementation details
│   ├── bow_generator.hpp ..... BowGenerator interface
│   ├── bow_vector.hpp ..... BowVector data structure
│   ├── common_types.h ..... Common definitions
│   ├── database.hpp ..... BowDatabase implementation
│   ├── dbow.hpp ..... DBoW implementation
│   ├── descriptor.hpp ..... Descriptor data structures
│   └── hashbow.hpp ..... HashBoW implementation
├── scripts/
│   └── clang-format-all.sh ..... Code formatting
├── src/ ..... Test / demo application
├── test/ ..... Unit tests
├── third_party/ ..... Third-party libraries
├── CMakeLists.txt ..... Project setup
└── README.md ..... Documentation
```

C. Evaluation Data

The full evaluation data and some pretrained vocabularies are stored in a third repository, located at `https://gitlab.vision.in.tum.de/vpr/vpr_data`. The repository contains the raw results of the benchmarking suite as well as the evaluation notebooks used to create the figures in chapters 5 and 6 for all three available datasets. In order to denote the version which is valid for the submission of the thesis, the tag `thesis-final-version` (commit hash: `aba4c9ec`) is used once again.

List of Figures

List of Tables

Bibliography

[1] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "Kaze features," in *Computer Vision – ECCV 2012*, Springer Berlin Heidelberg, 2012, pp. 214–227.

[2] P. F. Alcantarilla, J. Nuevo, and A. Bartoli, "Fast explicit diffusion for accelerated features in nonlinear scale spaces," in *Proceedings of the British Machine Vision Conference*, BMVA Press, 2013.

[3] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, 2006, pp. 459–468.

[4] A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer, "Fast and incremental method for loop-closure detection using bags of visual words," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1027–1037, 2008.

[5] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5297–5307.

[6] H. Badino, D. Huber, and T. Kanade, "Real-time topometric localization," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1635–1642.

[7] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Computer Vision – ECCV 2006*, Springer Berlin Heidelberg, 2006, pp. 404–417.

[8] M. Bosse and R. Zlot, "Keypoint design and evaluation for place recognition in 2d lidar maps," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1211–1224, 2009.

[9] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.

[10] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *Computer Vision – ECCV 2010*, Springer Berlin Heidelberg, 2010, pp. 778–792.

[11] Y. Cao, C. Wang, L. Zhang, and L. Zhang, "Edgel index for large-scale sketch-based image search," in *CVPR 2011*, 2011, pp. 761–768.

[12] D. Chekhlov, W. W. Mayol-Cuevas, and A. Calway, "Appearance based indexing for relocalisation in real-time visual slam," in *Proceedings of the British Machine Vision Conference*, BMVA Press, 2008.

[13] J. Cheng, C. Leng, J. Wu, H. Cui, and H. Lu, "Fast and accurate image matching with cascade hashing for 3d reconstruction," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1–8.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009, ISBN: 978-0-262-03384-8.

[15] I. Corporation, *Threading building blocks*, `https://github.com/intel/tbb`, 2017.

[16] M. Cummins and P. Newman, "Probabilistic appearance based navigation and loop closing," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 2042–2048.

[17] ——, "FAB-MAP: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893.

[19] M. M. Deza and E. Deza, *Encyclopedia of Distances*, 4th ed. Springer Berlin Heidelberg, 2016, 756 pp., ISBN: 978-3-662-52844-0.

[20] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "SegMatch: Segment based place recognition in 3d point clouds," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5266–5272.

[21] M. Eitz, K. Hildebrand, T. Boubekeur, and M. Alexa, "Sketch-based image retrieval: Benchmark and bag-of-features descriptors," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 11, pp. 1624–1636, 2011.

[22] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Computer Vision – ECCV 2014*, Springer Cham, 2014, pp. 834–849.

[23] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: A survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[24] D. Gálvez-López and J. D. Tardós, "Real-time loop detection with bags of binary words," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 51–58.

[25] ——, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[26] X. Gao, R. Wang, N. Demmel, and D. Cremers, "LDSO: Direct sparse odometry with loop closure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2198–2204.

[27] E. Garcia-Fidalgo and A. Ortiz, "Vision-based topological mapping and localization methods: A survey," *Robotics and Autonomous Systems*, vol. 64, pp. 1–20, 2015.

[28] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.

[29] L. C. González, R. Moreno, H. J. Escalante, F. Martínez, and M. R. Carlos, "Learning roadway surface disruption patterns using the bag of words representation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 2916–2928, 2017.

[30] Google Inc., *Google Test*, `https://github.com/google/googletest`, 2019.

[31] W. S. Grant and R. Voorhies, *Cereal - a C++11 library for serialization*, `http://uscilab.github.io/cereal/`, 2017.

[32] G. Guennebaud, B. Jacob, *et al.*, *Eigen v3*, `http://eigen.tuxfamily.org`, 2010.

[33] L. Han and L. Fang, "MILD: Multi-index hashing for appearance based loop closure detection," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, 2017, pp. 139–144.

[34] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of Fourth Alvey Vision Conference*, 1988, pp. 147–151.

[35] Hawk-Eye Innovations Ltd, *Hawk-eye*, 2018. [Online]. Available: `https://www.hawkeyeinnovations.com/` (visited on 2020-06-03).

[36] K. L. Ho and P. Newman, "Detecting loop closure with scene sequences," *International Journal of Computer Vision*, vol. 74, pp. 261–286, 2007.

[37] J. Huai, Y. Qin, F. Pang, and Z. Chen, "Segway DRIVE benchmark: Place recognition and SLAM data collected by a fleet of delivery robots," *arXiv preprint arXiv:1907.03424*, 2019.

[38] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, 1998, pp. 604–613.

[39] ISO, *ISO/IEC 14882:2017 Information technology — Programming languages — C++*, 5th ed. Geneva, Switzerland: International Organization for Standardization, Dec. 2017, p. 1605. [Online]. Available: `https://www.iso.org/standard/68564.html` (visited on 2020-06-03).

[40] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Computer Vision – ECCV 2008*, Springer Berlin Heidelberg, 2008, pp. 304–317.

[41] Q. Jin, C. Li, S. Chen, and H. Wu, "Speech emotion recognition with acoustic and lexical features," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4749–4753.

[42] F. Jing, M. Li, H.-J. Zhang, and B. Zhang, "A unified framework for image retrieval using keyword and visual features," *IEEE Transactions on Image Processing*, vol. 14, no. 7, pp. 979–989, 2005.

[43] D. E. Knuth, *The Art of Computer Programming: Volume 3: Sorting and Searching*, 2nd ed. USA: Addison-Wesley, 1998, ISBN: 978-0-201-89685-5.

[44] S. Krig, *Computer Vision Metrics: Survey, Taxonomy, and Analysis*. Apress, Berkeley, CA, 2014, ISBN: 978-1-4302-5930-5.

[45] R. M. Kumar and K. Sreekumar, "A survey on image feature descriptors," *International Journal of Computer Science and Information Technologies*, vol. 5, pp. 7668–7673, 2014.

[46] P. Lamon, I. Nourbakhsh, B. Jensen, and R. Siegwart, "Deriving and matching image fingerprint sequences for mobile robot localization," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, vol. 2, 2001, pp. 1609–1614.

[47] M. Lee, H. Yang, D. Han, and C. Yu, "Crowdsourced radiomap for room-level place recognition in urban environment," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2010, pp. 648–653.

[48] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, 3rd ed. Cambridge University Press, 2020, ISBN: 978-1-108-47634-8.

[49] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, 2011, pp. 2548–2555.

[50] D. G. Lowe, *The computer vision industry*, 2015. [Online]. Available: `https://www.cs.ubc.ca/~lowe/vision.html` (visited on 2020-06-03).

[51] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157.

[52] ——, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[53] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2016.

[54] E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," in *Computer Vision – ECCV 2010*, Springer Berlin Heidelberg, 2010, pp. 183–196.

[55] A. Majdik, D. Gálvez-López, G. Lazea, and J. A. Castellanos, "Adaptive appearance based loop-closing in heterogeneous environments," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 1256–1263.

[56] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008, 506 pp., ISBN: 978-0-521-86571-5.

[57] Merriam-Webster, *Place*, in *Merriam-Webster.com*, 2020. [Online]. Available: `https://www.merriam-webster.com/dictionary/place` (visited on 2020-06-03).

[58] ——, *Recognize*, in *Merriam-Webster.com*, 2020. [Online]. Available: `https://www.merriam-webster.com/dictionary/recognize` (visited on 2020-06-03).

[59] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, pp. 43–72, 2005.

[60] M. J. Milford and G. F. Wyeth, "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1643–1649.

[61] M. J. Milford, G. F. Wyeth, and D. Prasser, "RatSLAM: A hippocampal model for simultaneous localization and mapping," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 1, 2004, pp. 403–408.

[62] P. Moulon, P. Monasse, R. Perrot, and R. Marlet, "OpenMVG: Open multiple view geometry," in *Reproducible Research in Pattern Recognition*, Springer Cham, 2016, pp. 60–74.

[63] R. Muñoz-Salinas, *DBoW3*, `https://github.com/rmsalinas/DBow3`, 2017.

[64] R. Muñoz-Salinas and R. Medina-Carnicer, "UcoSLAM: Simultaneous localization and mapping by fusion of keypoints and squared planar markers," *Pattern Recognition*, p. 107 193, 2020, ISSN: 0031-3203.

[65] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[66] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based SLAM," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 846–853.

[67] ——, "ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[68] A. C. Murillo and J. Kosecka, "Experiments in place recognition using gist panoramas," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, 2009, pp. 2196–2203.

[69] T. Naseer, M. Ruhnke, C. Stachniss, L. Spinello, and W. Burgard, "Robust visual SLAM across seasons," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 2529–2535.

[70] T. Nicosevici and R. Garcia, "Automatic visual bag-of-words for online robot navigation and mapping," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 886–898, 2012.

[71] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 2161–2168.

[72] M. Nowicki and J. Wietrzykowski, "Low-effort place recognition with WiFi fingerprints using deep learning," in *Automation 2017*, vol. 550, Springer Cham, 2017, pp. 575–584.

[73] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy, "Berkeley MHAD: A comprehensive multimodal human action database," in *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, 2013, pp. 53–60.

[74] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, pp. 145–175, 2001.

[75] ——, "Building the gist of a scene: The role of global image features in recognition," *Progress in Brain Research*, vol. 155, pp. 23–36, 2006.

[76] OpenCV, *OpenCV: Feature detection and description*, 4.1.2, 2019. [Online]. Available: `https://docs.opencv.org/4.1.2/d5/d51/group__features2d__main.html` (visited on 2020-06-03).

[77] S. Pancoast and M. Akbacak, "Bag-of-audio-words approach for multimedia event classification," in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[78] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[79] ——, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[80] J. Philbin, M. Isard, J. Sivic, and A. Zisserman, "Descriptor learning for efficient retrieval," in *Computer Vision – ECCV 2010*, Springer Berlin Heidelberg, 2010, pp. 677–691.

[81] A. Pronobis, B. Caputo, P. Jensfelt, and H. I. Christensen, "A discriminative approach to robust visual place recognition," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 3829–3836.

[82] P. O. Ribeiro, M. M. dos Santos, P. L. Drews, S. S. Botelho, L. M. Longaray, G. G. Giacomo, and M. R. Pias, "Underwater place recognition in unknown environments with triplet based acoustic image retrieval," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 524–529.

[83] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An open-source library for real-time metric-semantic localization and mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[84] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006*, Springer Berlin Heidelberg, 2006, pp. 430–443.

[85] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[86] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "SLAM++: Simultaneous localisation and mapping at the level of objects," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1352–1359.

[87] T. Sattler, T. Weyand, B. Leibe, and L. Kobbelt, "Image retrieval for image-based localization revisited," in *Proceedings of the British Machine Vision Conference*, BMVA Press, 2012.

[88] D. Schlegel and G. Grisetti, "Visual localization and loop closing using decision trees and binary features," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4616–4623.

[89] ——, "HBST: A hamming distance embedding binary search tree for feature-based visual place recognition," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3741–3748, 2018.

[90] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart, "Maplab: An open framework for research in visual-inertial mapping and localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1418–1425, 2018.

[91] J. L. Schönberger, T. Price, T. Sattler, J.-M. Frahm, and M. Pollefeys, "A vote-and-verify strategy for fast spatial verification in image retrieval," in *Computer Vision – ACCV 2016*, Springer Cham, 2017, pp. 321–337.

[92] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4104–4113.

[93] T. Schöps, T. Sattler, and M. Pollefeys, "BAD SLAM: Bundle adjusted direct RGB-D SLAM," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 134–144.

[94] H. Schreiner, P. Top, *et al.*, *CLI11*, `https://github.com/CLIUtils/CLI11`, 2017.

[95] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[96] J. Shi and C. Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[97] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations*, 2015.

[98] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings Ninth IEEE International Conference on Computer Vision*, vol. 2, 2003, pp. 1470–1477.

[99] M. Skarupke, *A new fast hash table in response to google's new fast hash table | probably dance*, May 28, 2018. [Online]. Available: `https://probablydance.com/2018/05/28/a-new-fast-hash-table-in-response-to-googles-new-fast-hash-table/` (visited on 2020-06-03).

[100] S. M. Smith and J. M. Brady, "SUSAN—a new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, pp. 45–78, 1997.

[101] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3d," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 835–846, 2006.

[102] B. Steder, G. Grisetti, and W. Burgard, "Robust place recognition for 3d range data based on point features," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 1400–1405.

[103] H. Strasdat, A. J. Davison, J. M. Montiel, and K. Konolige, "Double window optimisation for constant time visual SLAM," in *2011 International Conference on Computer Vision*, 2011, pp. 2352–2359.

[104] S. Sumikura, M. Shibuya, and K. Sakurada, "OpenVSLAM: A versatile visual SLAM framework," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2292–2295.

[105] N. Sünderhauf and P. Protzel, "BRIEF-Gist – closing the loop by simple means," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 1234–1241.

[106] N. Sünderhauf, S. Shirazi, A. Jacobson, F. Dayoub, E. Pepperell, B. Upcroft, and M. Milford, "Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free," *Proceedings of Robotics: Science and Systems XII*, 2015.

[107] C. Sweeney, *Theia multiview geometry library: Tutorial & reference*, `http://theia-sfm.org`, 2015.

[108] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010, ISBN: 978-1-84882-934-3.

[109] S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 2018, pp. 1–10.

[110] E. C. Tolman, "Cognitive maps in rats and men," *Psychological Review*, vol. 55, no. 4, p. 189, 1948.

[111] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla, "24/7 place recognition by view synthesis," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1808–1817.

[112] I. Ulrich and I. Nourbakhsh, "Appearance-based place recognition for topological localization," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 1023–1029.

[113] P. Viola and M. Jones, "Robust real-time object detection," *International Journal of Computer Vision*, 2001.

[114] M. M. Waldrop, "The chips are down for moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[115] J. Wang and X.-S. Hua, "Interactive image search by color map," *ACM Transactions on Intelligent Systems and Technology*, vol. 3, no. 1, 2011.

[116] J. Wang and Y. Yagi, "Efficient topological localization using global and local feature matching," *International Journal of Advanced Robotic Systems*, vol. 10, no. 3, p. 153, 2013.

[117] J. Wolf, W. Burgard, and H. Burkhardt, "Robust vision-based localization by combining an image-retrieval system with monte carlo localization," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 208–216, 2005.

[118] Q. Yu, F. Liu, Y.-Z. Song, T. Xiang, T. M. Hospedales, and C.-C. Loy, "Sketch me that shoe," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 799–807.

[119] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1452–1464, 2017.