Learning and Deploying Local Features

Linguang Zhang

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Professor Szymon M. Rusinkiewicz

June 2019

Abstract

Many computer vision applications including image matching, image-based reconstruction and localization rely on extracting and matching robust local features. A typical local feature pipeline first detects repeatable keypoints in the image (i.e., keypoint detector), and then computes a short vector to uniquely describe each keypoint (i.e., feature descriptor). Both the keypoint detector and the feature descriptor are conventionally hand-crafted based on what is intuitive to the designer. For example, corners or blobs are popular choices of keypoints, and the image gradient is a useful clue for descriptors. However, it is often difficult to define these principles to accommodate various applications. In this thesis, we study data-driven approaches which can more easily tailor the local feature pipeline for target applications. We start with a mobile robotics application that leverages local features extracted from ground texture images to achieve high-precision global localization. The second part of the thesis addresses the problem that existing keypoint detectors that are optimized for natural images suffer from sub-optimal performance on texture images. We therefore learn a keypoint detector specifically for each type of texture using a deep neural network. Our detector automatically learns to identify keypoints that are distinctive in the target texture rather than relying on a set of pre-defined rules. Finally, we focus on a non-parametric approach for learning feature descriptors. Many well-performing local feature descriptors are trained using a triplet loss that includes a tunable margin, which limits its ability to generalize to other types of data and problems. We propose to replace the hard margin with a soft margin that self-tunes as learning progresses. To summarize, we first demonstrate through a novel visual-based localization system where a customized local feature pipeline is critical. Then, we tackle both the keypoint detector and the feature descriptor with generalizable data-driven approaches.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Szymon Rusinkiewicz, for his guidance, patience and encouragement throughout my PhD study. His research insights, positive attitude, and immense knowledge will continue to inspire me in my future journeys.

Besides my advisor, I would like to thank the rest of my committee members: Jia Deng, Adam Finkelstein, Thomas Funkhouser, and Olga Russakovsky, for their time and insightful feedback.

I am grateful to Adam Finkelstein for his contributions to Chapter 2 and 3, Olga Russakovsky for her helpful suggestions on Chapter 3, and all anonymous reviewers for their invaluable comments. I was fortunate to have the opportunities to work with many fantastic industrial researchers, including Vladlen Koltun, Alejandro Troccoli, Jan Kautz, Beibei Liu, and Robert Wang. I would like to thank them for generously sharing their knowledge and making my summer internships both eye-opening and rewarding. I am indebted to my undergraduate advisor, Chi-Keung Tang, for guiding me into the field of vision and graphics. He never ceases to inspire and encourage me during the hard times. I am also thankful to my labmates from the PIXL group. My past five years at Princeton would not be such a wonderful experience without them. I thank the coffee/tea squad for helping me keep hydrated. Special thanks go to Jeffrey Dwoskin for providing the LaTeX template, which is used for the creation of this work. My research is supported by NSF grants IIS1421435, CHS-1617236 and IIS-1815070.

A very special gratitude goes out to Elaine Zheng for her continuous understanding, care, and encouragement. She has witnessed my ups and downs during my study and has always been a great listener and companion.

Finally and most importantly, I would like to dedicate this thesis to my parents, for their endless love and support.

To my parents.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nearly every recently manufactured smartphone is equipped with digital cameras, and acquiring images has become extremely simple. Many smartphone users enjoy taking photos to record precious moments and sharing with others via the internet. In computer vision, being able to further leverage a considerable amount of photos captured by people from all over the world every day and build new applications is an important task. The well-known Photo Tourism project [85] utilizes internet photos and reconstructs the scene in 3D, which allows the user to view the photos while exploring tourist attractions in 3D. In a typical computer vision application like this, the first step of processing a large collection of images is to associate them through visual cues.

Modern smartphone cameras usually store the photos with geographical information (i.e., GPS coordinates), which can be used as an initial cue to associate images captured near the same location, but oftentimes the geographical information is too coarse to be used in applications that demand higher accuracy. With recent crowdsourcing tools such as Amazon Mechanical Turk (MTurk), obtaining detailed human labels for image contents has become feasible, although the cost may grow dramatically with the labelling difficulty as well as the number of images to be labelled. Due

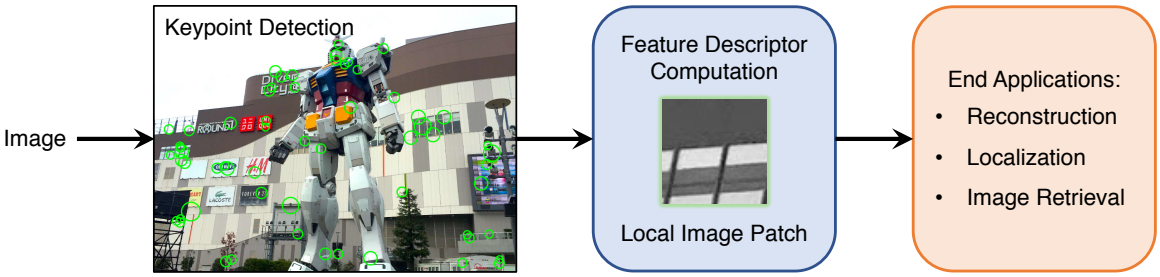
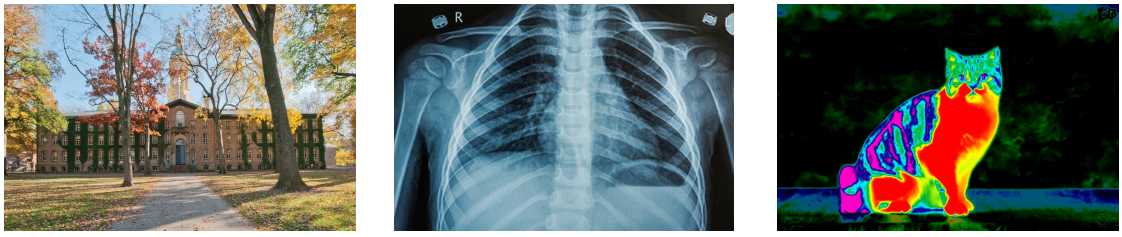


Figure 1.1: A standard local feature pipeline.

to the above constraints, automatic computational methods that can help establish the relationships among images are highly preferred. In almost any computer vision application, the most efficient way to abstract and further process an image is to find the right "feature". This thesis studies *local features*, as opposed to *global features* which summarize the entire image. Local features are typically more robust against geometric transformations as they are often only associated with local image regions. Therefore, image-based 3D reconstruction systems [85, 103, 79] conventionally rely on local features to register images taken from multiple views, and estimate their camera poses. Figure 1.1 shows a standard pipeline of local feature extraction and processing. Given an input image, we first identify image locations with locally distinctive appearances (i.e., keypoints). Then we compute the feature descriptor for each detected keypoint using a local image patch cropped around it. Finally, computer vision applications such as 3D reconstruction, localization and image retrieval can be driven by the detected keypoints along with the computed descriptors. In this thesis, we focus on improving both the keypoint detector and the feature descriptor in addition to a real-world application that immediately benefits from our improvement.

In most scenarios, a critical property of a good keypoint detector is high *repeatability*. A keypoint is called repeatable when the 3D location it corresponds to in the physical world can be detected again when observed from a novel camera view. Since a high repeatability is desired, existing keypoint detectors usually look for the center of every local region that is highly distinguishable in comparison to its neighboring



(a) natural image[1] (b) medical image[2] (c) infrared image[3]

Figure 1.2: Examples images from different domains.

regions. Depending on the application, a keypoint might need to be associated with a scale to account for the distance from the camera to the physical 3D point. Conventionally, keypoint detectors are handcrafted based on empirical principles, which assign a "keypointness" score to each pixel. Pixel locations with high scores are treated as keypoints. For example, corners and blobs are typically regarded as good keypoints because they are locally distinctive and more likely to be identified when observed from a different camera view. For a long period of time, these empirical principles have been proven effective for designing keypoint detectors. While existing handcrafted keypoint detectors have gained success in many classical computer vision problems (e.g., structure-from-motion [3], panorama stitching [14]), they are tuned to work best for "natural images". In other words, handcrafted keypoint detectors are mostly designed based on natural image statistics. Figure 1a shows an example of natural image; it is reasonable to assume that keypoint detectors designed for images of this kind could work well on many other daily images. Certain real-world computer vision applications, however, limit themselves to particular usage scenarios (e.g., indoor/outdoor only) or even specific types of image: Figure 1b and 1c are examples of medical and infrared images, respectively. Existing keypoint detectors handcrafted for natural images would not necessarily work properly on these domain-specific images, since they differ drastically from natural images. One possible solution is to

[1]Source: `https://www.princeton.edu`
[2]Source: `https://www.fda.gov`
[3]Source: `https://www.scienceabc.com`

learn a keypoint detector in a data-driven manner if we have access to a sufficient quantity of images of the domain of interest.

In Chapter 2, we introduce an image-based global localization system named Micro-GPS that is accurate to a few millimeters and performs reliable localization both indoors and outside. The key idea is to capture and index distinctive local keypoints in *ground textures*. This is based on the observation that ground textures including wood, carpet, tile, concrete, and asphalt may look random and homogeneous, but all contain cracks, scratches, or unique arrangements of fibers. These imperfections are persistent, and can serve as local features. Our system incorporates a downward-facing camera to capture the fine texture of the ground, together with an image processing pipeline that locates the captured texture patch in a compact database constructed offline. We demonstrate the capability of our system to robustly, accurately, and quickly locate test images on various types of outdoor and indoor ground surfaces.

One drawback to the Micro-GPS system is that the reliability is limited by the performance of existing feature detectors on textures, as opposed to natural images. In Chapter 3, we propose an effective and scalable method for learning feature detectors for textures, which combines an existing "ranking" loss with an efficient fully-convolutional architecture as well as a new training-loss term that maximizes the "peakedness" of the response map. We demonstrate that the learned detector is more repeatable than existing methods, leading to improvements in the Micro-GPS system.

The next step after keypoint detection is feature description. Centered on each detected keypoint, a small image patch with the radius proportional to the scale is cropped and used to compute the feature descriptor. A feature descriptor is often represented as a short real-valued vector, which can be used to compare the similarity between image patches, typically via the Euclidean distance. Storing real-valued vectors and computing the Euclidean distance, however, can be expensive for real-

time applications or embedded systems. Alternatively, one can represent the feature descriptor as a compact binary string, which leverages the Hamming distance to compare, but is often less effective than a real-valued descriptor of the same length. A well performing feature descriptor normally satisfies the following condition: image patches associated with the same physical 3D point should be consistently described although they could appear differently due to the variation of viewpoint and illumination. Feature descriptors are also conventionally handcrafted. For instance, in order to gain robustness against illumination changes, some descriptors leverage image gradient directions instead of raw pixel values.

In recent years, descriptors based on machine learning techniques have significantly outperformed their handcrafted competitors in situations where the test data shares similar statistics with the training data. Since the training data often only consists of image patches which are relatively easy to obtain in advance, data-driven approaches are highly practical in real applications. The general idea of performing such training is to encourage the descriptors of matching patches to be similar while forcing those of non-matching patches to be different. One of the most commonly used loss function that exercises this idea is the triplet loss. The triplet loss utilizes a hyper-parameter called margin to select only "hard" datapoints for training. The margin, however, is often empirically chosen or determined through computationally expensive validation, and stays unchanged during the entire training session. In Chapter 4, we propose a simple yet effective method to overcome the above limitations. The core idea is to replace the hard margin with a non-parametric soft margin, which is dynamically updated. The major observation is that the difficulty of a triplet can be inferred from the cumulative distribution function of the triplets' signed distances to the decision boundary. We demonstrate through experiments on both real-valued and binary local feature descriptors that our method leads to state-of-the-art performance on popular benchmarks, while eliminating the need to determine the best margin.

1.1 Contributions

We summarize the contributions of this thesis by chapter:

Chapter 2: High-Precision Localization Using Ground Texture

- Describing a low-cost global localization system based on ground textures.
- Capturing and making available datasets of seven indoor and outdoor ground textures.
- Describing an architecture for building a global map of commonplace texture, then expanding it at a later date to incorporate new data.
- Investigating the design decisions necessary for practical matching in texture-like images, as opposed to natural images. This includes the choice of keypoint detector, feature descriptor, strategies for reducing storage costs, and a robust voting procedure that can find inliers with high reliability.
- Demonstrating real-world applications of precise localization, including capturing the locations of footsteps as well as an autonomous vehicle that can use Micro-GPS to record a path and then follow it with sub-centimeter accuracy.

Chapter 3: Learning to Detect Features in Texture Images

- Designing a fully-convolutional network architecture that can be efficiently applied on a full-sized image without separate evaluation on each pixel.
- Describing a method to maximize the *peakedness* of the response map and proving that it is critical to improving the repeatability of the learned detector.
- Demonstrating on Micro-GPS that the learned detector is more effective than a handcrafted alternative.

Chapter 4: Learning Local Descriptors with Dynamic Soft Margin

- Proposing a novel loss function based on dynamic soft margin that can serve as a *drop-in replacement* for the existing triplet loss without user-tunable parameters.
- Unifying the real-valued and binary descriptor learning pipelines to adopt the same loss function for training.
- Demonstrating that the proposed loss advances the state of the art on *both* real-valued and binary descriptor learning.

Chapter 2

High-Precision Localization Using Ground Texture

2.1 Introduction

The Global Positioning System (GPS) receiver has become an essential component of both hand-held mobile devices and vehicles of all types. Applications of GPS, however, are constrained by a number of known limitations. A GPS receiver must have access to unobstructed lines of sight to a minimum of four satellites, and obscured satellites significantly jeopardize localization quality. Indoors, a GPS receiver either is slow to obtain a fix, or more likely does not work at all. This makes it impractical to depend on GPS for determining the room of a building in which a receiver is located, let alone the location within that room. Even outdoors, under optimal circumstances, accuracy is limited to a few meters (or perhaps a meter with modern SBAS systems). These limitations make GPS insufficient for fine-positioning applications such as guiding a car to a precise location in a parking lot, or guiding a robot within an indoor room or warehouse. To overcome the robustness and accuracy limitations of GPS, alternative localization technologies have been proposed, which are either less accurate than GPS

(e.g., triangulation of cellphone towers and WiFi hotspots), or too expensive and/or cumbersome to deploy (e.g., RFID localization or special-purpose sensors embedded in the environment). Inertial navigation and odometry, which are often used in robotics for fine-positioning tasks, require a known initial position, drift over time, and lose track (requiring manual re-initialization) when the device is powered off.

This chapter proposes a system that provides millimeter-scale localization, both indoors and outside on land. The key observation behind our approach is that seemingly-random ground textures exhibit distinctive features that, in combination, provide a means for unique identification. Even apparently homogeneous surfaces contain small imperfections – cracks, scratches, or even a particular arrangement of carpet fibers – that are persistently identifiable as local features. While a single feature is not likely to be unique over a large area, the spatial relationship among a group of such features in a small region is highly likely to be distinctive, at least up to the uncertainty achievable with coarse localization methods such as GPS or WiFi triangulation. Inspired by this observation, we construct a system called Micro-GPS that includes a downward-facing camera to capture fine-scale ground textures, and an image processing unit capable of locating that texture patch in a pre-constructed database within a few hundred milliseconds.

The use of image features for precise localization has a rich history, including works in graphics such as Photo Tourism [85] and Computational Re-Photography [8]. Thus, a main contribution of our work is observing that some of the algorithms used for feature detection and matching in "natural" images, as used by previous work, can also be used for "texture-like" images of the ground. In exploring to find a particular combination of known methods that can robustly work in this way, we also exploit two key advantages of ground-texture images. First, the ground can be photographed from much closer range than typical features in the environment, leading to order-

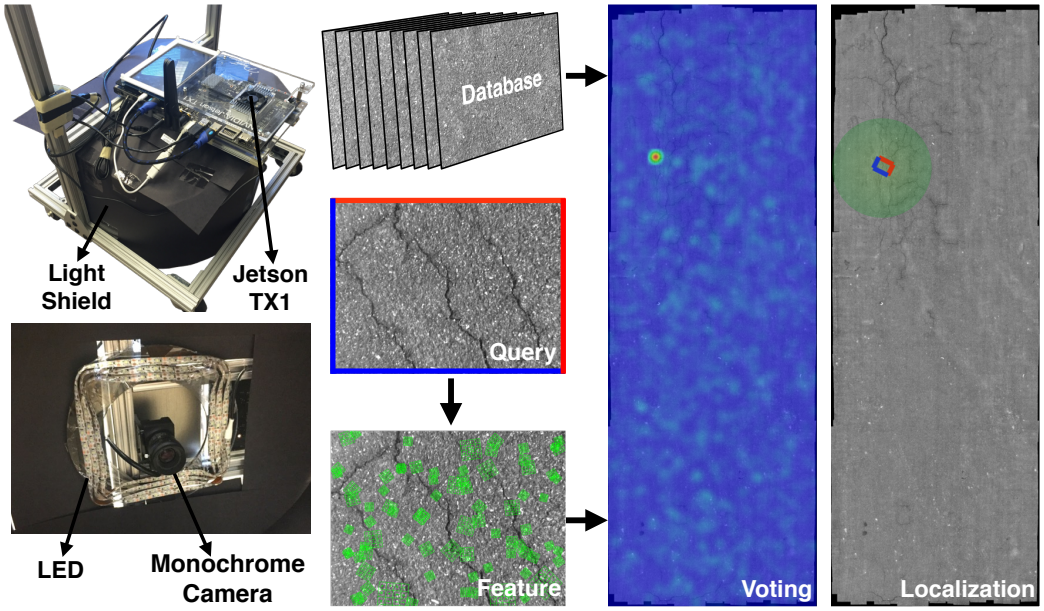


Figure 2.1: System overview. Our test robot features an NVIDIA Jetson TX1 development board, which controls a Point Grey mono camera. A light shield and ring of LEDs around the camera provide controlled lighting. We first capture database images as a preprocess, and stitch them into a globally consistent map. Later, to locate the robot, we extract features from a query image, and they vote for potential image poses in the map. A peak in the voting map determines inlier features, from which we recover the pose of the query image.

of-magnitude improvement in precision. Second, the statistics of texture-like images lead to a greater density of features, leading to greater robustness over time.

Our system consists of two phases: an offline database construction phase, and an online localization phase (Figure 2.1). We begin by collecting ground texture images and aligning them using global pose optimization. We extract local features and store them in a database, which is subsequently compressed to a manageable size. Next, during the localization phase, we find features in a query image and search the database for candidate matches using approximate nearest neighbor matching. We find that in typical textures more than 90% of the matches are spurious. We therefore use a voting method to reject outliers, based on the observation that inlier matches will vote for a consistent location whereas outliers distribute their votes randomly.

Finally, we use the remaining inlier matches to precisely calculate the location of the query image.

The major contributions of this work are:

- Describing a low-cost global localization system based on ground textures and making relevant code and instructions available for reproduction.
- Capturing and making available datasets of seven indoor and outdoor ground textures.
- Describing an architecture for building a global map of commonplace texture, then expanding it at a later date to incorporate new data.
- Investigating the design decisions necessary for practical matching in texture-like images, as opposed to natural images. This includes the choice of descriptor, strategies for reducing storage costs, and a robust voting procedure that can find inliers with high reliability.
- Demonstrating real-world applications of precise localization, including capturing the locations of footsteps as well as an autonomous vehicle that can use Micro-GPS to record a path and then follow it with sub-centimeter accuracy.

Knowing the precise location of a footstep both indoors and outside offers a range of HCI and IoT applications, in addition to the more obvious use in performance capture. Moreover, the ability to localize a vehicle or robot precisely has the potential for far-reaching applications. A car could accurately park itself (or guide the driver to do so) in any location it recognizes from before, avoiding obstacles mere centimeters away. A continuously-updated map of potholes could be used to guide drivers to turn slightly to the left or right to avoid them. The technology applies equally well to vehicles smaller than cars, such as Segway, electric wheelchairs, and mobility scooters for the elderly or disabled, any of which could be guided to precise locations or around

hard-to-see obstacles. Indoor applications include guidance of warehouse robots and precise control over assistive robotics in the home.

This chapter is derived from a conference preprint [111], which was accepted for presentation prior to the completion of this thesis.

2.2 Related Work

Feature Detectors and Descriptors: There is considerable research on finding interest points, estimating canonical scales and orientations, and computing discriminative fixed-dimensional descriptors; these are all well-investigated problems and have inspired many feature-based computer vision applications.

A seminal strategy for *detection* of interest points is the use of image corners, found for example by the corner detector of Harris and Stephens [25]. Instead of detecting corners, one can alternatively detect "blobs" in the images. The SIFT detector [49], which detects extrema over space and scale, has shown great success in terms of robustness and repeatability. The original SIFT approximates the Laplacian of Gaussian (LoG) filter with a Difference of Gaussians (DoG), while SURF [12] accelerates the process further by approximating LoG with a box filter. SIFT assigns an orientation to each interest point based on the dominant gradient orientation in its local image window. The scale and rotation invariance properties of SIFT make it popular in the domain of structure from motion [87, 103], where a particular feature is typically observed with different scales and orientations in different images.

Discriminative feature *descriptors* are crucial for uniquely identifying an image patch. SIFT [49] computes histograms of gradient orientations and is robust against modest pose and illumination changes. Several subsequent descriptors offer faster computation, including SURF [12], DAISY [91] and binary descriptors [71, 11]. A more recent approach named HardNet [59] learns a discriminative descriptor using a

convolutional neural network with online *hard negative mining*, and outperform SIFT on some natural image benchmarks.

Although feature descriptors are designed to be discriminative, the problem becomes more challenging for feature matching in *large databases*; a descriptor computed from a small image patch surrounding an interest point might not be discriminative enough. Von Hundelshausen and Sukthankar [97] therefore construct a network using SIFT keypoints, and the descriptors are computed as sampled intensity values along network edges. Compared with local patch-based descriptors, such descriptors are expected to be more globally unique. In contrast, our approach addresses this problem by gathering votes from multiple features in an area, which collectively discriminate the location. This approach is similar to the work on Implicit Shape Models [41], which votes for the object center by leveraging a pre-trained database of visual words along with displacement vectors.

Our system relies on SIFT for feature detection and description. While they have been employed extensively in research and practice, prior systems have generally applied SIFT to *natural* images. Since natural images have different statistics from the texture-like images studied in this work, it is not obvious that that SIFT should perform well for our application. However, Liu [48] showed that SIFT descriptors are sufficiently unique and discriminative to be used for matching in textures like the ones we investigate. We build upon this work by constructing a scalable system that incorporates dimensionality reduction and, most crucially, multi-feature voting, to robustly localize within such textures. Section 2.6.1 offers a comparison of different combinations of feature detector and descriptor, and shows that SIFT outperforms the others for our datasets.

Textures for Tracking and Localization: Textures such carpet, wood grain, concrete or asphalt all have bumps, grooves, and variations in color from location

to location, and we typically use the overall pattern or *statistics* of this variation to recognize a particular material. Indeed computer-based modeling and recognition of textures traditionally proceeded along statistical lines [19, 43]. Moreover, researchers have successfully synthesized new texture by example using parametric [29] and non-parametric [20] models. However, when we study the *particular* arrangement of bumps and variations present at any location in real-world textures, we find that it is unlikely to be repeated elsewhere.

Kelly et al. [38] introduce a warehouse automation system in which a downward facing camera installed on each robot is used to help track the robot. They observe that ground surfaces usually exhibit cracks and scratches, and it is possible to track the motion of the camera over a pre-constructed visual map. This work, however, still assumes a known initial location and surface textures are leveraged only for pairwise (*local*) frame matching, much as is done in an optical mouse. Fang et al. [21] adopts a similar idea and uses ICP to align the test frame with the best map frame, but this system still relies on odometry to determine a target map frame and refine positioning. In contrast, our approach performs *global* localization, which could be used to initialize tracking systems such as these. Clarkson et al. [17] demonstrate that seemingly-random textures can provide a means for unique identification. The authors observe that the fine-scale variations in the fibers of a piece of paper can be used to compute a "fingerprint" that uniquely identifies the piece of paper. Our work demonstrates that ground textures, including man-made ones such as carpet, share similar properties at sufficiently fine scales, and thus may be used for localization.

Relocalization: Structure from motion allows reconstruction of a large scale 3D point cloud offline, but relocating a newly captured image in the reconstructed point cloud without any initial guess about the camera position is challenging. Previous works explore direct 2D-to-3D matching [76] to estimate the 6 DoF pose of a photo

with respect to a reconstructed point cloud. Li et al. [44] proposes a method to leverage a co-occurrence prior for RANSAC and achieves relocalization on a larger georegistered 3D point cloud within a few seconds. Kendall et al. [39] trains a convolutional neural network (PoseNet) to regress the input RGB image to the 6-DoF camera pose. Relocalization is an essential module of modern SLAM systems, such as ORB-SLAM [63], which uses a bag-of-words model for matching. Researchers have also explored using skylines from omni-images to perform relocalization [68].

All the above approaches except PoseNet involve large-scale feature matching, which triggers a large amount of false matches and becomes the bottleneck of improving efficiency. To speed up feature matching, more compact models can be constructed by selecting a subset of stable 3D points from the original models [45, 16]. An effective approach to handle a high outlier ratio is voting [109]. This has also proven successful in the field of image retrieval, where spatial verification is commonly applied to rerank the retrieved list of images, and variants of Hough voting have been proposed to improve efficiency and robustness [6, 105, 80]. With more sensors available, one can utilize the gravity direction [89] as an additional constraint. Baatz et al. [7] leverages both gravity direction and a 3D scene model to rectify images, transforming the 6-DOF pose estimation problem into a 3-DOF problem.

Mobile devices are ideal deployment platforms for a relocalization system. Lim et al. [46] achieve localization on a micro aerial vehicle at interactive framerates by distributing feature extraction over multiple frames. Middelberg et al. [54] achieve real-time performance by combining online camera tracking and an external localization server. Irschara et al. [32] and Wendel et al. [99] demonstrate that GPUs, which are now widely available on mobile processors, can be used to accelerate localization.

Memory consumption quickly becomes an issue in relocalization when the offline database grows. Sattler et al. [75] address this problem by quantizing descriptors into a visual vocabulary so that only visual word IDs need to be stored. Approaches

based on visual vocabulary, however, have a nontrivial "sunk cost" to store the visual words [58]. An alternative solution to descriptor compression is to project descriptors into a low-dimensional space. Philbin et al. [67], for example, project the descriptor into a 32-D space while maintaining good performance on image retrieval. PCA-SIFT [36], despite its name, applies Principal Components Analysis (PCA) to a normalized patch to obtain a 36-dimensional descriptor. Nevertheless, Lynen et al. [52] show that aggressively compressed descriptors are weaker, although still sufficient for camera tracking.

Almost all of the above relocalization techniques rely on landmarks, such as buildings, that are normally positioned at least a few meters from the camera. This distance, combined with finite image resolution and inherent uncertainty in camera intrinsics, means that even a small error in feature localization results in a large uncertainty in estimated camera pose. This inaccuracy can be ameliorated by increasing the field of view of the camera [78, 5], because as more features are detected, more constraints can be introduced to improve pose estimation. Further uncertainty comes from the ambiguity in landmark identification, since it is not unusual to find buildings or parts of buildings (such as windows) that appear the same from a significant distance. Moreover, natural images used by the above systems suffer from perspective foreshortening, which brings difficulties to feature matching. Many features are not "time-invariant" in highly dynamic scenes. Thus, it is necessary to update the database frequently. Finally, these systems can be affected by changes in lighting. In contrast with these systems, our work positions the camera close to the texture being imaged and uses controlled lighting, leading to higher precision and robustness.

2.3 Mapping

This section describes the offline phase of our pipeline, including data collection, image stitching and database construction. Section 2.4 will describe the online phase – the process of locating a new frame in the pre-built map without knowing the prior frames. We begin by describing our hardware setup, then describe the process of building a database of features and locations.

2.3.1 Hardware Setup and Data Collection

Our imaging system consists of a PointGrey CM3 grayscale camera pointed downwards at the ground (Figure 2.1a). A shield blocks ambient light around the portion of the ground imaged by the camera, and a set of LED lights arranged symmetrically around the lens provides rotation-invariant illumination. The distance from the camera to the ground is set to 260 mm for most types of textures we have experimented with. Our system is insensitive to this distance, as long as a sufficient number of features can be detected.

The camera output is processed by an NVIDIA Jetson TX1 development board, which was chosen for both its GPU computation capabilities (256 CUDA cores) as well as its general-purpose specifications (quad-core ARM Cortex-A57 processor, 4 GB RAM). This processor has been used in tablets such as Google Pixel C, and it achieves computation speed within an order of magnitude of typical desktop machines despite being mobile and battery-powered.

Our prototype has the imager and development board mounted on a mobile cart, which may be moved manually or can be driven with a pair of computer-controlled motorized wheels. The latter capability is used for the "automatic path following" demonstration described in Section 2.7. For initial data capture, however, we manually move the cart to ensure that an area can be fully covered. This process could be

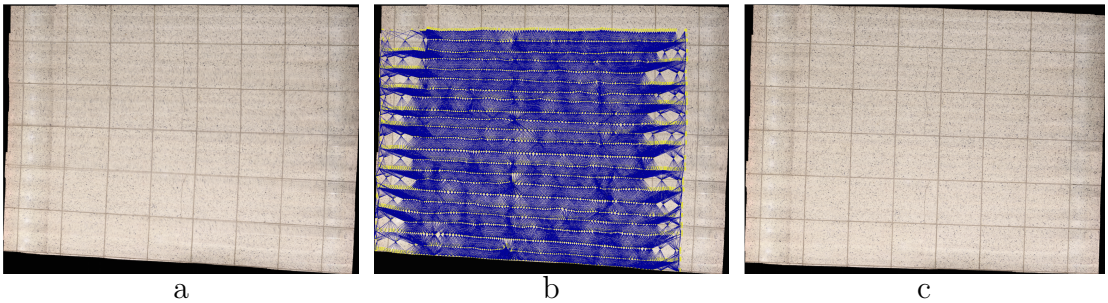
automated by putting more engineering effort or even through crowd-sourcing when there are more users. When capturing the data, we follow a zig-zag path so that many loop closures can be detected for global optimization. Unlike many 3D reconstruction problems, in which the user has to carefully move the camera and receive live feedback to guarantee full coverage, our capture process is simple and easily mastered by non-experts.

One limitation of our prototype system is the potential for blur caused by vibration of the cart over rough terrain. While we use relatively short exposures (3-5 ms) to address this challenge (as well as motion blur caused by high-speed motion) we expect that future systems may ameliorate the vibration problem with hardware suspension and stabilization systems. Although we use a short shutter speed, the frame rate is set to 5 fps to save disk space.

We have also tested our system on images captured on an Apple iPhone 6 to verify that the system also works for inexpensive smartphone cameras. We use the slow-motion mode with 240 fps, which is equivalent to a high shutter speed. In our experiments, we found vibration to be a more critical problem for the iPhone camera, so we put foam earplugs under the iPhone to reduce vibration.

2.3.2 Image Stitching

After data collection the next step is to construct a global map. We assume that the surface is planar, even though this is not strictly true for most outdoor surfaces. Fortunately, most ground surfaces are *locally* planar, making our planar surface assumption still valid for image stitching. Our image stitching pipeline consists of frame-to-frame matching followed by global optimization, following the general approach of Brown and Lowe [14]. Since the computation becomes significantly more expensive as the area grows, we use a fast method to scale image stitching to larger areas without compromising the quality too much.



a b c

Figure 2.2: Image stitching pipeline. (a) Initial stitched map, in which severe drift can be observed. (b) Loop-closure pairs found by matching neighboring frames between columns of images. (c) Stitched map after global optimization.

Frame-to-Frame Matching: Since the image sequence is captured continuously by following a zig-zag path, we are able to obtain an initial alignment of images captured. For each pair of adjacent frames, we follow a standard approach of extracting SIFT features from both frames, computing their SIFT descriptors, matching the descriptors using approximate nearest neighbors, and using RANSAC to obtain a set of consistent matches and estimate a 2D rigid transformation. In this way, we obtain an initial global pose for each frame, but these can be inaccurate due to accumulation of frame-to-frame error (Figure 2.2a). We therefore need to use global optimization to correct the drift.

Global Optimization: During capture, we ensure that every two consecutive columns of images are overlapping, which provides many potential loop closure pairs for global optimization. In particular, for each frame we find the ten closest frames that were *not* adjacent in the image sequence, given the initial global poses. We align each candidate "loop closure" pair using the method above. If the number of inlier feature pairs was below a threshold we discard the match; otherwise, we keep the matching features to use during global optimization. Figure 2.2b shows an example of all the loop closure pairs detected.

We perform a single global optimization, taking into account all the detected loop closure pairs as well as the frame-to-frame pairs. The optimization minimizes the

total re-projection error of all the matched correspondences from all the matched pairs:

$$E = \min_{\{[R|t]\}} \sum_{(i,j)} \sum_{(x_i^k, x_j^k)} \left\| x_i^k - [R|t]_i^{-1} [R|t]_j \cdot x_j^k \right\|^2 \tag{2.1}$$

where (i, j) are the indices of a pair of well-matched frames, (x_i^k, x_j^k) is the k-th pair of matched correspondences in (i, j), and $\{[R|t]\}$ is the set of global poses of all frames. We minimize the above energy function using the Ceres solver [2].

The final stitched image after global optimization is shown in Figure 2.2c. Most of the errors in the initial stitched image are fixed due to the added loop closure pairs and we almost never see blurred regions, implying that the stitching quality is acceptable.

2.3.3 Scaling to Larger Areas

We have found that the above pipeline is sufficient for mapping areas up to several square meters. Beyond this, however, it becomes challenging both to capture the data in one continuous session and to perform the global optimization in a reasonable time. Therefore, we propose a method to capture small portions of the map at a time, compute correspondences between these regions, and link them together into a single global coordinate system.

Our method first asks the user to manually specify all regions that neighbor each other. For each pair of neighboring regions, we obtain a rough initial rigid transformation by asking the user to click on two correspondences between the maps. Although these correspondences could also be found by using the Micro-GPS localization system itself, as demonstrated in Section 2.9, in our experience asking the user takes only a few seconds. These initial correspondences can be rough, since they will be automatically refined later.

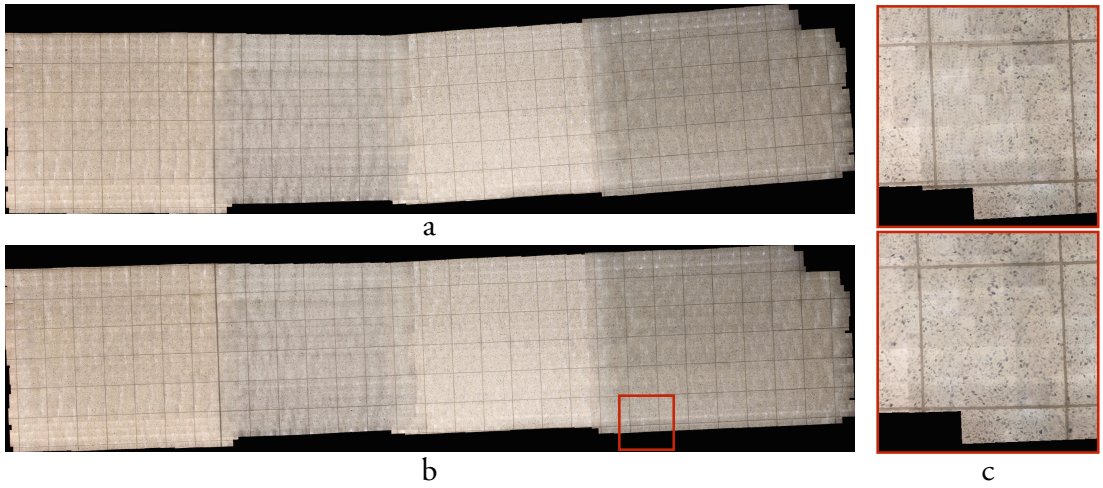


Figure 2.3: Region linking. The full map is constructed from four pre-stitched regions. (a) Initial linking result. (b) After global optimization. (c) Close-ups of the region marked in red, before and after local refinement.

Initial Linking: We treat each pre-stitched region as a large image and our first step is to compute the initial global poses of the regions. We fix one region as the reference and compute the shortest path in the region-adjacency graph from the reference to all the other regions. This is to minimize the number of pairwise transformations being concatenated so that drift can be reduced. Figure 2.3a shows an example of the full map after initial linking.

Global Optimization: We adopt the same global optimization method we use in image stitching. For each pair of neighboring regions, we match every pair of images across the two regions such that the distance between the image origins is within a threshold. We then compute a global pose for each region, taking the computed feature matches into account. Note that at this stage, the poses of images within each already-stitched region are constant, and we solve for only a single rigid-body transform per region.

As shown in Figure 2.3b, although the initial linking result is far from perfect, our global optimization is able to correctly align the pre-stitched regions relative to each

other. However, we notice some remaining misalignments in the overlaps between regions, which we fix with a local refinement step.

Local Refinement: We re-optimize for the optimal pose for each frame that overlaps two regions. These poses are optimized based on the region-to-region correspondences computed in the above global optimization step, as well as intra-region correspondences. Specifically, let $\mathcal{R}_i$ and $\mathcal{R}_j$ be overlapping regions, and let f be a frame in $\mathcal{R}_i$ that overlaps some frames in $\mathcal{R}_j$. The pose $[R|t]_f$ of f is optimized based on correspondences between f and those frames in $\mathcal{R}_j$ that it overlaps, as well as overlapping frames in $\mathcal{R}_i$. Note that the poses of frames that do not overlap the other region are held fixed. Figure 2.3c shows a comparison of zoom-in views after global optimization and local refinement.

Output: After global alignment, we subsample the set of images to reduce both storage and computational costs. We remove as many images as possible, while still ensuring that the image-space overlap between the images we keep is at least 30%. The final output of the image stitching pipeline is composed of the selected images and their optimized global poses. For the dataset shown in Figure 2.3, the size of the area is around 12.75m^2, and 1296 images are selected to achieve approximately full coverage.

2.3.4 Database Construction

The final stage in building a map is extracting a set of features from the images and constructing a data structure for efficiently locating them. There are many design decisions – which feature descriptors to use, how many features to keep, and how much dimensionality reduction to perform – all having a strong impact on computational efficiency, the size of the database, and the success rate of matching. Here we

describe our actual implementation, while some of the key decisions will be evaluated in Section 2.6.

Feature Database: We use the SIFT scale-space DoG detector and gradient orientation histogram descriptor [49], since we have found it to have high robustness and (with its GPU implementation) reasonable computational time. For each image in the map, we typically find 1000 to 2000 SIFT keypoints, and randomly select 50 of them to be stored in the database. This limits the size of the database itself, as well as the data structures used for accelerating nearest-neighbor queries.

To further speed up computation and reduce the size of the database, we apply PCA to the set of SIFT descriptors and project each descriptor onto the top k principal components. As described in Section 2.6, for good accuracy we typically use $k = 8$ or $k = 16$ in our implementation, and there is minimal cost to using a "universal" PCA basis constructed from a variety of textures, rather than a per-texture basis.

Each SIFT keypoint is associated with the image it came from (for which we know the global pose), as well as the keypoint coordinates (x, y), orientation θ, and scale s. The feature's pose in image coordinates can be found from (x, y, θ), and then transformed into *global* coordinates by composing with the image pose. A feature located in the overlap between two images can potentially have two copies, one extracted from each image. We detect this situation and keep only the feature that was closer to the center of its image.

Building the Search Index: One of the major advantages of our system is that the height of the camera is fixed, so that the scale of a particular feature is also fixed. This means that when searching for a feature with scale s in the database, we only need to check features with scale s as well. In practice, to allow some inconsistency, we quantize scale into 10 buckets and divide the database into 10 groups based on scale. Then we build a search index for each group using FLANN [60]. During testing,

given a feature with scale s, we only need to search for the nearest neighbor in the group to which s belongs.

2.4 Localization

The input to our online localization phase is a single image. We assume that the height of the camera above the ground is the same as during mapping (or that the ratio of heights is known), so that the image scale is consistent with the images in the database.

Feature Computation and Matching: We first extract SIFT features from the test image as in database construction. Then we use the same principal components computed during database construction to reduce the descriptors to dimension k. For each descriptor, we search for the nearest neighbor using the search index we have built (for the appropriate scale, as mentioned previously).

Precise Voting: Recall that we only keep 50 features per database image. Therefore, we expect only a small subset of the features found in the test image to have a correct match in the database. Finding this small set of inliers is challenging, given that standard methods such as RANSAC do not work well if outliers highly outnumber inliers. Instead, we adopt a voting approach.

Our method is based on the observation that, due to the randomness of ground textures, false matches are usually evenly distributed in the map. Fortunately, since true matches usually come from one or two images, they are concentrated in a small cluster. Figure 2.4, top left, shows a heat map of feature matches in a database, with red indicating high density, green intermediate, and blue indicating low density. While this might appear promising, and indeed we are able to build a system with

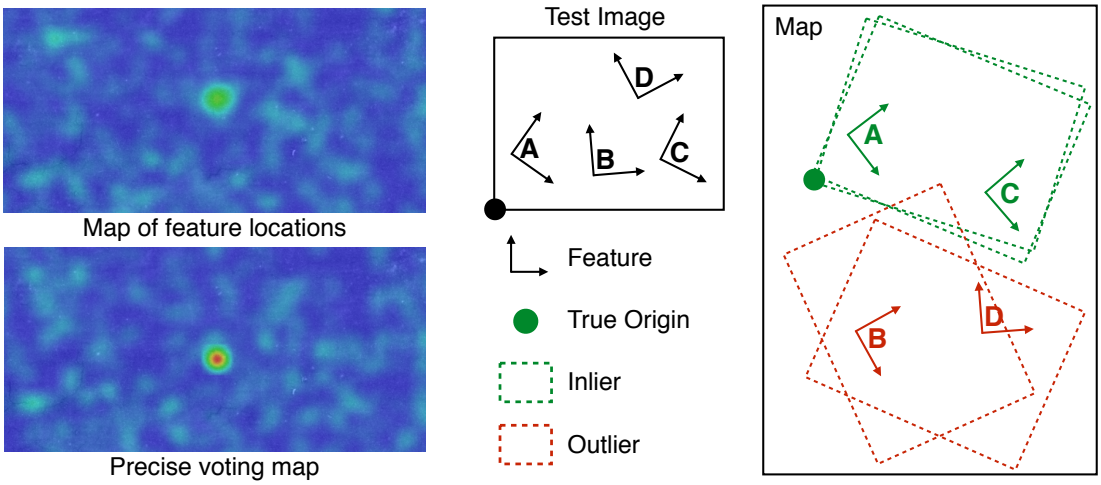


Figure 2.4: Top left: A simple strategy would be to vote for the locations of matched features. Right: We instead employ a "precise" voting strategy in which each feature votes for the location where the origin of the test image would be located in the map, if that feature were a correct match. True matches lead to a concentration of votes. Bottom left: The precise voting leads to a more defined local maximum.

modest success rates based on this principle, the relative concentration of correct matches is still insufficiently high relative to the density of outliers.

The problem with voting based on feature location is that the correct features are distributed throughout the entire area corresponding to the test image. This leads to only a moderately-high density of votes in the map near the location of the test image. The solution is to concentrate the votes: we want all of the true features to vote for the *same* point in the map, leading to a much greater difference between the peak corresponding to the true location and the background density of outliers.

In particular, each feature casts a vote for *the origin of the test image* by assuming that nearest neighbors are true matches (see Figure 2.4, right). Denote a feature extracted from the test image as f_t and its nearest neighbor in the database as f_d. If the feature pair $\{f_t, f_d\}$ is a true match, we can compute the pose of the test image T in world coordinates, denoted $[R|t]_T^W$, by composing the pose of f_d in world

coordinates and the pose of f_t in the test image:

$$[R|t]_T^W = [R|t]_{f_d}^W \, [R|t]_{f_t}^{f_d} \, [R|t]_T^{f_t}, \tag{2.2}$$

where $[R|t]_{f_t}^{f_d}$ is assumed to be the identity. We then vote for the location of the origin of the test image, which is the translational component of $[R|t]_T^W$.

Using this strategy, implemented via voting on a relatively fine spatial grid, we find a much tighter peak of votes, relative to the uniform background of outliers, as shown in Figure 2.4, bottom left.

Final Refinement: After voting, the cell with the highest score is very likely to contain the true origin of the test image. We select all of the features in that peak as likely inliers, and perform RANSAC just on them to obtain a final estimate of the pose of the image.

2.5 Datasets

We have experimented with a variety of both indoor and outdoor datasets, covering ground types ranging from ordered (carpet) to highly stochastic (granite), and including both the presence (concrete) and absence (asphalt) of visible large-scale variation (see Appendix A for details). While we believe that our system is applicable to a wide range of floor and ground materials, localization is of course only possible in the presence of unique and stable textures. We believe that our system would not work in areas containing, for example, snow, high-pile carpet, or vinyl flooring with highly-repeating printed texture. When a reliable ground surface is absent, leveraging subsurface features, such as what has been demonstrated in LGPR [18], could be a feasible solution.

2.5.1 Indoor Datasets

Tiles: This is a dataset of man-made indoor tiles containing colored chips in a white binder. While the tiles as a whole have a regular shape, the patterns on the tiles are random. We captured this dataset (shown in Figure A.1) in color, using the camera of the iPhone 6. The color of the overall map is not consistent, because the map is stitched from four individual regions captured on different days and because we intentionally captured the lower part of the map with no extra illumination, to test the robustness of our system. Note that the grid lines in the stitched map are straight, demonstrating good alignment. The accompanying video shows the robustness of localization in this dataset, despite the variation in color and contrast.

Carpet: This dataset contains a commercial low-pile carpet, typical of that found in many office buildings. We captured the dataset (shown in Figure A.2) using the PointGrey camera with ambient light blocked and illumination from the ring of LEDs. A challenging aspect of this material is that it has a pattern that is close to repeating, potentially making feature matching difficult. Nevertheless, with high resolution there is sufficient uniqueness everywhere to enable feature-based localization. Moreover, as shown in Section 2.6.4, it remains robust to our attempts to perturb the carpet.

Wood: This is a dataset of a stained hardwood floor, as commonly used in both residential and commercial applications (Figure A.3). Although the wood contains significant texture, this texture consists mostly of elongated elements, unlike the "blob-like" elements found in our other datasets. Since this is not a situation for which the SIFT detector was designed, we detect an order of magnitude fewer features in these images. As discussed in Section 2.9, this decreases the performance of our system. In an attempt to regain some of this lost performance, we index 100, instead of 50, features for each database image.

2.5.2 Outdoor Datasets

Granite: This is an outdoor dataset of large slabs of granite laid to form a sidewalk (Figure A.4). The surface of the granite is rough, leading to variability in shading from hour to hour and day to day, unless ambient light is blocked.

Asphalt with Fine Aggregate: Road surfaces and sidewalks are often coated with asphalt containing aggregate – stones with a particular size distribution. This dataset is of asphalt with fine aggregate on a roadway, as shown in Figure A.5. This dataset was captured near a loading zone where trucks are parked every day, which explains the presence of long cracks. In our stitched map, these cracks are faithfully recovered.

In order to test the robustness of our system under different lighting, we captured two versions of this dataset: under directional sunlight, with the cart's light shield removed (shown in Figure A.5) and at night, using LED illumination (Figure A.6). The results of this experiment are described in Section 2.6.4.

Asphalt with Coarse Aggregate: This dataset, shown in Figure A.7, differs in many ways from the previous one. First, it is relatively new, meaning that there is little large-scale variation from place to place. Therefore, localization can only leverage fine-scale texture. Second, the coarse surface caused significant vibration in our cart, leading to occasional motion blur during capture. Nevertheless, due to our use of a short shutter speed, our reconstructed map is sharp almost everywhere.

Concrete: Figure A.8 shows our concrete dataset. Stains on the ground can be clearly seen in the reconstructed map. The straight lines dividing the concrete also gives a sense of the quality of the reconstructed map.

Table 2.1: Performance of Micro-GPS. From left to right: texture type, elapsed time between capture of database and test sequence, number of test frames, and success rates using 8- and 16-dimensional descriptors.

Texture	**Elapsed**	**# frames**	**Rate-8**	**Rate-16**
fine asphalt	16 days	651	76.04%	95.24%
carpet	30 days	1179	99.49%	99.92%
coarse asphalt	17 days	771	97.54%	99.09%
concrete	26 days	797	83.31%	93.35%
granite tiles	6 days	862	79.47%	94.43%
tiles	18 days	1621	93.83%	98.40%
wood	0 days	311	59.48%	77.49%

2.6 Evaluation

In order to evaluate the accuracy and robustness of a localization system, a typical approach would be to obtain ground-truth location and pose using a precise external measurement setup. However, this is impractical in our case due to the large areas mapped and the precision with which we are attempting to localize. Moreover, we are more interested in *repeatability*, rather than absolute accuracy, given that most of the applications we envision will involve going to (or avoiding) previously-mapped locations.

We therefore adopt an evaluation methodology based on comparing the query image against an image captured during mapping. Using the pose predicted by Micro-GPS, we find the closest image in the database, and compute feature correspondences (using all SIFT keypoints in the image, not just the 50 used for the database). If there are insufficiently many correspondences, we mark the localization result as a failure. We then compute a best-fit relative pose using those features. If the pose differs by more than 30 pixels (4.8 mm) in translation or 1.5° in rotation from the pose output by Micro-GPS, we again mark the result as a failure. Finally, given a sequence of consecutive poses that should be temporally coherent, we detect whether the poses of any frames differ significantly from their neighbors.

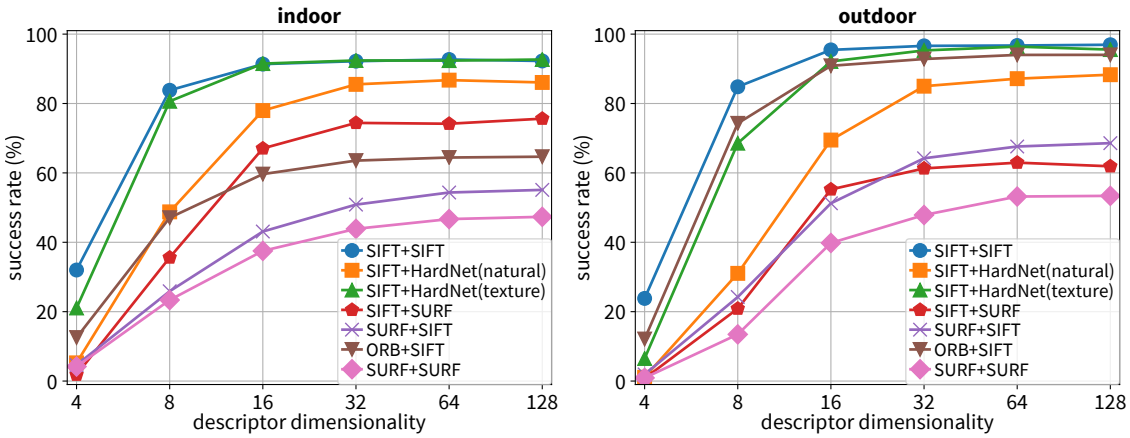


Figure 2.5: The average performance of different *detector* + *descriptor* combinations on both indoor and outdoor datasets. The horizontal axis indicates the dimensionality of descriptors after PCA.

The performance of our system, implementing the pipeline described in Section 2.4, is shown in Table 2.1. The two columns at right show performance with 8-dimensional and 16-dimensional descriptors, respectively. The correct pose is recovered over 90% of the time for most datasets (with independent per-frame localization and no use of temporal coherence), with the exception of the wood floor. This is because relatively few SIFT features are available in this dataset – see discussion in Section 2.9.

The following sections evaluate the impact of various design decisions, compare to image retrieval, and investigate the effect of deliberate attempts to introduce difficult matching conditions on the accuracy of the system.

2.6.1 Impact of Design Decisions

Selection of Feature: We first evaluate the impact on accuracy of using different combinations of feature detector and descriptor. While SIFT [49] has been popular since its introduction more than a decade ago, more recent alternatives such as SURF [12], ORB [71] have been shown to achieve similar performance at lower computational cost. Even more recently, convolutional neural networks have been used in learned descriptors [90, 59, 28, 114, 51, 37] to achieve much better matching perfor-

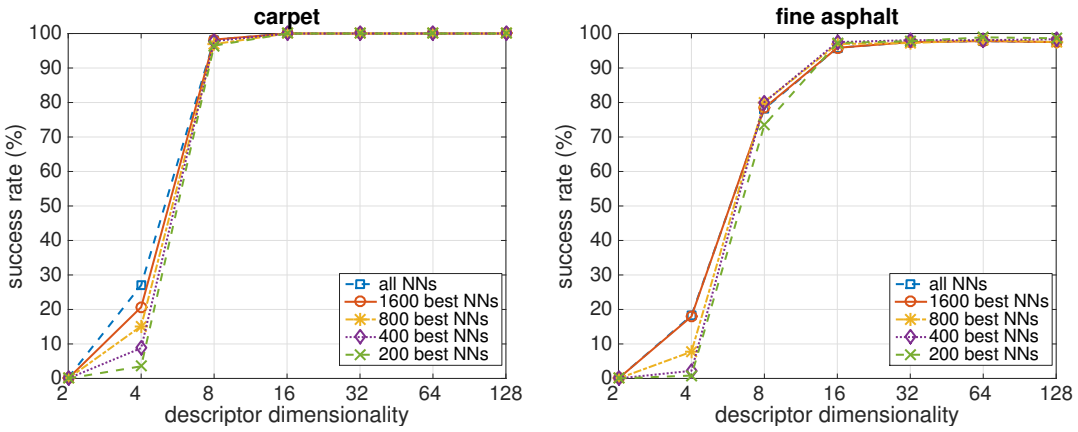


Figure 2.6: Success rate achieved by varying the number of features that participate in voting, as well as the dimensionality of feature descriptors.

mance than SIFT. In all of these cases, however, the performance has been optimized for *natural*, rather than texture-like, images. To evaluate the effectiveness of a learned descriptor on texture images, we select the recent well-performing HardNet [59] as our backbone network and learn a texture descriptor (HardNet-texture) using patches cropped from our dataset. During training, we also perform non-uniform intensity scaling to account for possible changes in exposure. Figure 2.5 compares the accuracy of different combinations of feature detector and descriptor. The SIFT *detector* outperforms both SURF and ORB, while both SIFT and HardNet-texture perform better than alternative *descriptors*. Because the SIFT descriptor can withstand more aggressive dimension reduction, it is the best choice for our current deployment. However, we observe that HardNet-texture shows significant improvement compared to the original HardNet optimized for natural images [101]. This suggests that domain-specific training may hold the promise for future improvements in the quality of learned descriptors.

Number of Features for Voting: We next evaluate the impact of restricting the number of features used for voting. Specifically, we limit voting to only the n feature matches having the smallest L_2 distance between the query and database

descriptors. As before, we run our experiments while retaining different numbers of PCA dimensions.

Given the results in Figure 2.6, we observe that our system achieves nearly perfect performance on the carpet dataset by using only 8 dimensions. The asphalt dataset requires higher descriptor dimensionality (16) to perform robustly. The number of nearest neighbors we include in the testing phase does not seem to have a significant influence on the performance when the descriptor dimensionality is large enough. Including more nearest neighbors, however, improves the success rate when the descriptor dimensionality is not sufficient (e.g. using 8 dimensions for the fine asphalt dataset). In all cases, we found that restricting the number of features to use for voting made little difference to computation time, since the majority of time was spent on computing the descriptors and performing nearest-neighbor lookups.

Method of Dimension Reduction: The effectiveness of Micro-GPS largely relies on the global distinctiveness of the descriptor, which can be evaluated by observing whether the true match of a descriptor can be found in a large collection of descriptors. Specifically, given n pairs of descriptors, we first search for the two nearest neighbors of every descriptor. We count the number of times (n_{true}) the second-nearest neighbor is a true match. The distinctiveness of the descriptor can be computed as $n_{\text{true}}/2n$. Unlike natural images used by most previous relocalization systems, our data seldom suffers from perspective foreshortening, making the feature matching process less noisy.

Dimensionality reduction must be efficient since it needs to be performed on every online frame. Therefore we limit our consideration to linear dimensionality reduction methods, which can be performed with just a matrix multiplication. Principal Components Analysis (PCA) is an obvious first choice, but we investigate whether a better set of bases for dimensionality reduction can be obtained through learning.

Table 2.2: Descriptor distinctiveness after dimensionality reduction using a fully-connected perceptron layer (1-fc), using either random or PCA initialization. We find that PCA is superior to both of these approaches.

Dimension	**1-fc**	**1-fc (PCA)**	**PCA**	**original**
128	-	-	-	57.01%
32	34.06%	45.56%	58.72%	-
16	29.31%	43.91%	55.72%	-
8	22.24%	30.79%	31.00%	-

Specifically, we train one fully-connected layer to reduce dimensionality, following the Siamese configuration as described in the Deep Descriptor [82]. We first extract pairs of matched SIFT descriptors and separate them into training and test sets. The loss function of the network minimizes the distance between matched descriptors and maximizes the distance between non-matched descriptors:

$$\mathcal{L}(d_i, d_j) = \begin{cases} \left\| f(d_i) - f(d_j) \right\|_2, & (i,j) \text{ is a match} \\ \max\left(0, C - \left\| f(d_i) - f(d_j) \right\|_2\right), & (i,j) \text{ is a non-match} \end{cases} \tag{2.3}$$

where (d_i, d_j) is a pair of descriptors, $f(.)$ is the dimension reduction function, and $C = 4$ is the maximum loss of a non-match pair. The batch size is set to 256, in which 128 are positive pairs and the other 128 are negative pairs. After pre-training, we also adopt hard negative mining: we only backpropagate the largest 32 losses, which correspond to the hardest 32 pairs of descriptors.

We experimented with both random initialization and initializing the fully connected layer with the parameters computed via PCA. Table 2.2 shows that PCA outperforms the learning-based method, even when the fully connected layer is initialized with principal components. This is partially because the loss function only penalizes based on pairwise distance, which does not account for global distinctiveness.

Table 2.3: For each type of texture, we evaluate the success rate (in percentage) by running our system with PCA bases computed from each texture and the union of all textures. Best numbers are bolded.

	Basis							
Texture	asphalt	carpet	coarse	concrete	granite	tiles	wood	union
asphalt	95.24	95.39	95.24	94.32	95.08	**95.70**	94.32	94.47
carpet	**100**	99.92	**100**	**100**	**100**	99.92	**100**	**100**
coarse	99.09	**99.35**	99.09	**99.35**	99.09	**99.35**	98.83	98.83
concrete	91.84	**93.73**	92.72	93.35	93.22	93.22	91.72	92.85
granite	**94.43**	93.85	94.08	93.62	**94.43**	94.08	93.16	93.97
tiles	98.27	**98.40**	97.90	98.27	98.09	**98.40**	97.59	97.78
wood	76.85	**78.78**	78.14	77.81	77.81	**78.78**	77.49	77.49

Choice of PCA Basis: We next investigate whether the PCA basis used for dimensionality reduction should be specific to each dataset, or whether a "universal" basis computed from the union of different textures achieves similar performance. Table 2.3 shows localization performance for each combination of texture and PCA basis, including a basis computed from the union of all datasets. The difference caused by switching PCA basis is negligible, and we conclude that there is no drawback of using a single PCA basis computed from all of the datasets.

Quantization Using Visual Words: An alternative solution to speeding up nearest-neighbor search when the database becomes large is to quantize descriptors using visual words. Visual words can be viewed as Voronoi cells, and descriptors belonging to the same cell are assumed to be matched.

We generate 2^{16} visual words by applying k-means clustering to descriptors extracted from all the database images in a single dataset. We also compare to visual words computed from a completely different dataset, and from the union of all the datasets. In the results shown in Figure 2.7, we observe that using visual words computed from each dataset itself (fourth column at left, third column at right) leads to better performance than using visual words from the other dataset (third column at

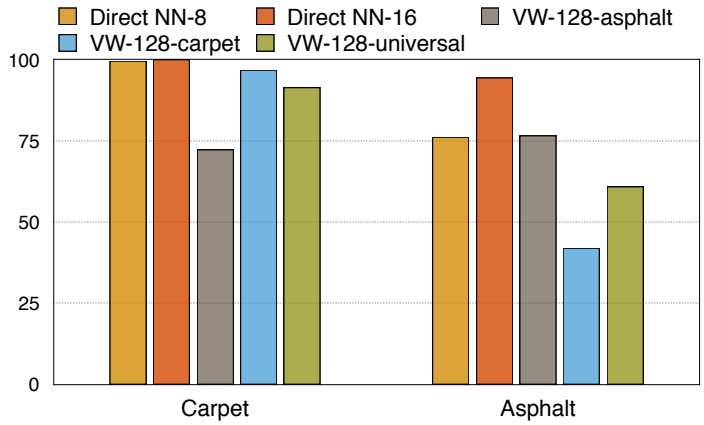


Figure 2.7: Comparing matching features using the original 8- and 16-dimensional descriptors, as well as using visual words (computed from the dataset itself, a different dataset, or the concatenation of all datasets).

left, fourth column at right), while a universal dictionary (fifth column) is somewhere in between. Nevertheless, directly using PCA dimension-reduced descriptors (first two columns) still shows better performance with less dependence on the dataset.

Resolution: The final design decision we evaluate is what impact image resolution has on localization performance. We expect that for most datasets there will be a "sweet spot" of resolution: using images that are too coarse will capture only large-scale features, which might not be discriminative enough. Conversely, using too fine a resolution might populate the database with tiny features that are easily disturbed over time.

To find out the best resolution to work on, we downsample the images with different ratios and evaluate the performance by extracting SIFT features from the downsampled images. The performance on different datasets is shown in Figure 2.8. We report both the downsampling rate and actual physical resolution in mm per pixel. In most cases, the performance is maximized when the resolution is 0.318mm per pixel (corresponding to 0.5× downsampling relative to our original capture resolution), and some datasets exhibit a dropoff in performance at higher resolution. Since downsampling the image by half can significantly speed up feature extraction, reduce

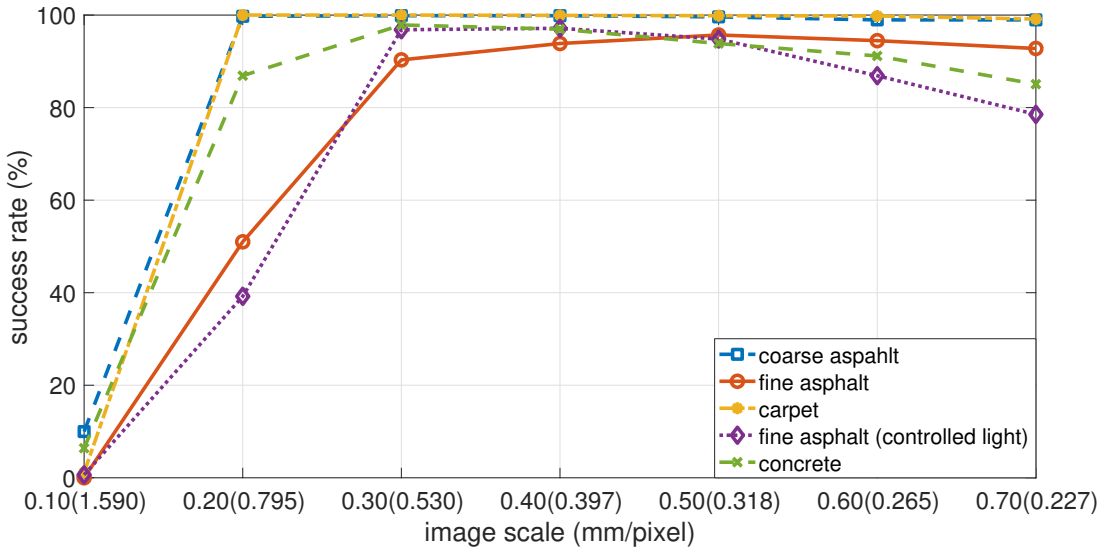


Figure 2.8: Success rate achieved by varying the scale of the image (horizontal axis) on which the SIFT detector is run.

noise, and maximize accuracy, we apply this downsampling as a standard component of our pipeline.

2.6.2 Comparison with Image Retrieval

One alternative to our feature-voting method is image retrieval: finding the nearest image to the query in the database, and performing image-to-image alignment afterwards. We compare Micro-GPS to a state-of-the-art image retrieval system [92] while taking storage, efficiency, and performance into account.

The retrieval system leverages visual words to avoid storing the original SIFT descriptors. In our experiments, we first generate 2^{16} visual words by clustering the descriptors using k-means. Descriptors assigned to the same visual word are then discriminated by a binary signature computed through Hamming Embedding. The length of the binary signature is the same as the length of the original descriptor (i.e., 128 if SIFT is used). This results in both low storage requirements and high efficiency in matching. Given the SIFT descriptors computed from the test image, binary signatures are computed and matched against the database. Each matched

Table 2.4: Performance of the image retrieval system, tuned to consume the same storage as Micro-GPS (cf. Table 2.1). From left to right: texture type, dimensionality of the descriptors used by Micro-GPS (used for setting the number of features for image retrieval, to ensure equivalent storage), and retrieval rates achieved using 8-, 16-, and 128-bit binary signatures.

Texture	**Micro-GPS Equivalent**	**Success rate**		
		8-bit	**16-bit**	**128-bit**
carpet	8-dim	74.39%	98.81%	99.07%
	16-dim	96.27%	99.92%	99.92%
fine asphalt	8-dim	26.57%	80.65%	90.63%
	16-dim	48.85%	94.16%	97.54%
wood	8-dim	33.44%	54.98%	59.16%
	16-dim	31.83%	54.02%	57.56%

signature votes for the image index with which it is associated. The image receiving the most votes is considered the "match."

Storing the binary signature as well as the location used for pose estimation for every descriptor, costs a non-trivial amount of storage. To ensure a fair comparison against our approach, we also apply PCA dimension reduction to the visual words and randomly sample features from each image so that the system consumes the same amount of storage as Micro-GPS. The dictionary of visual words also consumes some storage, but we ignore this because the cost is fixed and does not grow with the size of the map. The retrieval system is able to sample many more features when Micro-GPS uses high-dimensional descriptors. Micro-GPS nevertheless often only needs low-dimensional descriptors to achieve good performance, and the retrieval system does not show a strong advantage in storage in this case.

Table 2.4 shows the performance of the retrieval system. The retrieval system achieves the best performance when the full 128-bit binary descriptor is used, even though this requires the use of fewer features. In other words, given a fixed size of storage, the retrieval system prefers better features to more, but lower-quality, features. The performance of the retrieval system is comparable to that of Micro-

GPS on the carpet and asphalt datasets, but is considerably worse on the wood. This is due to the significantly smaller amount of features that can be detected on wood. In the retrieval system, votes from correct matches are often distributed to more than one database image, which can trigger failure when the number of features is small. In contrast, Micro-GPS utilizes all correct matches to estimate the pose.

The system runs in single-threaded Matlab on a server, and the average computational time on a typical dataset such as the carpet is 348ms, excluding feature extraction and final pose estimation. While an image retrieval system offers satisfactory performance on common datasets, our voting-based system appears to be both more accurate and faster for localization.

2.6.3 Downward- vs. Outward-Facing Cameras

Many existing systems focus on localization in natural images captured by an outward-facing camera. To compare our system with this approach to localization, we add an outward-facing camera (identical to the one used for Micro-GPS) to our setup and we trigger both cameras simultaneously. We use the state-of-the-art VisualSFM structure from motion system [103, 104] to recover the 3D trajectory of the outward-facing camera. In order to compare the resulting trajectory to ours, we need to project it onto 2D (which we do by estimating the plane that the trajectory lies in using least squares), and to recover the relative global scale, rotation, and translation (which we do by minimizing least-squares distance between points taken at the same timestamps).

Figure 2.9 compares both trajectories for two different environments, outdoors (with asphalt ground texture) and indoors (with the "tiles" texture). Ground truth trajectories that are accurate up to a few millimeters are not easy to obtain, especially outdoors. However, we observe that the trajectory of VisualSfM (in blue) is much noisier than that of Micro-GPS, with discrepancies of many centimeters. In contrast,

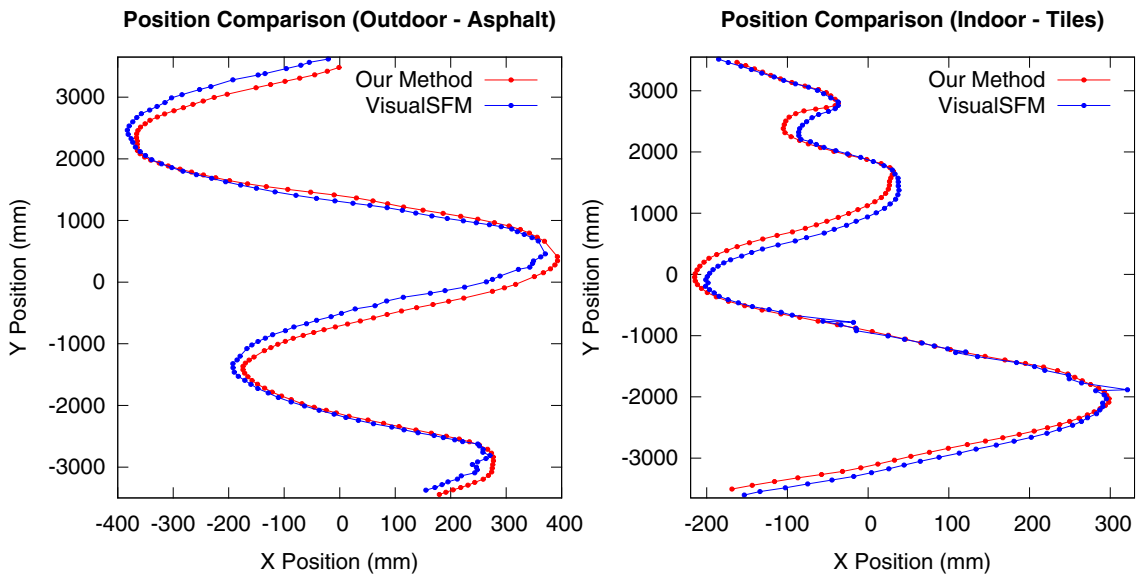


Figure 2.9: Comparison of camera trajectories obtained using our system with downward-facing cameras (red lines) and a state-of-the-art structure from motion system using outward-pointing cameras (blue lines). Left: trajectories on the outdoor *asphalt* dataset. The distance between the trajectories is 98.8 mm on average (maximum 211.5 mm) while the mean angle between camera poses is 0.5 degrees (maximum 1.3 degrees). Right: trajectories on the indoor *tiles* dataset. The mean distance is 62.9 mm (maximum 197.7 mm) and the mean anglular difference is 2.2 degrees (maximum 2.5 degrees).

the difference in estimated orientations is small (usually below 1 degree), suggesting that both methods were able to recover orientation successfully.

One factor that critically affects the performance of both systems is the number of SIFT features that can be detected. Our downward-facing camera detects an average of 1319 features per frame in the (outdoor) asphalt sequence and 2114 features in the (indoor) tile sequence, while the outward-facing camera detects only 637 and 256 features in the same settings. More detected features typically leads to more matched features, and hence greater localization accuracy.

Nevertheless, outward-facing cameras are commonly used for tracking and, at the same speed of motion, are less susceptible to motion blur than downward-facing cameras. We conclude that our system can be used in conjunction with a tracking

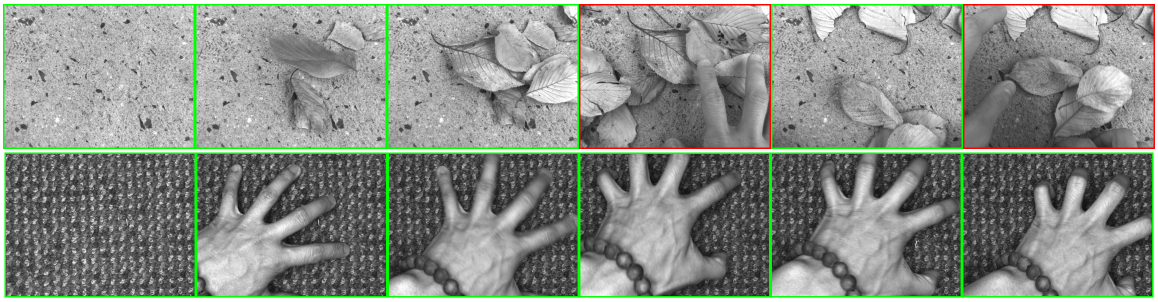


Figure 2.10: Introducing occlusion and perturbation. First row: introducing occlusion by adding leaves. Second row: introducing occlusion and perturbation by scratching. The green bounding box represents success while red represents failure

system based on an outward-facing camera, with comparable additional hardware and software costs.

2.6.4 Robustness

In the following section, we investigate the robustness of the proposed system by stress-testing under different real-world application scenarios.

Delay Between Mapping and Localization: Ground textures are unlikely to completely remain unchanged, especially for those outdoors. Since the database of our system might not be updated very frequently, it is important to test whether the system still works for test images captured a few days or even a few weeks after the database is constructed. Table 2.1 shows the performance of our system on different types of textures. It shows that our system is robust against changes in the scene.

Occlusion and Perturbation: Two particular ways in which ground texture can change over time are occlusion (e.g., dirt, leaves, etc.) and perturbation of soft materials (e.g., walking on carpet). Figure 2.10, top, shows frames from a sequence in which more and more leaves are piled on a patch of concrete. Frames outlined in green represent success, while frames outlined in red represent failure of localization. Note that our voting procedure is robust against a substantial amount of occlusion.

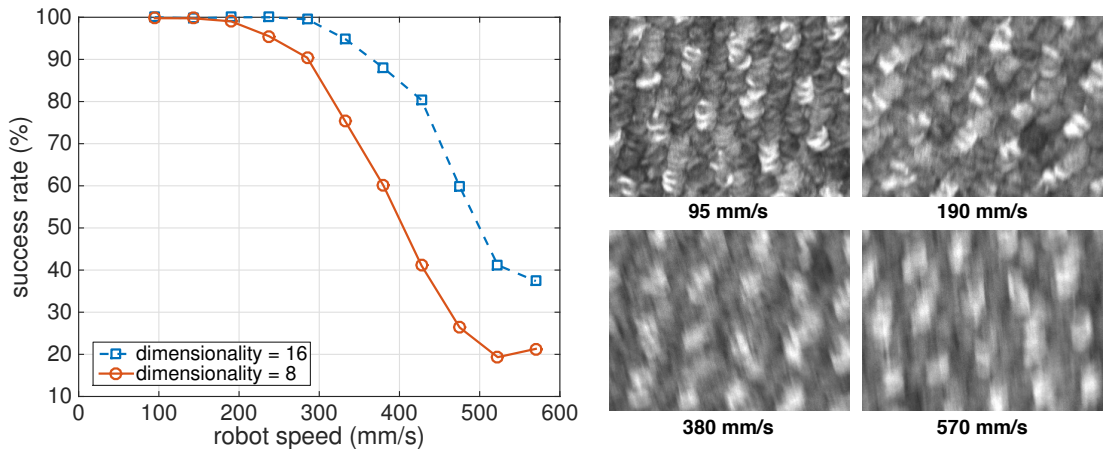


Figure 2.11: Left: Success rate achieved by varying the speed of the robot. The two curves correspond to feature dimensionality of 16 and 8, respectively. Note that this experiment uses a shutter speed of 10 ms, as opposed to the 3–5 ms we typically use. Right: Examples of motion-blurred images captured on the carpet.

At bottom, we show frames from a sequence in which we scratch the carpet by hand. All frames in this sequence resulted in successful localization. Note, however, that this was a heavy-duty low-pile commercial carpet; we expect that high-pile residential carpet may exhibit worse performance.

Motion Blur: Motion blur can easily happen when there is vibration or the camera is moving too fast. This perturbs both the feature detection and the computed feature descriptors, making localization less accurate. To evaluate how motion blur can affect the performance of our system, we use a robot which can run at a roughly-constant speed and evaluate performance by varying the speed. Unlike the previous experiments, we set the shutter speed to 10 ms (2 to 3 times longer than usual) to deliberately introduce more motion blur. Examples of images captured under different speeds are shown in Figure 2.11, right.

Figure 2.11, left, shows the performance of our system under different robot speeds. When lower-dimensional descriptors are used, we find that the performance drops quickly as the speed increases. However, if we are willing to increase the feature dimensionality, the system can handle more motion blur.

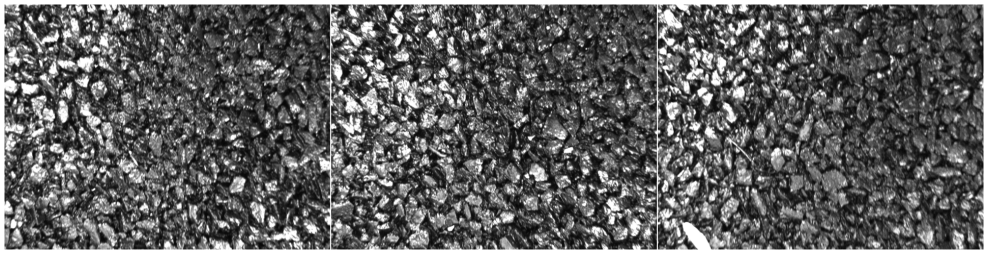


Figure 2.12: Coarse asphalt after rain (compare to Figure A.7). The added specularity causes our localization system to fail.

Effectiveness at Night: As opposed to differences in intensity in the image as a whole, as considered in the previous experiment, we have observed that changing the *pattern* of illumination results in a dramatic decrease in localization performance. In particular, we conducted an experiment in which we captured a dataset (fine asphalt) under direct illumination from sunlight (i.e., without the light shield shown in Figure 2.1), then captured our testing dataset at night, with illumination provided by LEDs. Because of the difference between directional sunlight and LED illumination, the features that were extracted in the two datasets were completely different, since a majority of the features in this surface are due to shading, not color variation. As a result, we were able to find almost no matches between the query images and the database.

To resolve this issue, we captured a second map (Figure A.6) with the light shade in place and using LEDs to illuminate the ground all the time, even during daylight. Comparing this dataset to Figure A.5, we see a significant difference in the appearance of small-scale features in the closeup at lower left. Localizing our nighttime-captured test set in our daytime-captured but light-controlled map, we achieve a localization success rate of 94.50%, showing that blocking the environment light is an effective way to guarantee the consistency of test images and database images.

Wet Surfaces: Another change that can cause dramatic differences between features extracted at different times is wetting of the surface. Figure 2.12 shows the

Table 2.5: Computational time breakdown for localization using 8- and 16-dimensional descriptors, measured on the carpet dataset.

Operation	**Time-8 (ms)**	**Time-16 (ms)**
SIFT extraction	61.30	67.52
NN Search	109.64	119.64
Voting	18.69	18.68
Pose Estimation	10.44	12.09
Others	45.51	43.63
Total	245.58	261.56

same surface as Figure A.7, but captured after a rainfall. The difference in surface specularity, combined with the bumpiness of the surface, caused completely different features to be detected, leading to almost complete failure of our system. (In contrast, we obtained very good results matching wet concrete to a dry-concrete database.)

2.6.5 Efficiency

Our system is implemented on a battery-powered mobile platform having computing power an order of magnitude lower than a desktop computer. In our implementation, we use SiftGPU [102] to speed up the extraction of SIFT features. We evaluate the efficiency on NVIDIA Jetson TX1, and the computational time breakdown is shown in Table 2.5. Although the system does not run in real time, it nevertheless delivers localization in under a second. This is sufficient to correct drift in odometry, and to re-localize the system when tracking or odometry is lost.

2.7 Application: Automatic Path Following

Our system provides a simple, inexpensive solution to achieve fine absolute positioning, and mobile robots having such a requirement represent an ideal application. In

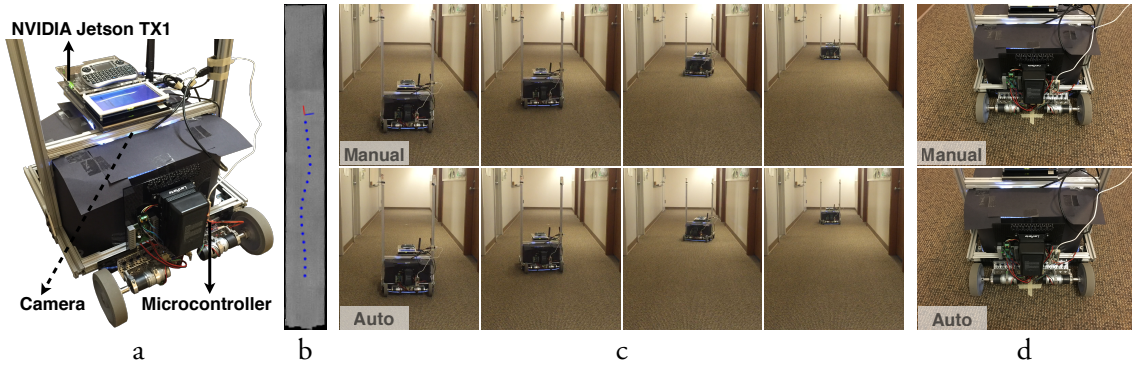


Figure 2.13: A demonstration of path following. (a) Micro-GPS is implemented as a component of a mobile robot. (b) We generate a path by manually driving the robot. (c) We then use Micro-GPS to repeat the path. Screen-shots captured under manual and automatic driving modes are highly consistent. (d) The robot reaches the same ending position with high accuracy.

practice, we believe that indoor service robots, human-free warehouse robots, and self-driving cars performing fine positioning tasks can benefit from our system.

To demonstrate the practicality of this approach, we build a robot that is able to follow a designed path exactly without any initialization of the position. Our robot (shown in Figure 2.13a) has a differential drive composed of two 24 V DC geared motors with encoders for closed-loop control of the motors. A Teensy Arduino board is used for the motion control. Using the encoder readings, we implemented dead-reckoning odometry on board so that it is feasible to track the position of the robot at reasonable accuracy within a short distance.

The drift in odometry is corrected using Micro-GPS running on the on-board NVIDIA Jetson TX1 computer, which communicates with the microcontroller via a USB connection. When a video frame is captured, the microcontroller is notified to save its current estimate of position, and to continue tracking the robot using dead reckoning. When Micro-GPS computation is finished, the ground-truth location of the frame is communicated to the microcontroller, which can update the estimate of its current location by applying the intervening tracking results to that ground-truth position.

a b c

Figure 2.14: Recording footprints. (a) The marker on the shoe is calibrated relative to the marker on the ground. (b) Captured image after performing perspective correction. (c) Recorded footprints.

To test the repeatability of navigation using this strategy, we first manually drive the robot along a particular path (shown in Figure 2.13b), and mark its final location on the ground using a piece of tape. We then send the robot back to its starting position, and ask it to re-play that same path. The sequences of the manual driving and automatic re-play are shown in the accompanying video; screen-shots from the video are compared in Figure 2.13c. As shown in Figure 2.13d, the robot ends up in almost exactly the same position after automatic path following as it did after the manual driving.

2.8 Application: Capture of Foot Placement

The applicability of our system is not limited to the field of robotics. Micro-GPS can enable many more applications when tiny, lightweight, wearable cameras foretold by GoPro become prevalent. One of the possibilities is precise recording of foot placement, for performance capture as well as a range of HCI and IoT applications.

We attach an iPhone low on a person's leg to capture ground images around the foot (Figure 2.14a). An ArUco marker [62] is fixed on the shoe to help in estimating

the orientation of the camera. We calibrate the intrinsics of the camera and its pose relative to another marker on the ground.

We record video at normal shutter speed while the person walks. The camera pose and metric scale of each frame can be estimated via the captured marker on the shoe. We warp the image to remove perspective foreshortening and match the scale of the database images (Figure 2.14b). Note that this correction cannot be performed when the person is moving, because the calibration is for a foot placed on the ground. Finally, Micro-GPS can be directly applied to the warped images, and Figure 2.14c shows that every foot step is successfully localized (to millimeter-level accuracy) in the map.

2.9 Discussion, Limitations, and Future Work

We present a low cost texture-based global positioning system that can provide millimeter-level precision. Our system does not require adding infrastructure to the environment, and we demonstrate that it can be implemented on a low-power platform such as those embedded in mobile vehicles. A variety of applications that require fine global positioning can benefit from the proposed approach.

Scalability: To accommodate larger working areas, we would need to increase the volume of the database, which could degenerate the robustness of our system due to noisier feature matching. Also, performing matching within a large database could raise the issue of efficiency. However, our system could work together with existing systems that provide coarse localization (e.g., GPS) to narrow down the working area.

Limitations in Feature Extraction: We have found (Table 2.1) that using SIFT features in the proposed system results in robust performance in most cases. One exception is the wood dataset, in which features are more difficult to find. In particular,

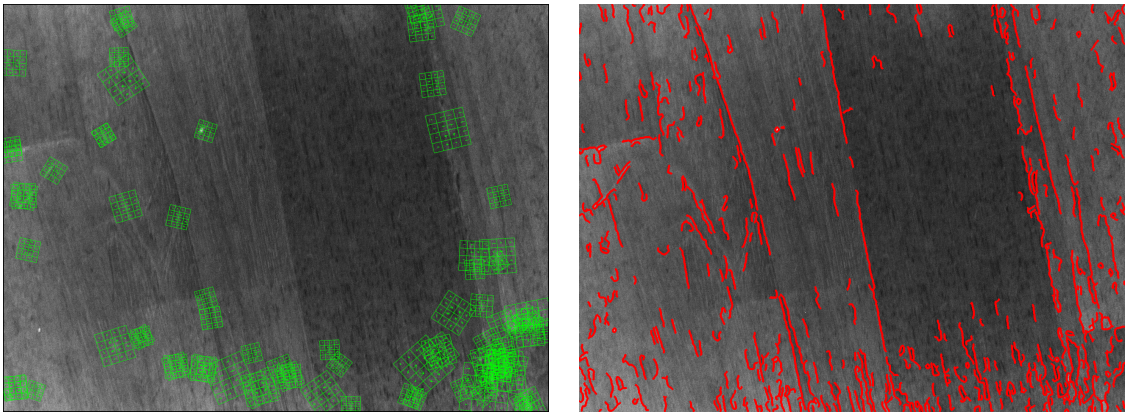


Figure 2.15: Performance on the wood floor is limited by the lack of SIFT features (left). As future work, we suggest using (a subset of) the detected edges instead (right).

the number of extracted features per frame in the wood dataset can be as low as 81 (Figure 2.15a), as compared to over 2000 on the carpet dataset. This is because the textures on wood are mainly composed of stripes, not "blobs."

This difficulty in feature detection leads to lower performance. Although we report a success rate of 75% in Table 2.1, we have determined by hand that this is an underestimate due to not having enough features for verification – the true success rate is 88%. Still, as future work we would like to explore combining the SIFT detector with others, such as those based on Canny edges (Figure 2.15b).

Accuracy: The actual localization in global space is determined by the alignment of the database images. Most of the maps we have obtained are visually plausible and of high quality because of the global optimization process which utilizes a large number of loop-closure pairs. However, even with global optimization, the alignment can still drift slightly as the map grows. A solution to that is to put sparse visual anchors in the map and incorporate these anchors into optimization as hard constraints. On the other hand, in most applications it is sufficient to locate the live frame in the pre-built map, even though the pre-built map does not perfectly match the real world. For example, when a warehouse robot is asked to navigate to a shelf in the map, driving

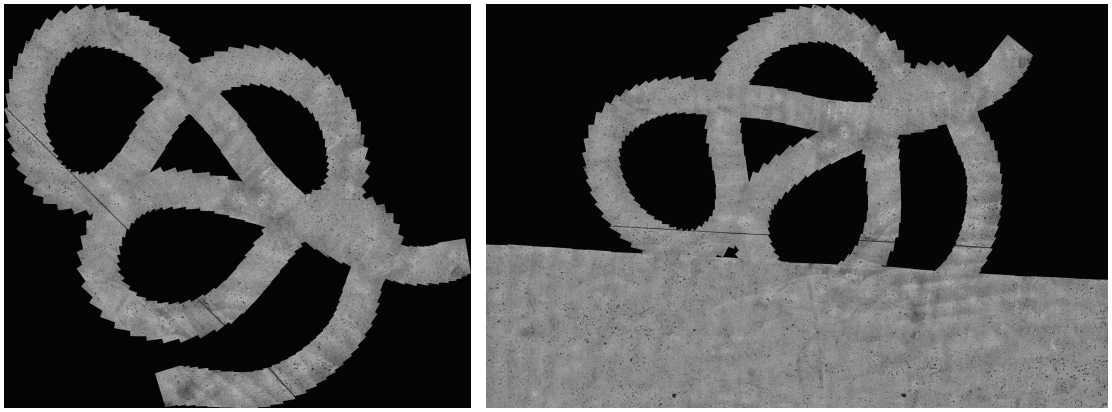


Figure 2.16: Map expansion. Left: Self-consistent map generated by stitching the test sequence. Right: The updated map generated by registering the new sub-map to the existing map.

the robot until it reaches the destination in the map is more important than knowing the location of the shelf in the real world.

Map Expansion: A long-term goal of this work is to enable crowdsourced building and updating of large-scale maps. Although it will be difficult to apply the existing global registration methods at global scale, this is not necessary if any rough location estimate is available. For example, outdoors it should be possible to obtain a GPS estimate accurate to, say, 10 m under almost all circumstances. Indoors, WiFi triangulation can be used to obtain an initial position estimate accurate to a few tens of meters. Therefore, each "map" need only cover an area of a few hundred square meters, which is well within the practical range of our system given that we perform dimensionality reduction on features and accelerate nearest-neighbor queries.

The remaining challenge is therefore a systems-building one: developing a back-end that will enable newly-acquired images to be incorporated into the database. We have conducted an initial experiment to evaluate the practicality of aligning a new set of images and optimizing their poses relative to an existing database, as long as there is some overlap between the new dataset and the existing map. Figure 2.16 shows a demonstration on the concrete dataset. We first use the image stitching pipeline

detailed in Section 2.3 to obtain a new globally consistent map (shown at left) of only the newly-acquired data. We then run Micro-GPS on each test image and register the new map to the existing map once Micro-GPS confirms a successful localization. Lastly, we find more correspondences between the existing map and the new map and optimize the test image poses again. The updated map after registering the new map to the original map is shown in Figure 2.16, right.

Efficiency: SIFT feature extraction makes our system far from real-time when implemented on a mobile platform, even with the GPU implementation. Replacing SIFT with a much more efficient feature detector can probably help. In particular, we would like to investigate deep-network-based approaches that are specialized to the domain of texture-like images, as opposed to the existing work on natural images.

Robustness: While we have shown that our system is robust to perturbations in both image capturing and changes to the texture, it does not work well with wet surfaces because the appearances of wet surfaces often change significantly. Features extracted from wet surface images can be completely different from those extracted from dry surface images, which is an even worse case than severe occlusion. A possible future direction is to capture the geometry (i.e., normal maps) of the ground texture in a manner similar to the previous work on fingerprinting blank paper [17]. Another possibility would be to use cross-polarized lighting to eliminate the specular reflections. In case a reliable ground surface is absent, leveraging subsurface features, such as what has been demonstrated in LGPR [18], could be a feasible solution.

In our current pipeline, we build our database by randomly selecting 50 features from each database image. However, some of the features might be non-repeatable and cannot be observed again. Note that features with higher LoG response are not necessarily highly repeatable features: they are just as likely to be due to noise, dust, etc. Intuitively, features with medium responses are likely to be highly repeatable.

Hartmann et al. [26] train a classifier to select repeatable SIFT features, but we have found that the correlation between a SIFT descriptor and its repeatability score using this method is not strong on our "texture-like" images. In the future, we would like to investigate predicting repeatable features based on the image patches, and to include only those in the database.

Chapter 3

Learning to Detect Features in Texture Images

3.1 Introduction

Many computer vision tasks require computing local features as the first step. These tasks include but are not limited to image alignment [115, 14], image retrieval [72, 34], image-based localization and reconstruction [86, 87]. A pipeline used for computing local features given a single input image typically consists of a **feature detector** and a **feature descriptor**, which are often performed sequentially. For both of these two important components, there is a substantial amount of work on designing hand-crafted solutions, and some of the best-performing hand-crafted pipelines such as SIFT [50] have become gold standards in real applications. Nevertheless, nearly all existing hand-crafted solutions are optimized for and evaluated on natural images. There are, however, many more types of images that are outside the scope of natural images, but also require a well-performing feature pipeline. In Chapter 2, we have demonstrated in the Micro-GPS project that precise global localization can exploit *textures* (such as those present on ground surfaces ranging from carpet to asphalt),

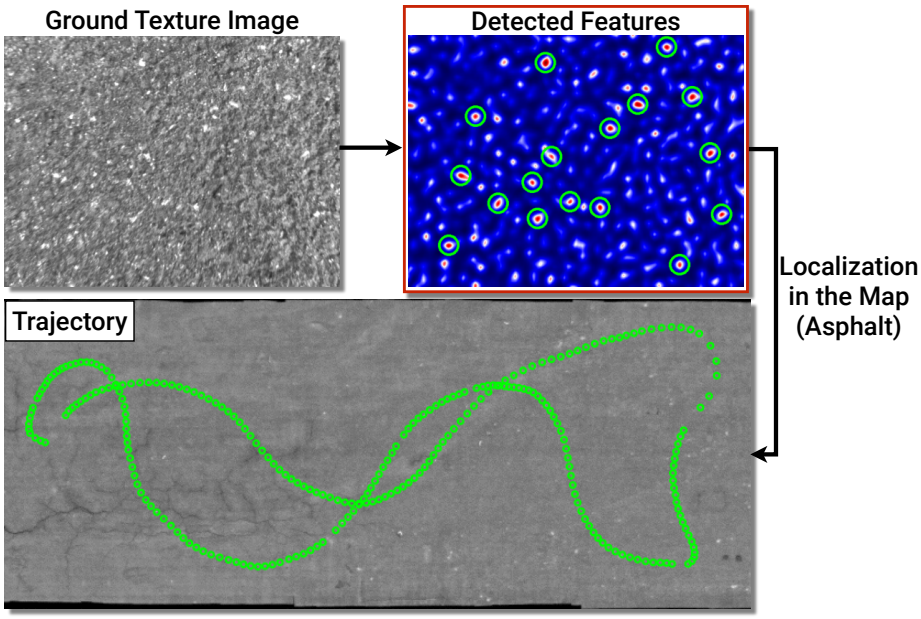


Figure 3.1: From each test image, our proposed detector extracts highly repeatable features, which can be utilized by Micro-GPS to achieve precise global localization in a pre-built map, such as the asphalt map being shown. Note that Micro-GPS locates each test image *independently* in the map (ignoring temporal coherence).

because textures globally exhibit many distinctive and persistent features that can be used for unique identification. While a hand-crafted feature pipeline such as SIFT indeed works properly in most of the textures demonstrated in the Micro-GPS system, its robustness varies on different textures. We observe that different textures consist of different "basic elements" that are not always ideal for a particular feature detector. For example, a typical blob or corner detector will find few features on the stripes present in a wood texture. In this work, we propose a learning-based feature detector optimized for texture images, and demonstrate its effectiveness in the Micro-GPS system(Figure 3.1).

In recent years, machine learning techniques have shown success in improving feature pipelines. Most of this previous work, however, focuses on learning feature descriptors [84, 24, 83, 110], which we argue is a more straightforward problem than learning a feature detector. While it is difficult to obtain ground-truth labels for both feature detection and feature description, learning a feature descriptor often re-

lies on *self-supervision*, which is usually achieved by training with corresponding and non-corresponding image patches. Exploiting such self-supervision is more natural for learning a feature descriptor, because it exactly matches how a descriptor is evaluated — minimizing the distance between corresponding descriptors and maximizing the distance between non-corresponding descriptors. In contrast, it is less obvious how to perform similar self-supervision when learning a feature *detector*, because the criteria used for evaluating a detector are very different. In general, a detector is considered well-performing if the interest points output by the detector 1) can be repeatedly detected even when the image undergoes certain transformations such as rotation; 2) are locally distinctive, which means that they cannot be easily confused with other nearby points; and 3) are sufficiently numerous to be useful in applications such as retrieval, registration, and localization. The combination of these criteria cannot be easily translated into a loss function to be used in (self-supervised) training.

Recent work has demonstrated that a feature detector can indeed be learned in an unsupervised manner [42, 77] without using existing feature detectors for supervision. Specifically, these methods attempt to output a "response map" with each pixel indicating how interesting the pixel is, and the only constraint used in these methods is that the response map should be *consistent* under certain criteria when the image undergoes transformations. Because this leads to only weak constraints, some work has proposed to add supervision by, for example, constraining the response map to fire on points output by a hand-crafted detector [113]. Although this approach was shown to work, it is limited by the effectiveness of the underlying TILDE detector [96], which our experiments show has only moderate performance for the texture images on which we focus.

We propose an approach to feature detection that retains self-supervision, augmenting the goal of consistency (as expressed by the "ranking loss" of Savinov et al. [77]) with the desire to make the response map as "peaked" as possible. This

leads to more localized local extrema in the response map, which in turn boosts the repeatability of the detector. Our detector also utilizes a fully-convolutional network architecture with a large receptive field, leading to both high efficiency and suitability for a wide variety of textures. Finally, and most crucially, we demonstrate that textures are sufficiently varied that optimal performance for each type of texture can be achieved by training on that texture alone. While we do evaluate the repeatability of detectors trained on one texture and tested on another, our main focus is on answering: *given sufficient amount of images of a specific texture, is it possible to learn a "perfect" feature detector for this texture alone?*

The major contributions of this work are:

- Proposing a fully-convolutional network architecture that can be efficiently applied on a full-sized image without separate evaluation on each pixel.
- Describing a method to maximize the *peakedness* of the response map and proving that it is critical to improving the repeatability of the learned detector.
- Evaluating design choices, demonstrating that maximal effectiveness requires a large receptive field but is relatively insensitive to the tuning of other parameters.
- Demonstrating on Micro-GPS that the learned detector is more effective than a handcrafted alternative.

The materials in this chapter were publicly presented [112] prior to the completion of this thesis.

3.2 Related Work

While classic features detectors were hand-crafted using a human-specified definition of "interestingness", recent ones use learning to improve performance. These either

select a subset of points output by a hand-crafted detector or learn a definition of "interestingness" from scratch.

3.2.1 Hand-Crafted Feature Detectors

There is extensive work on designing a feature detector that performs well on natural images. Detecting corners is one of the earlier strategies [25, 56]. Alternatively, one can detect "blobs". The SIFT detector [50] uses Differences of Gaussians (DoG) to approximate the Laplacian-of-Gaussian (LoG) filter, and looks for local extrema over scale and space. SIFT has shown great robustness in real-world applications and remains a gold standard. A major limitation of SIFT is its speed. While one can use GPUs to accelerate SIFT [102], SURF [13] approximates LoG using a box filter and significantly speeds up detection. As an alternative to SIFT, MSER [53] detects blobs by extracting covariant regions from the image and fitting ellipses to these regions. In addition to detecting blobs, SFOP [22] also detects junctions. WADE [73] demonstrates that salient symmetries can be leveraged to detect repeatable interest points even in images related to untextured objects, which are difficult cases for a corner or blob detector.

3.2.2 Learned Feature Detectors

FAST [70] achieves fast corner detection, and machine learning techniques are applied to accelerate detection. By minimizing the pose estimation error for stereo visual odometry, one can learn a convolutional filter (LCF) for feature detection [69]. A classifier can be learned from SIFT features surviving matching tasks, and combining the classifier with the original SIFT detector helps achieve better matchability [26]. TILDE [96] uses stack of pre-aligned images undergoing drastic brightness changes and learns a detector to predict highly repeatable SIFT keypoints across images. Similarly, LIFT [106] also uses patches corresponding to SIFT keypoints to train

their feature detector. Lenc and Vedaldi [42] show that it is possible to train a feature detector using the covariant constraint only, by forcing the network to output covariant transformations given an image patch and its transformed version. This work was extended by using features detected by TILDE as guidance [113].

Savinov et al. [77] propose to learn a detector by ranking image patches. This method is based on the assumption that if a patch has a higher score than another patch in the response map, this ranking relationship should remain unchanged when the image is transformed. While this method shows improvement over previous detectors, and we incorporate its ranking loss into our work, it has two major limitations. First, the network is underconstrained because only ranking loss is used. Guaranteeing a low ranking loss does not necessarily imply high repeatability of the detector, and we show that this is data-dependent. Second, all network architectures demonstrated in the original paper are constrained by a small receptive field (17×17 kernel in their work). The small receptive field is inadequate for many of the textures we consider. In addition, the deep convolutional architecture, which shows the best performance in the cross-modal detection task, requires per-pixel traversal in the test image. We ameliorate these problems by incorporating a peakedness loss term that effectively improves repeatability, and using an efficient fully-convolutional network that can be directly applied to the test image.

3.3 Approach

Defining the desired appearance of a feature in texture images is a non-trivial task because even humans cannot reliably label repeatable features. Hand-crafted detectors have one definition of "feature," and fail to work when textures do not fit that definition. For example, wood textures often contain stripes that cannot be identified by a blob detector or a corner detector. Since there is no obvious way to obtain labels

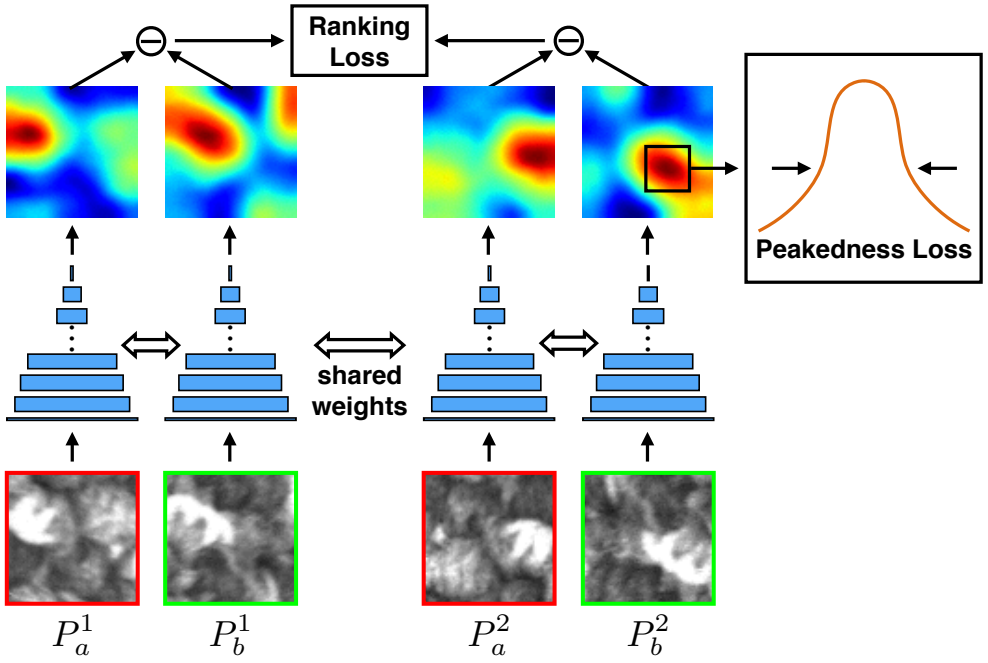


Figure 3.2: Illustration of the training pipeline. The network is pretrained with the ranking loss only, and tuned using both the ranking loss and the peakedness loss.

to enable supervised learning, we choose to perform unsupervised training, following the basic idea of Savinov et al. [77].

3.3.1 Feature Detection by Ranking

The network we train will take as input an image, and produce as output a *response map*. Each value in the response map indicates how likely this pixel is to be a distinctive interest point. Specifically, we would like the deep-neural-network to learn a scoring function $\mathcal{F}(.)$ that maps the input image patch to a single-valued score. We want this score to be consistent under transformations: if one point is more "interesting" than another, then it should still be more interesting when the image is transformed. That is, given any pair of patches $\{P_a^1, P_b^1\}$ and their randomly transformed versions $\{P_a^2, P_b^2\}$, we want the scoring function to satisfy either of the

following two cases:

$$\begin{cases} \mathcal{F}(P_a^1) > \mathcal{F}(P_b^1) \quad \text{and} \quad \mathcal{F}(P_a^2) > \mathcal{F}(P_b^2) \\ \mathcal{F}(P_a^1) < \mathcal{F}(P_b^1) \quad \text{and} \quad \mathcal{F}(P_a^2) < \mathcal{F}(P_b^2) \end{cases} . \tag{3.1}$$

Combining into a single inequality:

$$\mathcal{R} = \big(\mathcal{F}(P_a^1) - \mathcal{F}(P_b^1)\big)\big(\mathcal{F}(P_a^2) - \mathcal{F}(P_b^2)\big) > 0. \tag{3.2}$$

This inequality ensures consistent ranking of P_a and P_b, and we apply a hinge loss to obtain our *ranking loss* term:

$$\mathcal{L}_{\text{rank}}(P_a^1, P_a^2, P_b^1, P_b^2) = \max(0, M_{\text{rank}} - \mathcal{R}), \tag{3.3}$$

where M_{rank} is the margin and correlates to the confidence of ranking consistency. We set $M_{\text{rank}} = 1.0$ throughout our experiments. In each training iteration, the network takes in two pairs of corresponding image patches which are randomly rotated. An illustration of the ranking network is shown in Figure 3.2.

In selecting an appropriate architecture for the ranking network, we take into consideration both performance and efficiency. Performance is strongly affected by the size of the receptive field. While one could let the network see a larger area by downsampling the original image, significant downsampling erases subtle details. We show that while a small window can indeed work in some "easy" texture images, a larger window is critical to better adaptability. Our network has a receptive field of 65×65, leading to improved performance relative to the original networks of Savinov et al. [77], which have a receptive field of size 17×17. When considering efficiency, we notice that the best-performing architecture (Deep convolutional network) used by Savinov et al. requires per-pixel traversal of the input image. We instead use a fully-

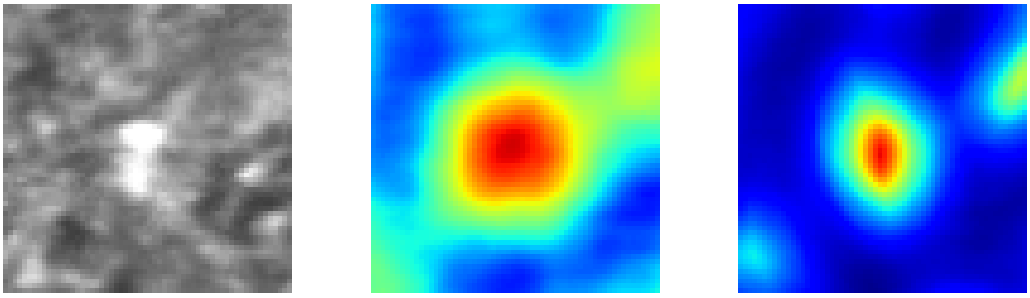


Figure 3.3: From left to right: zoomed-in view of a stain in the asphalt texture, response map output by the network trained using ranking loss only, and response map after optimizing the peakedness. The latter response map is more robust against noise when performing non-maximum suppression.

convolutional network containing only convolutional layers and rectified linear units (ReLU), with no pooling or padding. This architecture is efficient because the trained model can be directly applied to the whole image, yielding a complete response map.

3.3.2 Optimizing Peakedness of the Response

Detection purely by ranking has several limitations due to its unconstrained nature. First, the the loss is unchanged if the ranking of image patches is flipped. In theory, this is not a problem because we can consider both local maxima and local minima as good candidates for features. In practice, however, we observe that each training run results in a network in which either maxima or minima yield better (and more visually plausible) results. Therefore, we use a validation set to evaluate the repeatability of both maxima and minima, and we negate the response map if the minima perform better. In other words, we always end up with a response map whose local maxima are the features we use.

Another, more serious, limitation of a ranking network is that there are many functions that all result in the same *relative* rankings of different pixels, while yielding different repeatability in matching. For example, consider the patch in Figure 3.3. The response map shown at center is broad, meaning that the feature was not localized

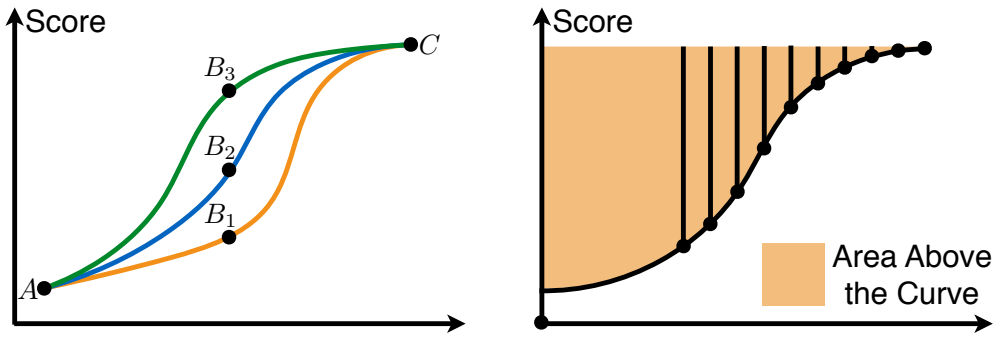


Figure 3.4: Illustration of response curves. Left: different response curves can lead to same ranking. Right: peakedness of the response curve can be evaluated as the area above the curve; specifically, for the k highest values in a local window, we sum up the difference between each value and the maximum.

precisely. In contrast, the response map at right is more "peaky," which ultimately results in more accurate matching across images.

In order to encourage the network to learn the response map at right, we include a peakedness term in our training loss. This is based on looking at all the responses in a patch, and encouraging only a few of them to be large. For example, if we look at the responses within the neighborhood of a maximum, sorted from lowest to highest, different networks might produce any of the three curves in Figure 3.4, left. We prefer the lowest curve, since it has the most-peaked response. This is accomplished by maximizing the *area above the curve* (Figure 3.4, right): this forces the values in the neighborhood to be as small as possible.

Specifically, during training we consider not just 65×65 input patches (which would result in a single output value per patch), but rather patches $\hat{P}$ of size (64+w)×(64+w). Because our scoring network $\mathcal{F}$ is fully convolutional, this results in an output response map of size $w \times w$, where w =7 in our experiments. We sort the scores in this map to obtain a response curve $\tilde{\mathcal{F}}(\hat{P})$. The peakedness of the patch is computed as the difference between the maximum and the average of the largest k scores:

$$\gamma(\hat{P}) = \max\big(\tilde{\mathcal{F}}(\hat{P})\big) - \frac{1}{k}\sum_{i}^{k} \tilde{\mathcal{F}}(\hat{P})_i. \tag{3.4}$$

The *peakedness loss* function forces the peakedness to be above a certain margin:

$$\mathcal{L}_{\text{peak}}(\hat{P}) = \max\big(0, M_{\text{peak}} - \gamma(\hat{P})\big), \tag{3.5}$$

where M_{peak} is the desired peakedness (3.0 in our experiments). We compute the total training loss by averaging the peakedness loss of the four input patches, and combining with the ranking loss:

$$\begin{aligned}\mathcal{L}_{\text{total}}(\hat{P}_a^1, \hat{P}_a^2, \hat{P}_b^1, \hat{P}_b^2) = \mathcal{L}_{\text{rank}}(P_a^1, P_a^2, P_b^1, P_b^2) + \\ \alpha \cdot \tfrac{1}{4}\big(\mathcal{L}_{\text{peak}}(\hat{P}_a^1) + \mathcal{L}_{\text{peak}}(\hat{P}_b^1) + \mathcal{L}_{\text{peak}}(\hat{P}_a^2) + \mathcal{L}_{\text{peak}}(\hat{P}_b^2)\big),\end{aligned} \tag{3.6}$$

where $\{P_a^1, P_a^2, P_b^1, P_b^2\}$ are the center 65×65 patches of $\{\hat{P}_a^1, \hat{P}_a^2, \hat{P}_b^1, \hat{P}_b^2\}$ and α is a weighting parameter used to balance the ranking loss and the peakedness loss. Note that within each batch, not all the patches strongly correlate to good features, thus it can be harmful to maximize the peakedness for patches with relatively weak responses. Therefore we exclude the 25% of patches with the smallest maximal scores when computing the peakedness loss.

3.3.3 Implementation

3.3.4 Datasets

The datasets used in Micro-GPS consist of texture images densely captured along overlapping paths and registered to each other. Previous works usually train their networks using patches cropped around SIFT features that survive a structure-from-motion pipeline [106], because this is perhaps the most efficient way to generate corresponding image patches. Savinov et al. [77] use the DTU Robot Image Dataset [1], for which ground truth 3D points are available, but such a dataset is difficult to acquire

Table 3.1: Architecture of the proposed scoring network. Note that "Conv3-9" means repeating the layer for 7 times.

Layer	Conv Filter	Input Size	# Input Channels	# Output Channels	Activation	Stride
Conv1	9×9	65	1	16	ReLU	1
Conv2	7×7	57	16	32	ReLU	1
Conv3-9	7×7	51	32	32	ReLU	1
Conv10	9×9	9	32	32	ReLU	1
Conv11	1×1	1	32	32	ReLU	1
Conv12	1×1	1	32	1	ReLU	1

and its size is small considering the broad space of natural images. The advantage of the texture datasets we use is that generating a pair of corresponding patches is as simple as cropping around any point in the overlapping region, and, more importantly, these patches are not biased to any existing feature detector. We use one region in each texture for training and the others for validation and testing. We only crop image patches from image pairs with an overlapping area over 40%. For training, we randomly crop 512k pairs of corresponding image patches with random orientations, which aims to make the scoring network rotation-invariant. In each training iteration, the quadruple used is constructed by randomly combining two pairs of corresponding patches.

3.3.5 Training

Our network consists of 12 convolutional layers, with no padding or pooling: detailed architecture can be found in Table 3.1. We train our network on an NVIDIA M40 GPU using the PyTorch framework [65], using the Adadelta algorithm [108] to minimize loss. The batch size is 256 and the network is trained using only the ranking loss for 10000 iterations. The network is then tuned with both ranking and peakedness loss for 2000 iterations. We observe that this schedule leads to better performance than training from scratch with both losses. We also observe that batch normalization

blurs the response map and lowers repeatability, so we omit it. Training the network typically takes three hours.

3.3.6 Feature Detection in a Test Image

Given a test image of arbitrary size H×W, we first reflection-pad the image by 32 pixels on each side, because this is how much our network removes from each border. Applying the network to the padded image results in an output response map of size H×W. We then follow the general interest point localization pipeline detailed in [50]. Gaussian blurring with σ =2 is performed before non-maximum suppression, which effectively prevents multiple detections in a small neighborhood. To limit the number of output interest points, we simply select the n largest local maxima based on score. Finally, we apply sub-pixel localization based on the second-order Taylor expansion of the scoring function to refine the integer-valued interest point locations.

3.3.7 Computational Efficiency

The proposed architecture is efficient, as compared to state-of-the-art methods that require evaluating each pixel separately (i.e., running the network on the neighborhood around every pixel, independently). Our full pipeline runs at 2.5fps (1288×964). In comparison, QuadNet [77] (our implementation) runs at 0.00775fps with batch processing, while our network computed independently per pixel (as opposed to over the entire image at once) would run at 0.00408fps due to a larger receptive field. Handcrafted detectors such as SIFT are faster: the detector portion of SiftGPU [102] runs at 25fps.

3.4 Results

3.4.1 Evaluation Protocol

The test set we used for evaluating repeatability is generated by randomly sampling pairs of overlapping images with an overlap area above 50%. However, the images in the texture dataset tend to have similar orientation, which is not sufficient for evaluating rotation-invariant feature detectors. We therefore randomly rotate one image in each pair and recompute the transformation between the two images. Given two images and the transformation matrix between them, we run the feature detector on each image, select features in the overlapping region, and count detected features as "repeatable" if they are found in both images, within 5 pixels after applying the transformation.

For a fair evaluation, we must ensure that methods are not rewarded for finding too many features (in which case finding a matching feature by chance would be too easy) or too few (in which case the matching percentage may be high even though the number of detected features may be too low for many applications). We therefore keep only the 200 strongest features (if that many exist) in the overlapping region for each image, similarly to what has been done in Zhang et al. [113]. We perform bidirectional matching to prevent multiple features from matching to the same feature in the other image. Finally, we report the *number* of matches, rather than the *fraction* of matching features, to penalize methods that could not detect at least 200 features (in original images of resolution 1288×964 and 1280×720).

3.4.2 Performance

We compare our detector to hand-crafted detectors including SIFT [50], SURF [13], MSER [53], WADE [73], SFOP [22], Harris Laplace [56] (HarrLap), Hessian Laplace [56] (HessLap), Harris Affine [56] (HarrAff) and Hessian Affine [56] (Hes-

Table 3.2: For each method, we show the number of repeatable features detected (across 20 images, keeping a maximum of 200 features per image), the total number of detected features and the overall repeatability represented in percentage.

	indoor-carpet			**indoor-tile**			**indoor-wood**		
Method	# rep	# det	per (%)	# rep	# det	per (%)	# rep	# det	per (%)
HarrAff [56]	394	4000	9.85	706	4000	17.65	34	82	41.46
HarrLap [56]	390	4000	9.75	702	4000	17.55	35	83	42.17
HessAff [56]	447	4000	11.18	84	4000	2.10	507	4000	12.68
HessLap [56]	444	4000	11.10	126	4000	3.15	490	4000	12.25
MSER [53]	590	4000	14.75	1556	4000	38.90	68	115	59.13
FAST [70]	1708	4000	42.70	2276	4000	56.90	1017	4000	25.42
SIFT [50]	2022	4000	50.55	1510	4000	37.75	610	1648	37.01
SURF [13]	2451	4000	61.27	2271	4000	56.77	1455	3974	36.61
LCF [69]	811	4000	20.28	1068	4000	26.70	740	4000	18.50
SFOP [22]	2504	4000	62.60	2328	4000	58.20	1074	3474	30.92
WADE [22]	2898	4000	72.45	1230	1946	63.21	20	61	32.79
TILDE-P24 [96]	2029	4000	50.72	2770	4000	69.25	2380	4000	59.50
TILDE-P [96]	1777	4000	44.42	2625	4000	65.62	2182	4000	54.55
Linear17 [77]	3550	4000	88.75	2730	4000	68.25	1567	4000	39.17
DCNN17 [77]	3589	4000	89.72	2650	4000	66.25	2260	4000	56.50
Pretrained	3290	3681	89.38	3383	4000	84.58	1892	3961	47.77
Tuned	**3715**	**4000**	**92.88**	**3397**	**4000**	**84.92**	**2906**	**4000**	**72.65**

	outdoor-asphalt			**outdoor-granite**			**outdoor-concrete**			**outdoor-coarse**		
Method	# rep	# det	per (%)	# rep	# det	per (%)	# rep	# det	per (%)	# rep	# det	per (%)
HarrAff [56]	350	4000	8.75	223	4000	5.58	495	4000	12.38	466	4000	11.65
HarrLap [56]	351	4000	8.77	224	4000	5.60	494	4000	12.35	468	4000	11.70
HessAff [56]	247	4000	6.17	233	4000	5.83	149	4000	3.72	799	4000	19.98
HessLap [56]	249	4000	6.22	230	4000	5.75	150	4000	3.75	803	4000	20.08
MSER [53]	475	4000	11.88	404	4000	10.10	952	4000	23.80	799	4000	19.98
FAST [70]	1524	4000	38.10	1574	4000	39.35	1802	4000	45.05	1583	4000	39.57
SIFT [50]	1474	4000	36.85	1385	4000	34.62	1869	4000	46.73	1456	4000	36.40
SURF [13]	2879	4000	71.97	2496	4000	62.40	2738	4000	68.45	2125	4000	53.12
LCF [69]	779	4000	19.48	817	4000	20.42	795	4000	19.88	1093	4000	27.32
SFOP [22]	1831	4000	45.77	1993	4000	49.83	2219	4000	55.47	1989	4000	49.73
WADE [22]	2296	4000	57.40	2254	4000	56.35	1870	2775	67.39	1992	4000	49.80
TILDE-P24 [96]	2185	4000	54.62	2607	4000	65.18	3244	4000	81.10	2093	4000	52.33
TILDE-P [96]	1869	4000	46.73	2329	4000	58.23	3039	4000	75.98	1975	4000	49.38
Linear17 [77]	3106	4000	77.65	3143	4000	78.57	3304	4000	82.60	3182	4000	79.55
DCNN17 [77]	3304	4000	82.60	3079	4000	76.98	3006	4000	75.15	2755	4000	68.88
Pretrained	3332	4000	83.30	3384	4000	84.60	2716	4000	67.90	3188	4000	79.70
Tuned	**3344**	**4000**	**83.60**	**3431**	**4000**	**85.78**	**3607**	**4000**	**90.18**	**3314**	**4000**	**82.85**

sAff). We also compare our detector to learned detectors including FAST [70], LCF [69], TILDE [96] (TILDE-P, TILDE-P24), and Quad-networks [77] (Linear17, DCNN17). The TILDE detector is trained using time-lapse image sequences, thus we directly use their trained model. TILDE-P24 is an approximation of TILDE-P. Linear17 stands for the one-convolutional-layer network that regresses a 17×17 patch to a single-valued score. DCNN17 stands for the deep-convolutional network which was originally designed for cross-modal detection; it requires per-pixel traversal of the input image.

We also attempted to retrain the covariant detector [42] and its extended version assisted by "standard patches" using the publicly available code [113], but training did not converge, even on the relatively simple carpet texture. This might be due to the significant difference between natural images used by their methods and texture images. We observe that the image patches cropped from natural images have obvious variations in appearance, but the patches cropped from texture images appear similar to each other.

Table 3.2 shows the total number of repeatable features detected using the above detectors and the corresponding repeatability (in percentage). Each detector is applied to 20 pairs of images for each type of texture, which means that at most 4000 repeatable features can be retained. As expected, none of the hand-crafted detectors perform consistently well on all types of textures, which motivates the need for a texture-specific detector. Similarly to the original Quad-network detectors, our pretrained detectors also suffer from the drawbacks of using only ranking loss. However, on textures which contain large elements, such as the colored chips in the tile texture, our pretrained detector outperforms Quad-network detectors since our architecture has a much larger receptive field. After tuning our detectors by introducing the peakedness loss, our detectors outperform all other detectors on every texture, with particular gains on difficult textures such as wood. It is also worth pointing out that

the TILDE detectors, which are trained using natural images, achieve reasonably good performance on all textures although surprisingly they perform worse on the easiest carpet texture.

The good performance of our tuned detectors can be better understood through the response maps shown in Figure 3.5. Although training using the ranking loss is sufficient on "easy" textures, the network tends to output smooth response maps for challenging textures such as wood and concrete. Adding the peakedness loss forces the network to output a sharper response map, making feature localization more robust to noise. Also note that the response map is reversed on the concrete texture, which explains the significant repeatability gain.

3.4.3 Impact of Parameters

There are three major parameters that can influence the performance of the detector: the weight assigned to the peakedness loss (α), the size of the window used to compute the peakedness loss (w), and the number of pixels used to compute the area above the curve (k). We investigate the effects of these parameters by, without loss of generality, beginning with α=0.5, w=7, and k=20, then varying each parameter independently around this configuration. For each parameter setting, we report how much repeatability the detector gains after the tuning stage.

Varying $\boldsymbol{\alpha}$: The weight assigned to the peakedness loss influences how much the network is willing to increase the ranking loss in order to improve the peakedness of the response function. Figure 3.6 shows that the tuned detector typically reaches the best performance when the weight is set to 0.5. Beyond this, repeatability often decreases because the detector starts to overlook the ranking loss.

Varying $\boldsymbol{w}$: The network must see a sufficiently-large local window in order to learn to maximize local peakedness. Increasing window size too far, however, makes

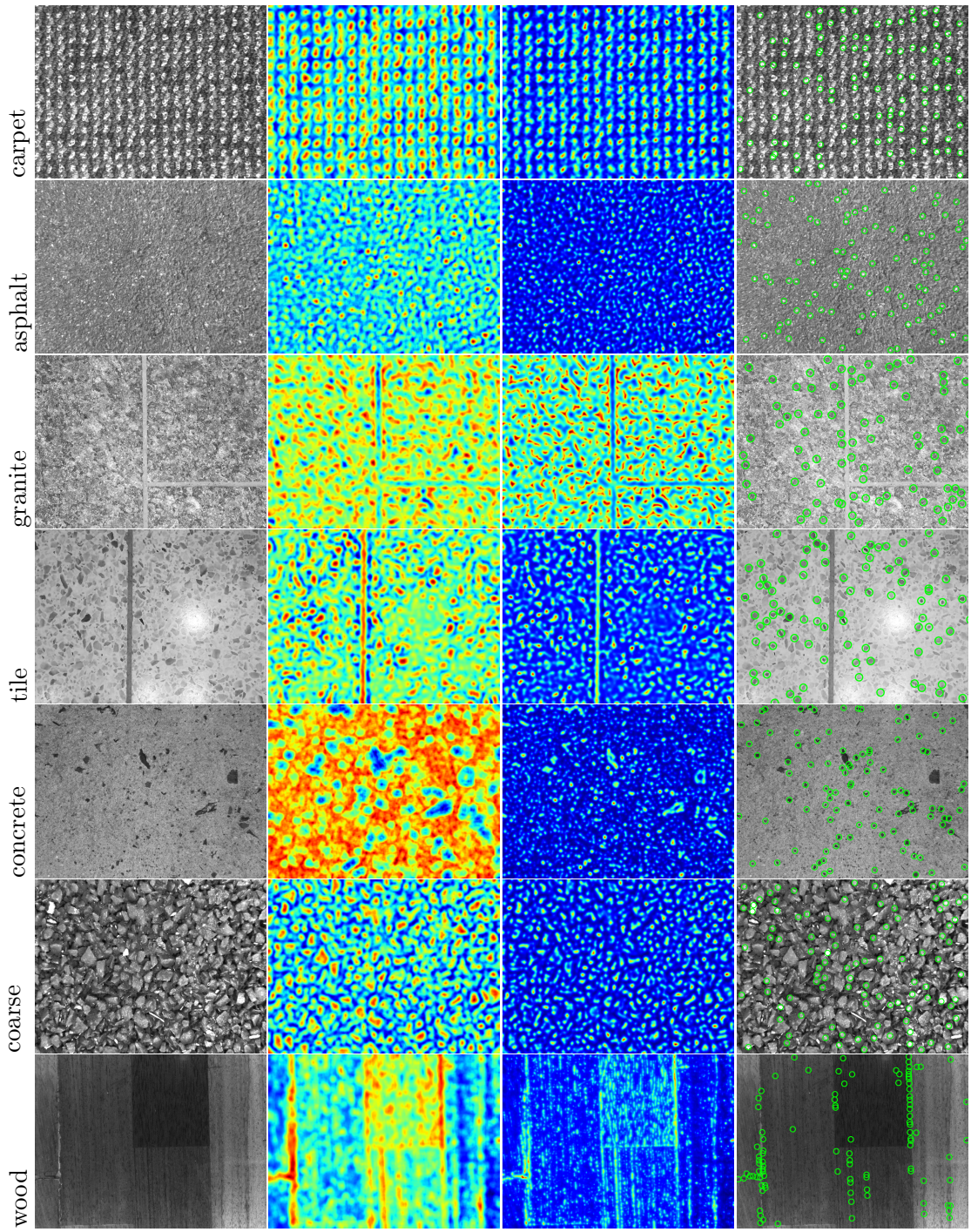


Figure 3.5: Left to right: input, response maps (ranking loss only), response maps (ranking and peakedness loss), top 200 features.

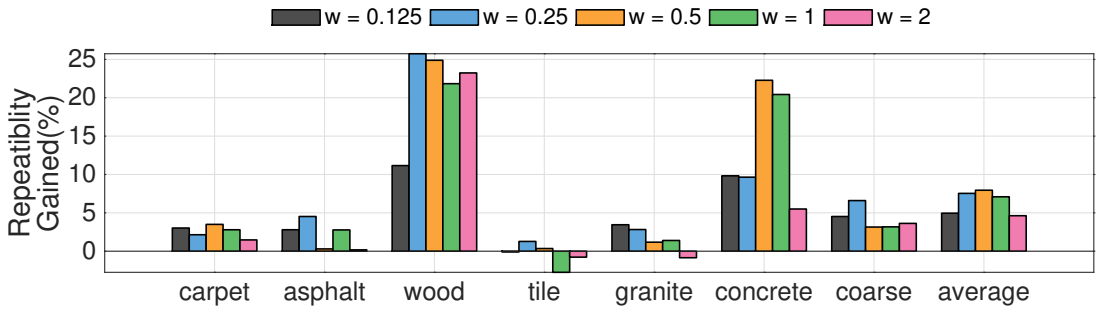


Figure 3.6: Repeatability gain of detectors tuned with different α.

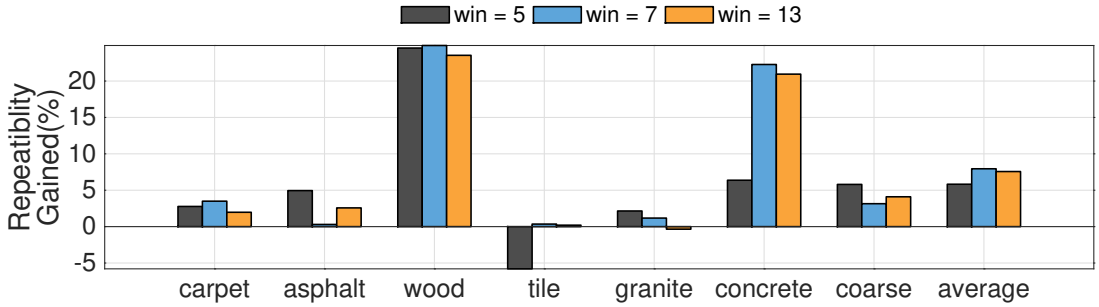


Figure 3.7: Repeatability gain of detectors tuned with different w.

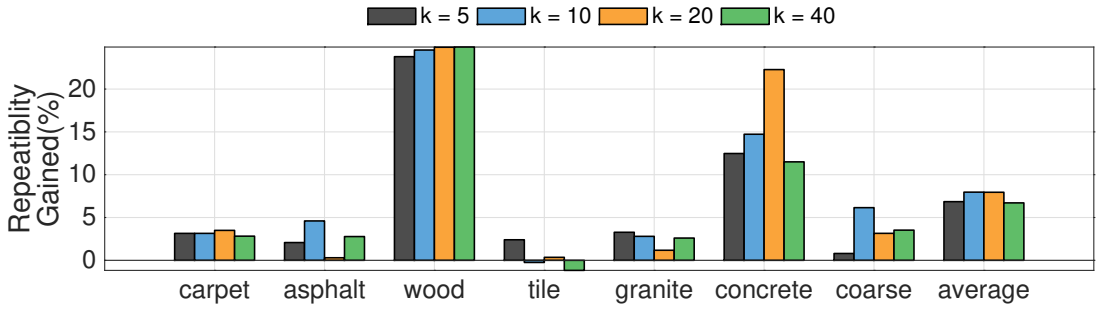


Figure 3.8: Repeatability gain of detectors tuned with different k.

it more likely that the window will contain multiple peaks. We observe that using w=7 results in the best repeatability gain on average (Figure 3.7).

Varying *k*: It is often not a good idea to use all the pixels in a local window for computing the area above the curve. This is because the sorted scores only change drastically in the first k pixels. In Figure 3.8, we observe that using 10 or 20 pixels results in the best repeatability gain on average. Note that when using 20 pixels, the detector trained on the concrete texture yields superior repeatability.

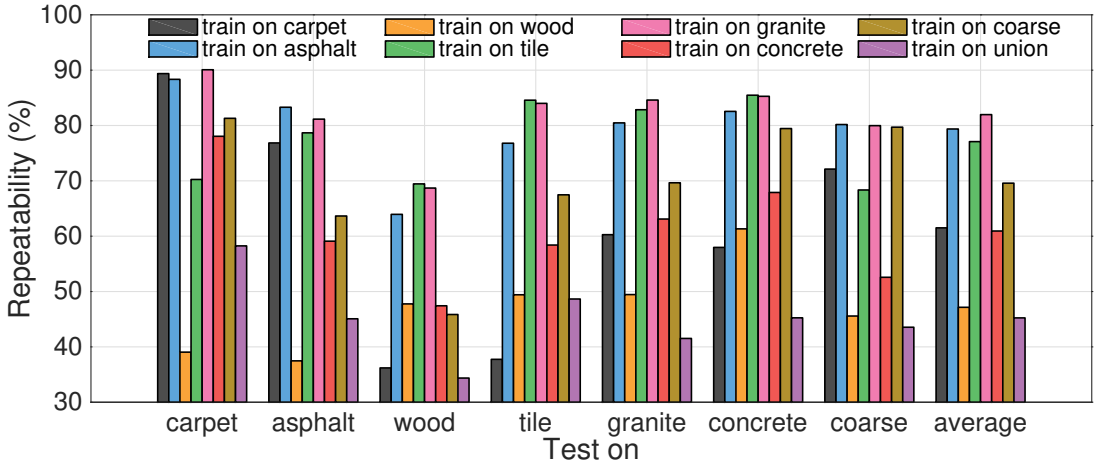


Figure 3.9: Cross evaluation result of the pretrained detectors.

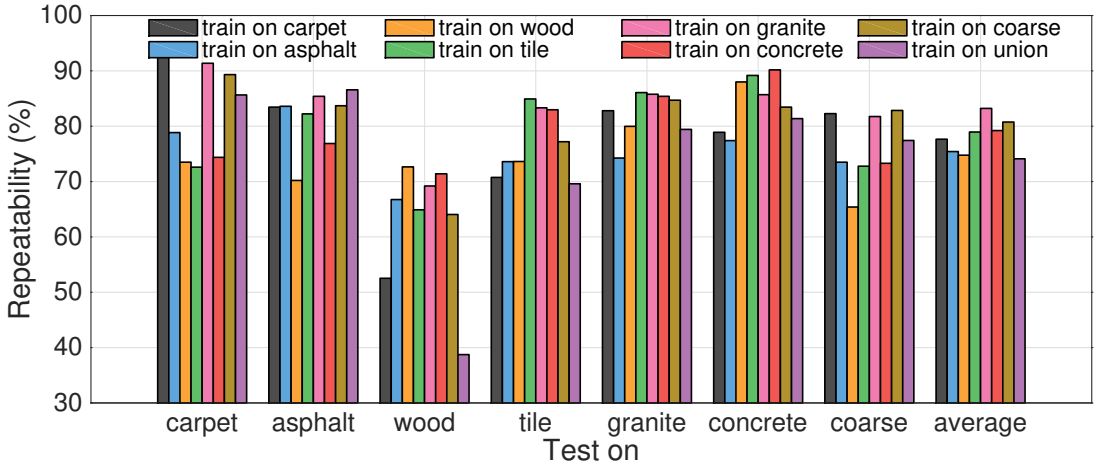


Figure 3.10: Cross evaluation result of the tuned detectors.

3.4.4 Cross Evaluation

An important question one may ask is: *how does a detector trained on one texture perform on a completely different texture?* Another interesting question is: *can we train a good "universal" detector using the union of all texture images?* Below we show the cross evaluation result of both the pretrained models (Figure 3.9) and the tuned models (Figure 3.10). The universal model trained using the ranking loss only is close to unusable. This is likely because the network cannot find a consistent ranking across various textures. However, with the peakedness loss which only requires the universal detector to optimize local maxima, the detector is able to achieve rea-

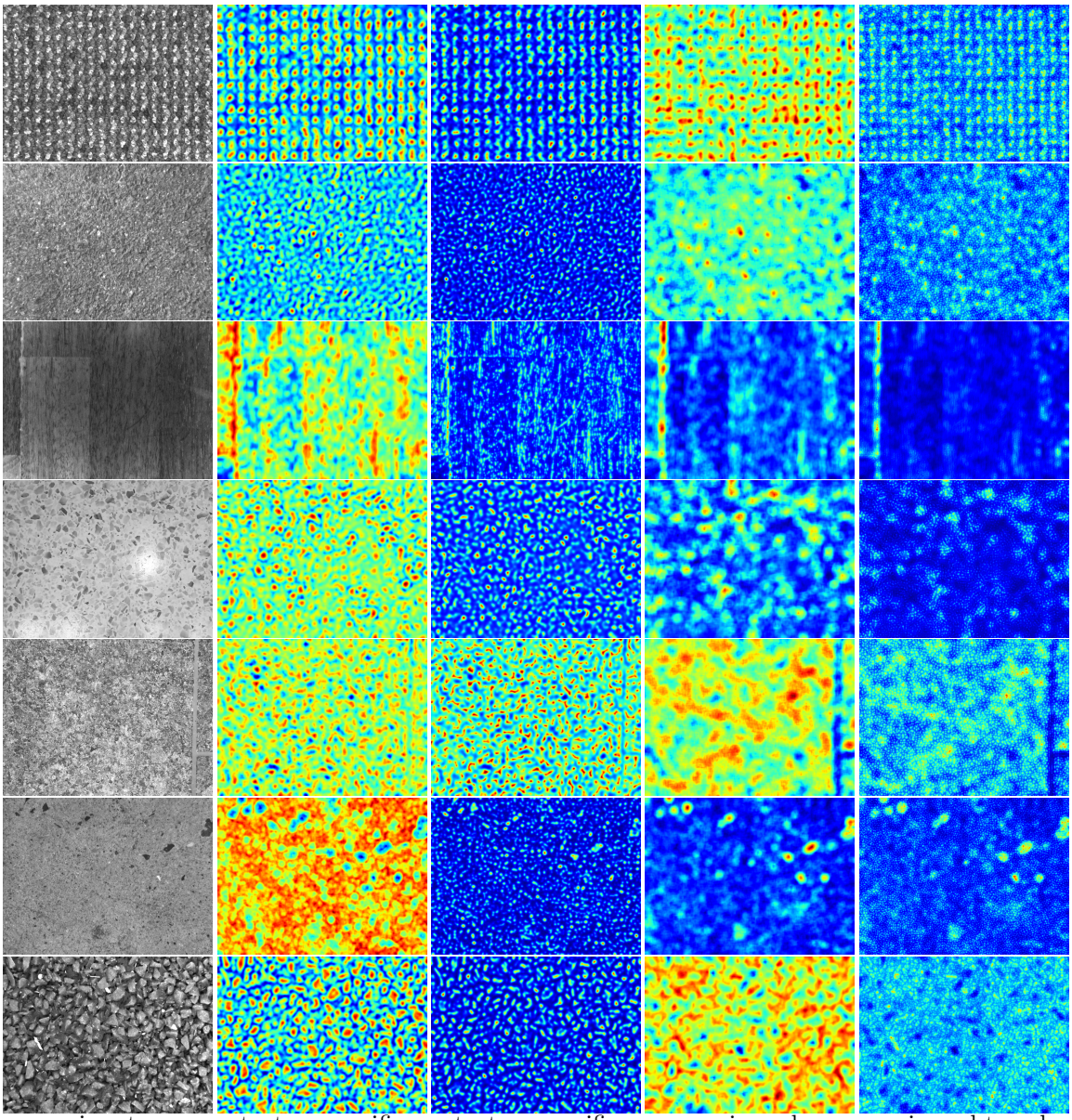


Figure 3.11: Visualization of the response maps generated by networks trained on specific textures and networks trained on the union of all textures. We show both the results of the pretrained model (using ranking loss only) and the tuned model (using both ranking and peakedness loss).

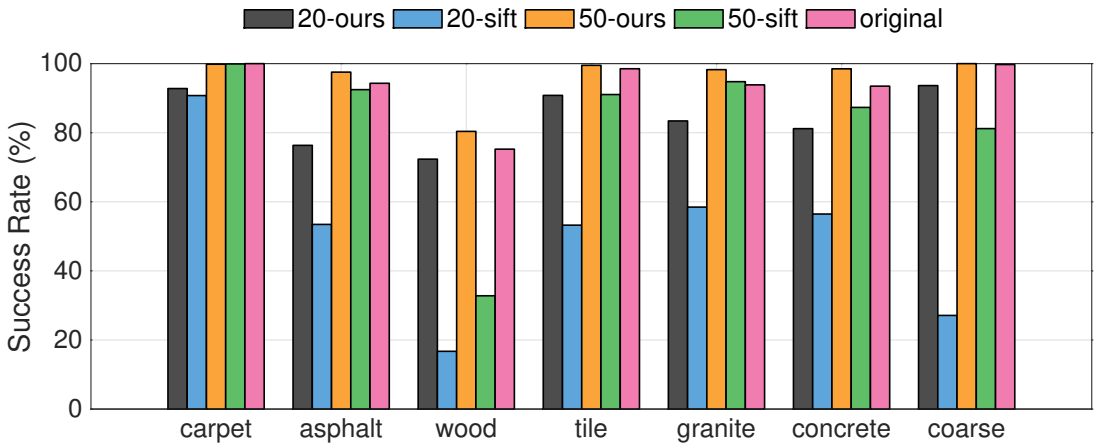


Figure 3.12: Performance on Micro-GPS.

sonably good repeatability, although the wood texture is still too challenging. The poor performance of the universal detector can be better understood through visualizing the response maps. From Figure 3.11, we can observe that the response maps produced by the pretrained universal detector are often overly smoothed. On textures besides the carpet, there exist grid structure which might be learned from the carpet texture. The tuned universal detector, although makes the response map more fragmented, also makes local maxima more "peaky", which explains the improvement in performance. Another interesting finding is that the detector trained on the granite texture performs much better than the universal detector on all types of textures. We observe that both local maxima and local minima found in the response map of the granite texture correspond to very good interest points, which implies that the detector trained on the granite texture potentially has better adaptability. This experiment partially explains why hand-crafted features are still preferred to learned features in real applications in the domain of natural images [74, 81]: using more data for training does not necessarily help learning-based methods generalize well, and using a smaller set of representative data might be a better solution.

3.4.5 Effectiveness in Micro-GPS

In the Micro-GPS system we have presented in Chapter 2, the number of features stored in the database can affect the matching performance because because as the feature space becomes denser, matching becomes not only slower but also more inaccurate due to the larger number of false positives. In the previous SIFT-based implementation, we observe that keeping 50 SIFT features per image is sufficient to guarantee reasonably good performance on across all types of textures. Therefore further improvement of Micro-GPS is mainly constrained by feature detection. Due to the setup of Micro-GPS, there are only three requirements that the feature detector needs to satisfy: high repeatability, rotation invariance, and global distinctiveness. Scale or lighting invariance is not required here, because the images used by Micro-GPS are captured at a constant height (constant scale) and with generally stable illumination.

We combine our detector with the SIFT descriptor [50] and plug into the Micro-GPS pipeline with the compressed descriptor dimensionality set to 16. The feature orientation required by SIFT descriptor is also computed using the orientation estimator of the SIFT pipeline. We compare the SIFT detector and our detector by sampling 50 and 20 features according to the response from each database image, respectively, to build the feature database. The success rate of the system demonstrates the usefulness (distinctiveness and repeatability) of the selected features. Figure 3.12 shows performance under different configurations. The original Micro-GPS clusters features based on their scales, and feature matching is performed within each scale group, which effectively improves the system performance (last bar in each column of Figure 3.12). For a fair comparison, the feature scale detected by SIFT is not used, and the descriptor is computed for both SIFT and our detector using a fixed scale of 6.0. Our learned detector is a clear winner even with feature scale absent.

3.5 Conclusion and Future Work

We present a pipeline for training a feature detector specialized in detecting locally distinctive features in texture images. Explicitly defining what a good feature should look like in texture images is challenging, and so we choose to learn the scoring function in an unsupervised manner. We demonstrate that learning the scoring network purely through a simple ranking loss makes the response curve highly underconstrained, because the response curve corresponding to a desired ranking is not unique. We propose to tune the network by maximizing local peakedness, which significantly improves the repeatability on challenging textures. Moreover, the network architecture we use allows the network to see a much larger area than previous work while guaranteeing testing efficiency by avoiding pixel traversal. Lastly, we show that our detector outperforms SIFT in the "Micro-GPS" texture-based global localization application.

In the future, several immediate directions can be pursued. First, we would like to extend our approach to scale space using a spatial transformer [33]. We experimented with an approach from previous work based on image pyramids [77] and found that some apparently-good features are suppressed by failing to become local extrema in scale space. Second, a locally-distinctive interest point does not imply that the descriptor computed using the point is also sufficiently globally-distinctive for applications that involve feature matching. We believe that a feature descriptor can be introduced to guide feature detection.

Chapter 4

Learning Local Descriptors with Dynamic Soft Margin

4.1 Introduction

Efficient image matching is a fundamental problem in computer vision, robotics, and graphics. Generally, image matching is a two-step procedure consisting of extracting repeatable local keypoints using an interest point *detector*, followed by matching of *feature descriptors* corresponding to those points. For example, the simplest way to describe a local feature is to serialize a local image patch around the detected keypoint, then compare the resulting vectors using Euclidean distance. While this simple descriptor is sufficient in some highly-constrained scenarios, it fails if the illumination or viewpoint is significantly changed. Matching "long" descriptors like this also poses a computational efficiency challenge for many applications, especially those requiring real-time performance. To solve this problem, researchers have attempted to design "short" descriptors. One representative example is SIFT [49], which has proven successful in a variety of applications.

With the development of deep-learning techniques, recent advancements to feature descriptors have been mainly *learning*-based, as opposed to handcrafted descriptors. These learned descriptors are often designed to have the same length (e.g., 128 floats, as with SIFT), but have higher matching performance. Further reduction in storage and matching costs can be achieved by *binary descriptors*, which are interpreted as bit vectors and are compared using Hamming instead of Euclidean distance.

In this chapter, we demonstrate that a common architecture (based on L2-Net [90]) and training procedure (based on HardNet [59]) can be modified to learn not only floating-point but also binary descriptors. The modified network advances the state of the art in descriptor performance, but it also highlights a frequently-encountered problem: the matching accuracy depends on hyperparameter tuning.

In particular, many well-performing learned descriptors are trained using the same loss function: a *triplet loss* that encourages the distance between a negative pair to exceed the distance between a positive pair by some *margin*. The purpose of the margin is to force the network to update its weights using the gradients computed from "harder" triplets, while excluding training samples classified as "easy" by the margin. While modern approaches improve performance by incorporating hard negative mining [10, 59] or regularization [40, 114], the effectiveness of training fundamentally depends on the setting of the margin. Because the optimal margin is problem- and often dataset-dependent, in practice the margin is either specified by hand based on an educated guess or exhaustively tuned at great computational expense.

In this work, we propose a novel triplet loss function that has three major features: 1) We use a *soft* instead of *hard* margin to fully utilize each mini-batch. 2) The soft margin *dynamically* adapts to the current state of training. 3) The method is *parameter-free*: the two goals above are accomplished without the need for any user-tunable hyperparameters. In short, as opposed to the ***Static Hard Margin*** used

in the traditional triplet loss, we think of our method as an instance of a ***Dynamic Soft Margin*** strategy that could be applied to a variety of learning problems.

The traditional triplet loss makes a binary decision of whether a triplet should contribute to the gradient, using the *hard* margin as a constant threshold. We instead make the margin *soft* by using all the triplets in each mini-batch, while following the simple intuition that "difficult" triplets should receive greater weight than "easy" ones. In contrast to previous approaches that use soft margins (such as SVMs with slack variables), our formulation does not require a separate user-tunable parameter for the "softness" of the margin. The dynamic nature of our method is obtained by maintaining a *moving* Probability Density Function (PDF) of the *difference of distances* between the positive and negative pairs in each triplet. This may be thought of as the signed distance of each triplet to the decision boundary. Weights are assigned based on the integral of this PDF, in essence weighting each datapoint proportionally to the probability that it is more difficult to classify than other recently encountered datapoints. Therefore the weighting function is continuously updated as training progresses.

4.2 Related Work

There is a large volume of previous work on designing local feature descriptors. We focus on reviewing descriptors that are widely being used or have recently achieved state-of-the-art performance. Local feature descriptors can be generally categorized into real-valued descriptors and binary descriptors.[1]

Real-valued Descriptors: Probably the most successful handcrafted feature descriptor is SIFT [49], which computes smoothed histograms using the gradient field

[1]The field of image retrieval also uses this terminology, but they emphasize category-level matching instead of instance matching.

of the image patch. Instead of computing the histogram, PCA-SIFT [36] applies principal component analysis directly to the image gradient. DAISY [91] addresses efficiency and is fast enough to support dense matching. Recently, learning-based methods have started to demonstrate effectiveness. Simonyan et al. [84] formulate feature learning as a convex optimization problem. DeepDesc [83] uses paired image patches and adopts a Siamese network to learn a discriminative descriptor while performing hard mining to boost performance. DeepCompare [107] develops a two-stream network with one stream focusing on the central part of the image. TFeat [10] learns the descriptor using a triplet loss and applies an in-triplet hard mining method named anchor swapping. More recently, Tian et al. have proposed L2-Net [90], which adopts a deeper network and designs a new loss function requiring the true matches to have the minimal ℓ_2 distances in the batch. HardNet [59] further simplifies the idea by looking for hard negatives in each batch and achieves state-of-the-art performance using a single triplet margin loss. Instead of using the triplet margin loss as the proxy, DOAP [28] directly optimizes the ranking-based retrieval performance metric and achieves more competitive results using the same network architecture. Keller et al. [37] propose a mixed-context loss that combines triplet and Siamese loss, which performs better than using either one alone. GeoDesc [51] further leverages geometric constraints from multi-view reconstruction and demonstrates significant improvement on 3D reconstruction tasks.

Binary Descriptors: Although real-valued descriptors demonstrate good performance and applicability, they expose challenges to both storage and matching. Popular real-valued descriptors (such as SIFT) and recent learning-based descriptors use 128 floating point numbers, or 512 bytes. While the storage requirement could be aggressively reduced by quantizing the floating point values [95, 100] or applying principal component analysis (PCA) [100, 35] to reduce the length of the descriptor,

comparing the shortened real-valued descriptors still requires computing a real-valued Euclidean distance. Efficient handcrafted binary descriptors ameliorate these problems by directly building a binary string using the input image patch. The metric used to evaluate the distance between two binary descriptors is the Hamming distance, which is the number of set bits after performing the XOR operation. Efficient computation of Hamming distance can exploit specialized instructions on supported hardware. Popular binary descriptors include BRIEF [15] and rotated BRIEF used by ORB [71], which typically rely on intensity comparisons using a predefined pattern. This significantly lowers the computation cost, although this also implies that these binary descriptors are less robust against drastic illumination changes. D-BRIEF [94] learns from corresponding image patches a set of discriminative projections, which can be further decomposed into fast small filters that can be computed using integral images. DOAP [28] demonstrates that a good binary descriptor can also be learned by optimizing the average precision on the retrieval task. Some recent real-valued descriptors can be trivially converted to binary descriptors by taking the sign of each dimension, and L2-Net has already shown promising results in generating binary descriptors by doing this. There exist more sophisticated ways to convert floating-point vectors into binary strings, such as LSH [23] or LDAHash [88].

While most existing works address *either* real-valued or binary descriptors, we show that our approach is applicable to *both*. To the best of our knowledge, among recent papers only DOAP shares this feature; but their formulation differs vastly from the triplet loss that many methods are based on, implying that their method is unlikely to be a drop-in replacement like ours.

Replacing Static Hard Margin: Instead of setting a hard margin, DeepDesc [83] back-propagates the hardest 1/8 of samples, which can be interpreted as a ***Dynamic Hard*** Margin. Person re-identification is a related topic that also uses the triplet

margin. Wang et al. [98] apply two separate hard margins for positive and negative pairs. The margins are dynamically adjusted using a hand-crafted function with cross-validated, but highly unintuitive, hyper-parameters ($\mu = 8$ and $\gamma = 2.1$). Their loss function still classifies training samples by making binary decisions, thus is yet another case of Dynamic Hard Margin. Hermans et al. [30] use the softplus function to mimic the soft margin. However, this function is fixed throughout the entire training session and we view it as an instance of ***Static Soft*** Margin. Moreover, softplus has an implicit scale hyper-parameter: $1/\beta \cdot \log(1+\exp(\beta x))$, where β controls smoothness and is 1.0 by default. Section 4.6.1, includes comparisons against these alternatives to demonstrate the effectiveness of our ***Dynamic Soft*** Margin.

4.3 Learning Local Descriptors

Our goal is to examine the effectiveness of the proposed Dynamic Soft Margin strategy, and we are motivated by the application of learning real-valued and binary local feature descriptors. As background, we first revisit the original triplet margin loss, together with the network architecture and hard-negative mining method used by state-of-the-art methods such as HardNet [59]. Next, we introduce a modified training procedure that can learn high-quality binary descriptors.

4.3.1 Real-Valued Descriptors

A Siamese network with two streams sharing the same deep architecture and weights is one of the natural choices for learning a descriptor. Most recent works adopt L2-Net [90] as the backbone network due to its good performance. We also use L2-Net in this work and only replace the loss function to show the effectiveness of our method. L2-Net, denoted as $\mathcal{F}(\cdot)$, takes an image patch x as input and produces a k-dimensional descriptor. Given two input image patches x and x', the distance between

them in descriptor space is written as $\mathcal{D}(\mathcal{F}(x), \mathcal{F}(x'))$, where $\mathcal{D}$ is a distance metric. For real-valued descriptors, the Euclidean distance is often used as the distance metric. While evaluating the Euclidean distances among a large number of descriptors can be expensive, the computational cost can be reduced if the feature vectors are unit-length (i.e., $\|\mathcal{F}(x)\| = 1$). Since the Euclidean distance between two unit vectors can be computed using the dot product:

$$\mathcal{D}_{\text{Euclidean}}\big(\mathcal{F}(x), \mathcal{F}(x')\big) = \sqrt{2 - 2\,\mathcal{F}(x)^T \mathcal{F}(x')}, \tag{4.1}$$

it becomes possible to compute the pair-wise distance matrix of a batch of descriptors with a single matrix multiplication. To leverage this nice property, L2-Net normalizes the network output into a unit-length descriptor. Given N pairs of matching image patches, where each pair corresponds to a unique 3D point in the physical world, HardNet [59] computes a pair-wise distance matrix from the descriptors output by the Siamese network. The diagonal elements in the distance matrix correspond to the distances between matching pairs. HardNet mines the hardest negative for each matching pair in its row and column from the non-diagonal elements in the distance matrix. Note that this mining is a different process from the weighting of triplets by our soft margin, as described below. For more details, we refer the readers to the original paper.

If we denote the distance between a matching pair as d_{pos} and the distance between the corresponding hardest non-matching pair as d_{neg}, HardNet trains using the standard triplet margin loss with margin $\mu = 1.0$:

$$\mathcal{L}_{\text{triplet}} = \max(0, \mu + d_{\text{pos}} - d_{\text{neg}}). \tag{4.2}$$

The triplet margin loss simply forces the network to learn to increase the difference between d_{neg} and d_{pos} until the condition $d_{\text{neg}} - d_{\text{pos}} > \mu$ is satisfied. If μ is set

sufficiently large that $d_{\text{neg}} - d_{\text{pos}} > \mu$ is never satisfied, $\frac{\partial \mathcal{L}_{\text{triplet}}}{\partial d_{\text{pos}}} = 1$ and $\frac{\partial \mathcal{L}_{\text{triplet}}}{\partial d_{\text{neg}}} = -1$, where μ is no longer relevant. Though $\mu = 1.0$ is shown to perform well for HardNet, it remains a question whether there exists a μ that works better.

4.3.2 Binary Descriptors

Real-valued descriptors are potentially costly to store and compute with, due to the fact that they are usually represented using 32-bit floating-point numbers. On the other hand, binary descriptors are both more compact to store and faster to compare (using Hamming distance), and hence are popular in real-time applications. Unfortunately, HardNet [59] *only* addresses real-valued descriptor learning. Below, we propose how to adapt it for binary descriptor learning, which is inherently non-differentiable.

At testing time, it is easy to convert a real-valued L2-Net output to a binary descriptor: we simply use the sign function to convert the output to a length-k vector of the values -1 and 1. This reduces the Hamming distance between two descriptors to a dot product:

$$\mathcal{D}_{\text{Hamming}}\big(\mathcal{F}(x), \mathcal{F}(x')\big) = \big(k - \mathcal{F}(x)^T \mathcal{F}(x')\big) \;/\; 2. \tag{4.3}$$

Therefore, as with real-valued descriptors, the pair-wise Hamming distance matrix of a list of binary descriptors can be computed using a single matrix multiplication.

The gradient of the sign function, however, is undefined at the origin and zero everywhere else. So, we need a differentiable proxy for training purposes. Therefore instead of normalizing the output of L2-Net, we use a hyperbolic tangent function (tanh) to compress the output of each element into the range $(-1, 1)$. During training, we use tanh whenever we need the Hamming distance to be differentiable, and use sign in other cases that require full binarization. For instance, the descriptors are

fully binarized before computing the distance matrix, because we need to mine the hard negatives as if the current batch were being tested.

Given the mined hard negatives and the distance metric, we can still use a triplet loss to learn the binary descriptor. Selecting an optimal margin becomes even more challenging this time: the difference between d_{neg} and d_{pos} can be as large as the maximum Hamming distance, which is the descriptor length k. It is difficult to determine a proper margin without running a few training/validation sessions. In fact, we have determined that for $k = 256$ the optimal margin is the unintuitive value $\mu = 32$.

4.4 Dynamic Soft Margin

In this section, we discuss how we replace the triplet margin loss used by state-of-the-art descriptor learning methods with our dynamic-weighted triplet loss. We first analyze the training behavior of the triplet margin loss and explain with an example why choosing a good margin is important for optimal performance. Then, we explain how our method eliminates the margin while improving performance.

4.4.1 Behavior of the Triplet Margin Loss

Let us first analyze how the triplet margin loss behaves when learning a real-valued descriptor. For a unit-length real-valued descriptor, the largest difference between d_{neg} and d_{pos} is 2.0 ($d_{\text{neg}} = 2.0$ and $d_{\text{pos}} = 0$). Setting the margin larger than or equal to 2.0 simply transforms the triplet margin loss into a basic triplet loss (i.e., without margin), where no triplet is affected by truncation. In practice, a margin smaller than 2.0 is expected to improve performance, since the "easiest" triplets, having larger $d_{\text{neg}} - d_{\text{pos}}$, are not involved in back-propagation. To better understand this behavior, we take a pretrained HardNet model (trained on UBC PhotoTourism -

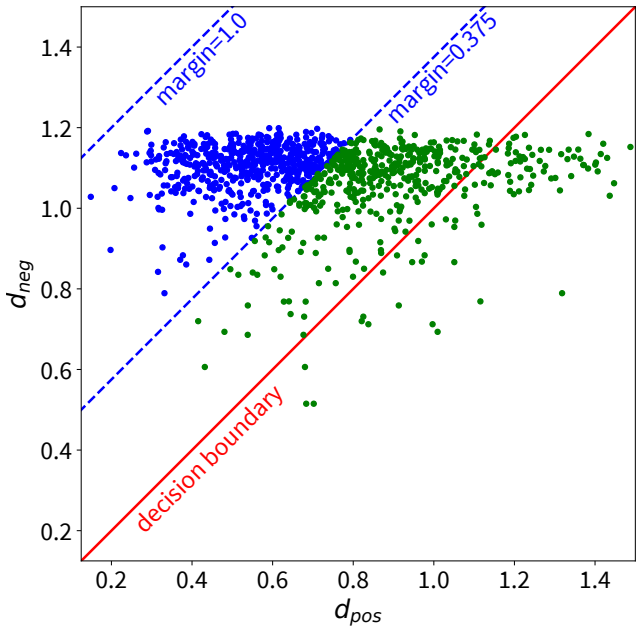


Figure 4.1: Scatter plot of $(d_{\text{pos}}, d_{\text{neg}})$ for triplets in one batch (1024 samples), using standard triplet margin loss. The red line is the decision boundary: d_{neg} is correctly greater than d_{pos} to its upper-left. The blue dotted lines are potential margins. The (optimal) margin of 0.375 separates points into two clusters (green and blue), where the blue cluster is considered "good enough" and is not used for back-propagation.

Liberty) and visualize all triplets in a sample batch of data by plotting $(d_{\text{pos}}, d_{\text{neg}})$ as scattered points in 2D (Figure 4.1).

In the case of a perfect descriptor, d_{pos} is expected to be smaller than d_{neg}. This corresponds to the region to the upper-left of the decision boundary shown in red. A straightforward approach to optimize $\mathcal{F}(\cdot)$ is to maximize the signed distance from the point $(d_{\text{pos}}, d_{\text{neg}})$ to the decision boundary, which is *equivalent* to minimizing the basic triplet loss: $\mathcal{L}_{\text{triplet}} = d_{\text{pos}} - d_{\text{neg}}$. The triplet margin loss sets a margin so that points that are sufficiently far on the correct side of the decision boundary (shown in blue, for a hypothetical margin $\mu = 0.375$) are excluded from optimization. By doing this, we force the network to focus on harder triplets (near to, or on the wrong side of, the decision boundary) without having the gradients influenced by the easy triplets. However, Figure 4.1 shows that a margin of 1.0, as recommended by the HardNet paper, would have almost no effect because all triplets are still within the margin.

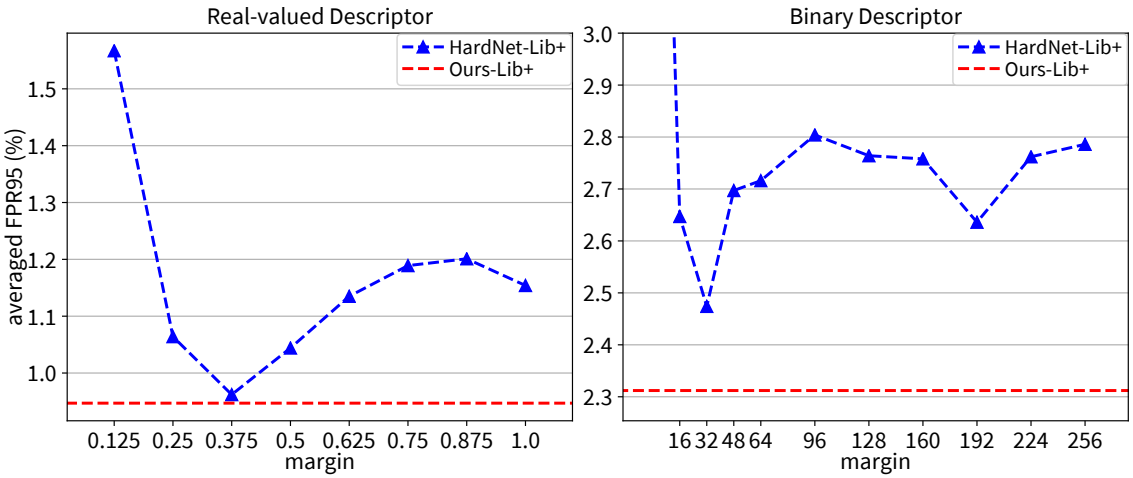


Figure 4.2: Varying the margin used by the triplet margin loss. The network is trained on the *Liberty* subset of UBC PhotoTourism and evaluated on the other two subsets. The left and right figures show the performance on the real-valued and binary descriptors, respectively. For our results, we keep all other configurations used by the triplet margin loss and only replace the loss function.

To investigate whether there is a "sweet-spot" for the margin, we vary it across a wide range and re-train the real-valued descriptor (always training on *Liberty* and evaluating the false positive rate at 95 percent recall – FPR95 – on the other two subsets of the UBC PhotoTourism dataset [101]). As demonstrated by the blue curve of Figure 4.2, left, there indeed exists a better choice of margin. As shown in Figure 4.1, $\mu = 0.375$ excludes a substantial number of easy triplets and lets the network focus on the harder cases. Figure 4.2, right, shows the corresponding graph for our new binary descriptor. Note that the shapes of the curves are different, and the optimal margin is problem-dependent. Of course, it is possible that even better margins could be found at greater computational cost by increasing the precision of the search.

We conclude that finding the best margin is a non-trivial job in practice and may require extensive validation. In the following section, we introduce our dynamic triplet weighting method, which avoids setting a hard threshold and weights the triplets in a mini-batch based on the training status of the network. Results produced by our approach are displayed as the red dotted lines in Figure 4.2.

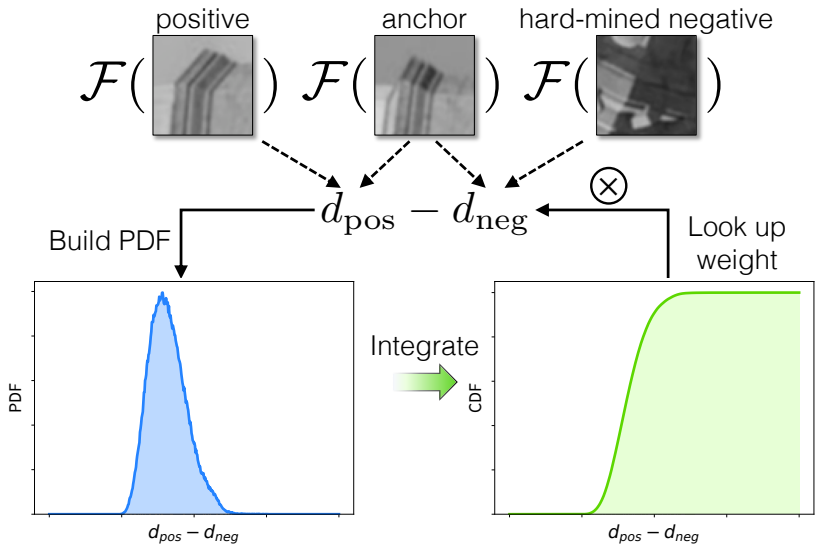


Figure 4.3: Our scheme for Dynamic Triplet Weighting. We compute $d_{\text{pos}} - d_{\text{neg}}$ for each hard-mined triplet, then build a moving histogram (PDF) of these values and integrate to obtain the CDF. The loss for each triplet is $d_{\text{pos}} - d_{\text{neg}}$, weighted by the corresponding value from the CDF.

4.4.2 Dynamic Triplet Weighting

Our approach shares the same motivation as the triplet margin loss, in that the "harder" triplets in a mini-batch are more useful for training. In other words, "easy" triplets should be suppressed in the loss function because the network's performance on these triplets is already likely to be saturated. The concept of emphasizing harder training examples is also the major reason why hard negative mining has become recognized as vital to good performance in recent learned descriptors.

The key observation of this work is that we can directly measure how hard a triplet is compared to other triplets in the same mini-batch, by seeing how its signed distance to the decision boundary ($d_{\text{pos}} - d_{\text{neg}}$) compares to the distribution of these distances. To measure this, we would like to know the Probability Distribution Function (PDF) of signed distances, which in practice we discretize into a histogram. To make the aggregated histogram more accurate, we compute $d_{\text{pos}} - d_{\text{neg}}$ for each triplet, and then linearly allocate it into two neighboring bins in the histogram. In our implementation, since a temporally stable PDF is preferred, we maintain it as an

exponentially-decaying moving histogram, similarly to other neural network modules that utilize moving averages (e.g., batch normalization [31]). An example of the PDF is shown in Figure 4.3, bottom left.

Given the distribution of difficulty in recent batches, the relative difficulty of a *particular* triplet corresponds to the fraction of triplets that have a lower $d_{\text{pos}} - d_{\text{neg}}$. This is just the integral of the PDF, or the Cumulative Distribution Function (CDF), shown in Figure 4.3, bottom right. The hardest triplet in a mini-batch results in a CDF of 1.0, while the easiest triplet corresponds to ≈ 0. More generally, a triplet with a CDF value of $k\%$ means that it is empirically "harder" than $k\%$ of triplets within recent batches.

Because these CDF values have an intuitive interpretation as difficulty, we use them directly as weights. Given a mini-batch of size N, we define our weighted triplet loss (without a hard margin) as:

$$\mathcal{L} = \frac{1}{N} \sum_i w_i \cdot (d^i_{\text{pos}} - d^i_{\text{neg}}), \tag{4.4}$$

$$w_i = \text{CDF}(d^i_{\text{pos}} - d^i_{\text{neg}}). \tag{4.5}$$

This loss function automatically rejects "easy" triplets by assigning them low weights. One may wonder how the loss function behaves when the variance of $d_{\text{pos}} - d_{\text{neg}}$ is very small, so that the CDF is close to a step function. In fact, when such a case happens, the loss function weights all triplets nearly equally and the optimization continues. This is not always possible for the original triplet margin loss because the optimization would stop when every triplet satisfies $d_{\text{neg}} - d_{\text{pos}} > \mu$ (though this scenario rarely happens in practice). The red lines in Figure 4.2 show that our method consistently leads to better performance than the triplet margin loss on both the real-valued and binary descriptors.

4.5 Experiments

We have experimented with three benchmarks: UBC PhotoTourism [101], HPatches [9], and the Oxford Affine benchmark [57]. UBC PhotoTourism is a classic patch-based dataset that is mainly evaluated on the patch verification task, which can be quickly computed and is effective for preliminary analysis of the descriptor performance. The patch verification task is often not sufficient for estimating the performance of a descriptor in practical applications where patch retrieval is a more important task. HPatches is a more comprehensive benchmark that contains a much larger collection of image patches and evaluates a descriptor on three different tasks: patch verification, image matching, and patch retrieval. The Oxford Affine benchmark contains image sequences with different types of distortion, which is useful for understanding the robustness of a descriptor when the input images are less than ideal.

4.5.1 Implementation

We adopt a training configuration as similar as possible to that used by previous work, to ensure that our new loss function is the major factor in the final results. For training, we use the UBC PhotoTourism dataset [101]. Each of its three subsets, known as *Liberty*, *Yosemite*, and *Notre Dame*, consists of more than 400k image patches, cropped to 64×64 and re-oriented using Difference-of-Gaussians (DoG) keypoints [49]. We train one model using each subset and test on the other two subsets. We downsample each patch to a 32×32 input, which is required by L2-Net. Each patch is then normalized by subtracting the mean pixel value and dividing by the standard deviation. Online data augmentation is achieved by random flipping and rotating the patch by 90, 180 or 270 degrees. The UBC PhotoTourism dataset assigns each patch with its 3D point ID, which is used to identify matching image patches. Each 3D point ID is associated with a list of patches that are assumed to

be matching. To form a mini-batch of size N for training, we randomly select N 3D points without replacement and select two patches for each chosen 3D point.

We use Stochastic Gradient Descent (SGD), with momentum and weight decay equal to 0.9 and 10^{-4}, respectively, to optimize the network. Inspired by HardNet and DOAP, the network is trained for 50k iterations, with the learning rate linearly decaying from 0.1 to 0. The batch size is set to 1024 for all experiments to match the publicly available implementations of HardNet and DOAP. To facilitate future research, we package our implementation as a standalone PyTorch [66] module that could be easily deployed in other contexts.

4.5.2 UBC PhotoTourism

Each of the UBC PhotoTourism subsets includes a test split containing 100k pairs of image patches, with half of them being true matches and the rest being false matches. We adopt the commonly used false positive rate at 95% true positive recall (FPR95) to evaluate how well the proposed descriptor classifies patch pairs. We compare with a collection of existing real-valued descriptors including both handcrafted (SIFT [49] and root-SIFT [4]) and learned (DeepDesc [83], TFeat [10], GOR [114] PCW [61], L2-Net [90], HardNet [59], DOAP [28]). GeoDesc [51] is not evaluated because it is trained on a custom dataset. We also compare against existing binary descriptors including ORB [71], BinBoost [93], LDAHash [88], DeepBit [47], L2-Net [90], and DOAP [28]. The results are shown in Table 4.1. Our approach outperforms all existing methods under the same configuration. DOAP-ST+ uses a larger input (42×42) to augment DOAP+ with the Spatial Transformer [33], which noticeably improves the performance by correcting geometric noise. Note that our method also surpasses DOAP-ST+ in most cases even without the Spatial Transformer. Compared to HardNet, our method automatically produces better performance that otherwise

Table 4.1: Evaluation on the UBC PhotoTourism dataset, demonstrating that both real-valued and binary descriptors trained using our method outperform the state of the art. Second column shows the descriptor dimensionality. Numbers shown are FPR95(%) – lower is better. "+" and "*" denote training with data augmentation and anchor swapping [10]. DOAP-ST+ represents the DOAP descriptor with a Spatial Transformer [33] to compensate for geometric noise.

Descriptor	Train →	Notredame	Yosemite	Liberty	Yosemite	Liberty	Notredame	Mean
	Test →	Liberty		Notredame		Yosemite		
Real-valued Descriptors								
SIFT [49]	128	29.84		22.53		27.29		26.55
DeepDesc [83]	128	10.9		4.40		5.69		7.0
TFeat-M* [10]	128	7.39	10.31	3.06	3.80	8.06	7.24	6.64
TL+GOR* [114]	128	4.80	6.45	1.95	2.38	5.40	5.15	4.36
PCW [61]	128	7.44	9.84	3.48	3.54	5.02	6.56	5.98
L2-Net+ [90]	128	2.36	4.70	0.72	1.29	2.57	1.71	2.23
CS-L2-Net+ [90]	256	1.71	3.87	0.56	1.09	2.07	1.30	1.76
HardNet+ [59]	128	1.49	2.51	0.53	0.78	1.96	1.84	1.52
DOAP+ [28]	128	1.54	2.62	0.43	0.87	2.00	**1.21**	1.45
DOAP-ST+ [28]	128	1.47	2.29	**0.39**	0.78	1.98	1.35	1.38
Ours+	128	**1.21**	**2.01**	**0.39**	**0.68**	**1.51**	1.29	**1.18**
Binary Descriptors								
ORB [71]	256	59.15		54.57		54.96		56.23
BinBoost [93]	64	20.49	21.67	16.90	14.54	22.88	18.97	19.24
LDAHash [88]	128	49.66		51.58		52.95		51.40
DeepBit [47]	256	32.06	34.41	26.66	29.60	57.61	63.68	40.67
L2-Net+ [90]	128	7.44	10.29	3.81	4.31	8.81	7.45	7.02
CS-L2-Net+ [90]	256	4.01	6.65	1.90	2.51	5.61	4.04	4.12
DOAP+ [28]	256	3.18	4.32	1.04	1.57	4.10	3.87	3.01
DOAP-ST+ [28]	256	2.87	4.17	0.96	1.76	3.93	3.64	2.89
Ours+	256	**2.70**	**4.01**	**0.93**	**1.44**	**3.69**	**2.98**	**2.63**

would have required fine-tuning the margin, or even manually adjusting the margin at different stages of training.

4.5.3 HPatches

The recently-introduced HPatches benchmark of Balntas et al. [9] evaluates descriptors in a more sophisticated setting. Different amounts of geometric noise are intro-

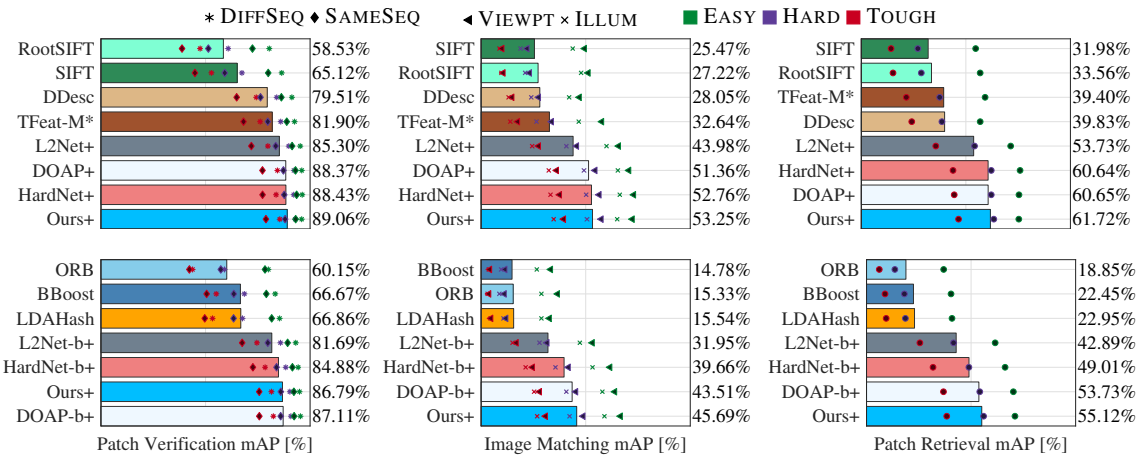


Figure 4.4: Evaluation on the HPatches dataset [9]. The evaluation is carried out on the "full" split of HPatches. The patch retrieval task is evaluated with the maximum amount of distractors (same setting used in the original HPatches paper). Top row: real-valued descriptor comparison. Bottom row: binary descriptor comparison. While both HardNet and DOAP perform well in easy cases, our descriptor is more robust in tough cases, leading to state-of-the-art performance overall.

duced into the test image patches, which are then categorized as "Easy", "Hard" or "Tough". HPatches evaluates a descriptor on three different tasks: patch verification, image matching, and patch retrieval. For a more detailed description of the tasks, we refer the readers to their paper.

Figure 4.4 compares descriptors trained with the proposed method and top-performing real-valued and binary descriptors. As is common practice, learned descriptors are evaluated using a model trained on the *Liberty* subset of the UBC PhotoTourism dataset, with data augmentation. For all descriptors, we do not apply the ZCA normalization that is originally used in HPatches. HardNet does not come with a binary version and we simply take the sign to obtain HardNet-b+. It is not surprising to see that both the real-valued and binary descriptors learned using our loss function perform well on the patch verification task, which is consistent with our observation on the UBC PhotoTourism dataset. On the more challenging image matching and patch retrieval tasks that require the descriptor to be more distinctive, our descriptors outperform all existing methods.

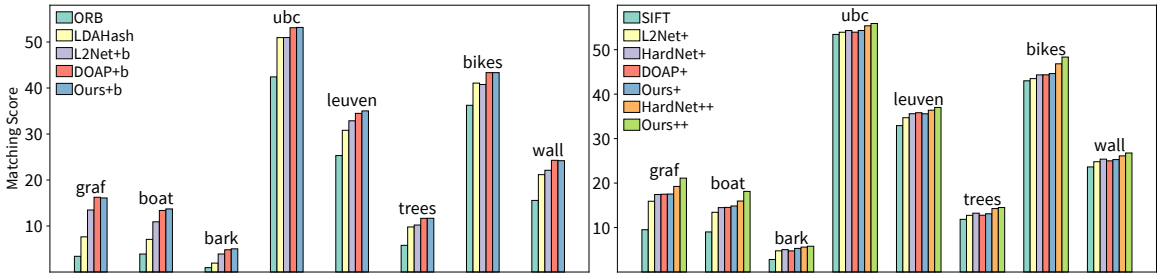


Figure 4.5: Evaluation on the Oxford Affine dataset, for binary (left) and real-valued (right) descriptors. All are trained on the UBC *Liberty* subset with data augmentation, except the models suffixed with "++", which are trained on the union of UBC PhotoTourism and HPatches.

4.5.4 Image Matching on the Oxford Dataset

In real image matching scenarios, images may undergo diverse distortions including geometric transformations, blurring, illumination changes, and JPEG compression. In order to verify whether the descriptors learned with our method are vulnerable to a particular type of distortion, we further evaluate the image matching performance using the Oxford Affine Dataset [57], which contains all the above-mentioned transformations. In this dataset, homography matrices are provided to help verify correspondences. We choose the Harris-Affine detector [55] to extract keypoints from the images and crop image patches using a magnification factor of 6. We strictly follow the public evaluation protocol [57]. The matching scores are reported in Figure 4.5. The result shows that our descriptors can withstand various type of distortions presented in this dataset and achieve state-of-the-art results. Also note that our descriptor trained with the union of UBC PhotoTourism and HPatches outperforms HardNet trained with the same data.

Table 4.2: Comparing existing alternatives to *Static Hard Margin* with our *Dynamic Soft Margin*.

Method	Dynamic	Soft	UBC	HPatches (mAP %)		
			FPR95(%)	Verification	Matching	Retrieval
Real-valued Descriptors						
softplus [30]	✗	✓	1.20	89.00	52.84	60.72
Wang [98]	✓	✗	1.18	88.88	52.63	60.36
Hardest 1/8 [83]	✓	✗	2.19	85.82	46.74	56.51
Ours	✓	✓	**0.95**	**89.06**	**53.25**	**61.72**
Binary Descriptors						
softplus [30]	✗	✓	2.73	86.35	45.45	54.23
Wang [98]	✓	✗	2.76	86.37	45.60	54.19
Hardest 1/4 [83]	✓	✗	3.09	85.58	43.24	52.97
Ours	✓	✓	**2.31**	**86.79**	**45.69**	**55.12**

4.6 Ablation Studies

To be consistent, we again use the models trained on the *Liberty* subset of UBC PhotoTourism in the following experiments. FPR95 is evaluated on the other two subsets.

4.6.1 Existing Alternatives to Static Hard Margin

In Section 4.1, we have discussed three previous attempts to replace the sub-optimal static hard margin. Recall that none of them is both *dynamic* and *soft* like ours. Since these baselines are either originally proposed in a different context (e.g., person re-identification [98, 30]) or with a different learning scheme (e.g., contrastive loss with two-stage training [83]), we re-implement and adapt these methods into our pipeline to ensure a fair comparison. Table 4.2 shows the results on both UBC PhotoTourism and HPatches. Our dynamic and soft strategy outperforms all baseline methods. Training the binary descriptor with the hardest 1/8 of triplets [83] did not converge in our experiment, most likely because 1/8 is too selective; we therefore use 1/4 instead.

Table 4.3: Comparing different ways to construct the PDF.

PDF built from	UBC	HPatches (mAP %)		
	FPR95 (%)	Verification	Matching	Retrieval
Real-valued Descriptors				
d_{pos}	0.98	**89.10**	52.93	61.37
d_{neg}	1.20	88.55	53.15	60.36
Gaussian	1.07	89.05	52.95	61.38
$d_{\text{pos}} - d_{\text{neg}}$	**0.95**	89.06	**53.25**	**61.72**
Binary Descriptors				
d_{pos}	**2.30**	**86.93**	**45.81**	**55.19**
d_{neg}	3.02	85.58	45.21	53.27
Gaussian	2.33	86.72	45.68	54.86
$d_{\text{pos}} - d_{\text{neg}}$	2.31	86.79	45.69	55.12

4.6.2 Different Ways to Construct the PDF

In the approach described above, we weighted samples based on a moving PDF of $d_{\text{pos}} - d_{\text{neg}}$, mainly because it is well-correlated with how "hard" a triplet is. In a more general context, we believe that any variable that effectively reflects the "hardness" of a triplet can be used to build the PDF. For example, we observe from Figure 4.1 that the variation of the visualized points mainly happens along the d_{pos} axis, whereas the variation along the d_{neg} axis is smaller. This implies that instead of maintaining the moving PDF of $d_{\text{pos}} - d_{\text{neg}}$, using a PDF of d_{pos} could also work well, while we would expect a PDF of d_{neg} to be less effective. We also observe that Figure 4.3 suggests that the PDF is approximately Gaussian, and might be summarized by its mean and variance. This indicates that we can potentially use a parametric PDF to replace the histogram representation that we currently use, and save memory and computation. We have therefore explored simply maintaining a running mean and variance of $d_{\text{pos}} - d_{\text{neg}}$, and then weighting triplets based on the *analytic* Gaussian CDF, which may be computed in terms of the standard error function (erf). The comparisons are shown in Table 4.3, from which we observe that building the PDF from d_{neg} indeed leads to the worst performance. As expected, d_{pos} is as good an

indicator as $d_{\text{pos}} - d_{\text{neg}}$. Approximating the PDF with a simple Gaussian distribution is also feasible, but with a small sacrifice in performance, suggesting that the actual distribution is non-Gaussian.

4.7 Conclusion

In this work, we observe that the triplet loss previously used in descriptor learning requires manually setting the margin, which usually leads to sub-optimal results. Setting a hard margin is equivalent to making a binary decision of whether a triplet should be excluded from optimization. Also, a constant margin cannot account for the continuous improvement in performance of the network during training.

We instead propose a "dynamic soft margin" strategy that automatically assigns lower weights to datapoints that are too "easy" for improving the network, at each stage of training. The key insight is that the relative "hardness" of a triplet can be inferred from the moving Probability Distribution Function (PDF) of the difference of distances. We demonstrate that using the CDF computed from this PDF as a weighting function is an effective way to make the network focus on harder triplets. Through extensive experiments we show that our method can be applied to both real-valued and binary descriptor learning, and lead to state-of-the-art performance. Future work includes generalizing the proposed method to other similar domains where empirical margins are being used.

Chapter 5

Conclusion and Future Work

A central problem of computer vision is to establish correspondences across a collection of images. This challenge is becoming increasingly important since a large quantity of images are made available every day. Many computer vision applications including image-based reconstruction, panorama stitching, and image retrieval are traditionally based on local features.

In this thesis, we design a new texture-based high-precision localization system named Micro-GPS, which utilizes handcrafted local features. The Micro-GPS system is built based on the observation that seemingly random and homogeneous real-world ground textures contain unique visual landmarks, which could be leveraged to achieve fine global localization. To further address the limitation that a handcrafted keypoint detector cannot work reliably across all types of textures, we propose a method to learn the keypoint detector on a per-texture basis, which prominently improve the robustness of Micro-GPS. The keypoint detector is learned with a novel peakedness loss, which makes keypoint localization more precise, thus highly repeatable. Besides keypoint detection, the quality of feature descriptors influences the matching of local features, and we propose an easy-to-use loss function that can be applied to descriptor learning and achieve state-of-the-art matching and retrieval performance.

As indicated by this thesis, the topic of local feature detection and matching is not yet fully explored. Below we summarize a few potential future research directions.

Handcrafted vs. Data-driven

Local features are conventionally handcrafted based on a general understanding of images containing common scenes. As cameras have become extremely ubiquitous in many fields, the availability of images is no longer just limited to those related to natural scenes, paintings, pictures of common objects, etc. There are increasingly more domain-specific images that look drastically different from common images, and oftentimes existing handcrafted features cannot work properly out of the box. A variety of texture images we have demonstrated in Chapter 2 are exactly examples of domain-specific images. We have a detailed analysis in Chapter 3 showing that existing keypoint detectors perform poorly on a collection of challenging textures.

Given the broad range of applications that have started to embrace computer vision, designing an effective local feature purely based on the empirical analysis of each kind of image is not scalable. In the long run, we believe that data-driven methods will become the mainstream. This is implied by the fact that many real-world applications come with specific usage scenarios, or even use particular types of camera. This prior knowledge makes data-driven approaches more advantageous; with a sufficient amount of training data collected beforehand, learning a robust local feature for a particular application is feasible.

Nevertheless, handcrafted methods have the advantage of explicitly enforcing various types of invariance. In contrast, data-driven approaches often acquire invariance through data augmentation. For instance, in Chapter 3, we randomly rotate the training patches to let the neural network learn a rotation-invariant keypoint detector. A handcrafted baseline such as SIFT [49], however, uses the Difference-of-Gaussian filter, which is naturally (and guaranteed to be) rotation-invariant. Therefore, we believe

that a more promising direction of developing data-driven methods is to incorporate domain knowledge we have accumulated for decades in designing handcrafted features. In a more general context where we do not have any prior knowledge, handcrafted methods could stay popular because of the simple and fast deployment.

Scale-Invariant Descriptor Learning: Recent learning based descriptors are constrained by the input resolution. The commonly used 32×32 resolution is determined by the network architecture. Note that this can be insufficient to retain fine details when the detected feature scale is large, but the cropped patch still has to be 32×32. While one could alter the network architecture to accommodate a larger resolution, processing high resolution patches would lower the overall efficiency and increase the memory footprint. As a result, an input image patch with a different resolution needs to be re-sampled to this constant size before network inference. In contrast, many handcrafted descriptors are able to take arbitrarily sized input. For example, SIFT [49] uses a 4×4 grid and aggregates the gradients into 8 bins within each cell, which always results in a 128-dimensional feature descriptor. BRIEF [15] performs intensity comparison, which does not depend on the resolution either. The ability to fully leverage the original image patch without re-sampling is the major reason why we still use a handcrafted descriptor like SIFT in Micro-GPS. A future direction could be adapting the network architecture to take image patches with different resolutions as input, and produce a fixed size output. One possible way to achieve this functionality is via Spatial Pyramid Pooling (SPP) [27], which is originally designed for object detection. Meanwhile, the network needs to be *scale-invariant*, which means that resizing an image patch should not affect the output descriptor. This property can possibly be obtained by randomly resizing the image patches used for training.

Synergy Between the Detector and Descriptor

There remain many more aspects to be improved in keypoint detection. The most immediate direction could be strengthening the interactions between the keypoint detector and feature descriptor. Currently in most cases, the detector and descriptor work as independent components, but the performance of feature matching depends on *both*. First, using the repeatability as the key criterion to evaluate keypoint detectors is not sufficient. Imagine that we have a "perfect" keypoint detector that achieves a perfect repeatability (i.e., 100%), but keypoints are only (magically) localized in un-textured regions. Matching descriptors computed from these un-textured regions could be extremely ambiguous, and most likely we would not obtain useful correspondences. Another drawback to existing keypoint detectors is that they do not take the global context into consideration. This can be better illustrated by Figure 5.1, where each window could be treated as a large blob and triggers the keypoint detector. These windows, however, look almost identical and it is reasonable to expect the same output when applying any feature descriptor, which again makes descriptor matching highly ambiguous. Therefore a future direction could be making the detector find keypoints that are globally unique within the input image. This would require feedback from the feature descriptor; keypoints with similar descriptors should be less preferred by the detector.

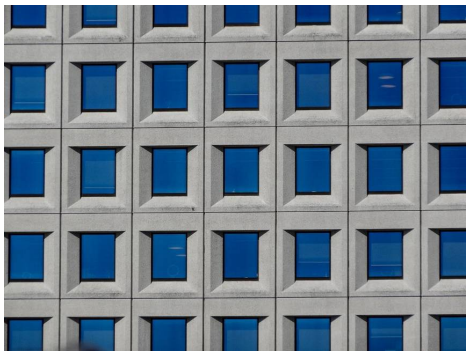


Figure 5.1: A typical building with nearly identical windows.

As for descriptor learning, one major bottleneck comes from the image patches used for training. In Chapter 3, we obtain the training patches by randomly cropping from the overlapping regions between images. In most other cases, however, image

patches are cropped around the keypoints produced by a particular detector. For example, patches in the UBC PhotoTourism dataset [101] correspond to SIFT [49] keypoints. It is inevitable that a descriptor learned on this dataset works best with SIFT. This implies that the learning of detector and descriptor need to be performed simultaneously.

The major practical difficulty of jointly learning the detector and descriptor is to obtain the training data. For such end-to-end training, we would need pairs of images with overlapping views, as well as dense depth maps for establishing ground-truth correspondences. This is because the detector could make any location in the image as a keypoint, which needs to be paired with the corresponding location in the other image, for descriptor learning. Ono et al. have already demonstrate that training the entire local feature pipeline end-to-end is feasible [64]. A sophisticated public dataset could further benefit the research community.

Assuming that such a dataset is available, there are several directions one can pursue. The most straightforward way to combine the detector and descriptor is to mimic a traditional pipeline; image patches can be cropped and re-oriented using the differentiable Spatial Transformer [33], around the keypoints produced by the detector, and used for descriptor learning. This direct analogy, however, neglects the fact the low-level features maps leveraged by the detector and descriptor could be *shared*. Thus, one could crop *feature* patches instead of image patches to save computation and memory. Note that if the feature maps are by nature rotation-invariant (i.e., not through data augmentation), the cropped feature patches could be directly treated as descriptors, which is an even more efficient option.

Improved Micro-GPS

With an improved keypoint detector and feature descriptor in mind, we envision a more robust Micro-GPS system. By utilizing a keypoint detector that takes the

context into consideration and only looks for globally unique keypoints, it is feasible to store just a few most distinctive features for each database image. This not only further reduces the overall storage on top of the optimizations we have done, but also makes the global feature matching less ambiguous. A learned descriptor that can be directly applied to an arbitrarily-sized image patch can avoid downsampling and fully utilize the fine details in the original image. This provides the learned descriptor an opportunity to outperform the currently used SIFT descriptor.

Appendix A

Micro-GPS Datasets

The figures below show overviews and close-ups of the indoor and outdoor datasets that we have gathered. In each figure, we show:

- Top left: An overview of the scene where the images are captured.
- Bottom left: An example of one captured image.
- Right: A portion of the reconstructed map, with the full stitched map shown in the inset at bottom right.

Figure A.1: Indoor tiles. The map includes 1296 images covering approximately 1.5×8.5 m.

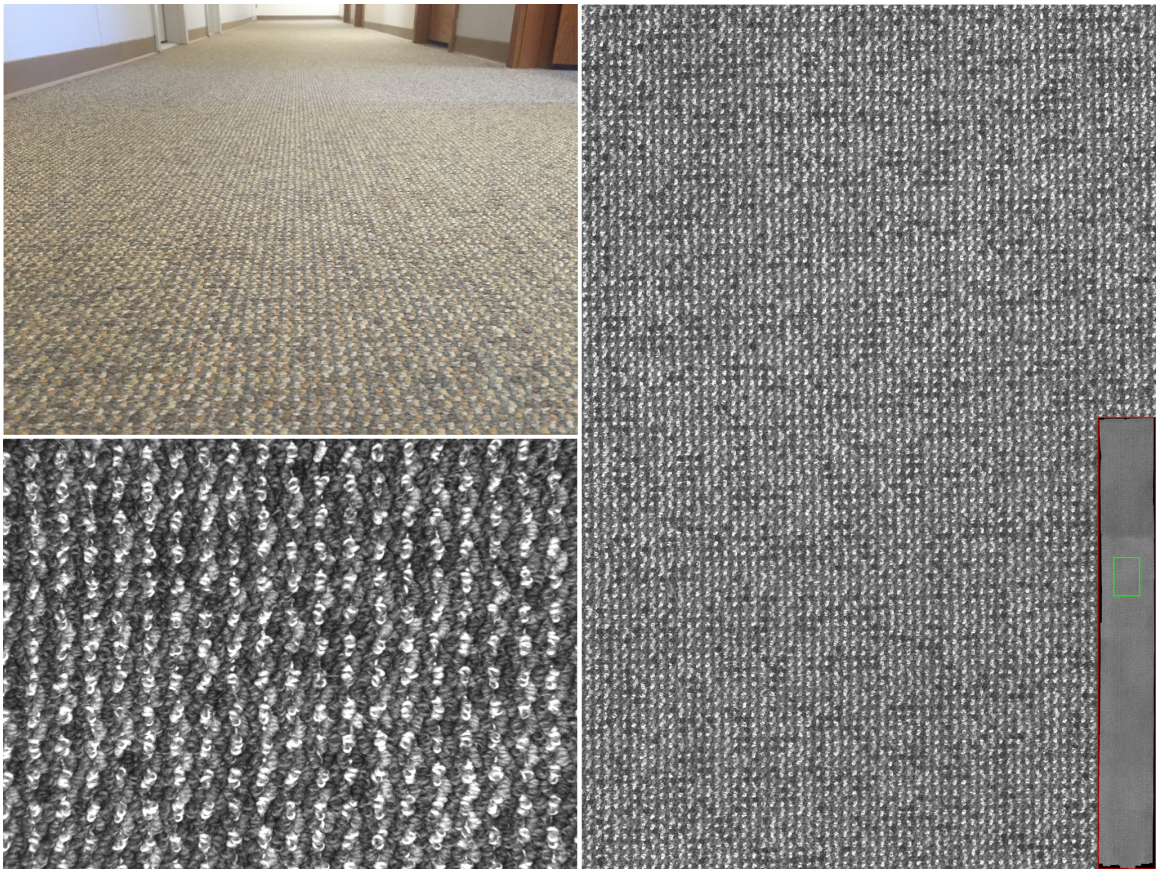


Figure A.2: Indoor carpet. This dataset contains 2014 images covering approximately 1.5×11.8 m.

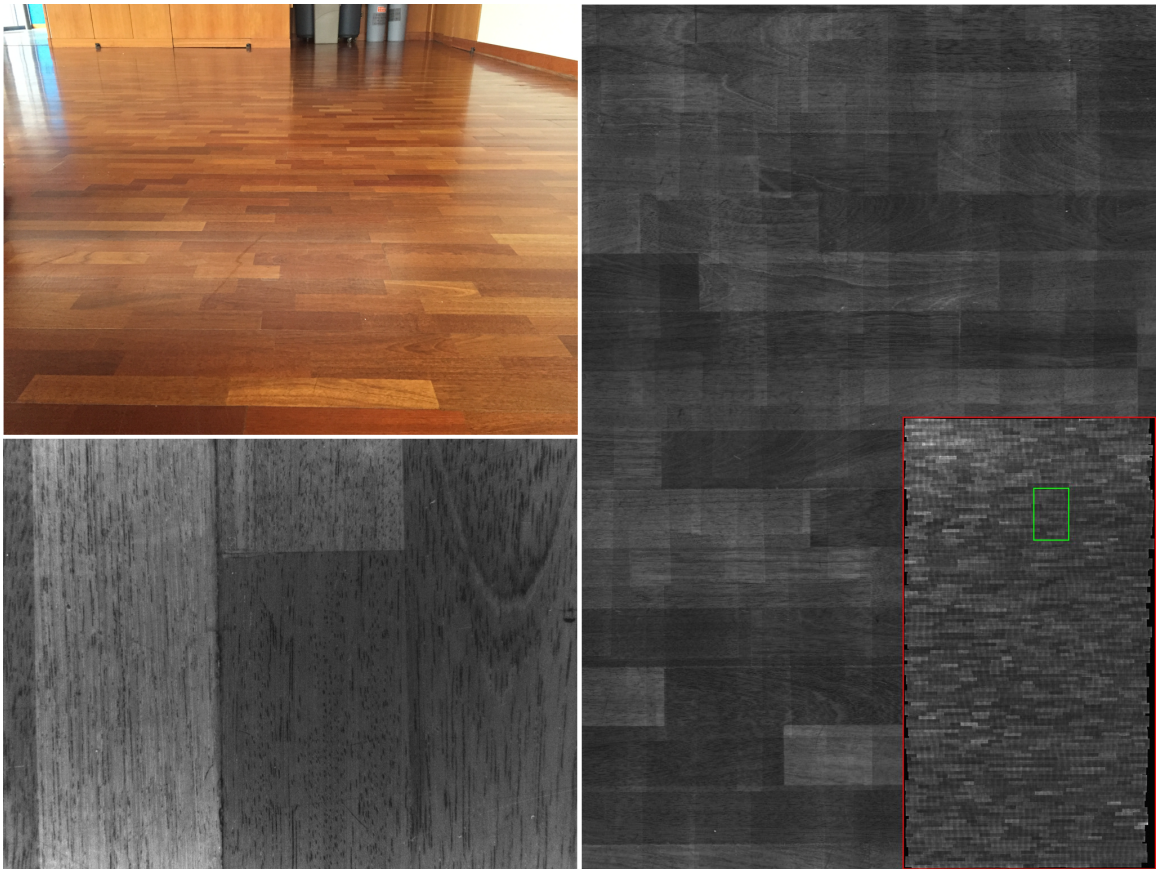


Figure A.3: Indoor wood floor. This dataset contains 3826 images covering approximately 8.7×4.8 m.

Figure A.4: Outdoor granite tiles. This dataset contains 1229 images covering approximately 8.6×3.2 m.

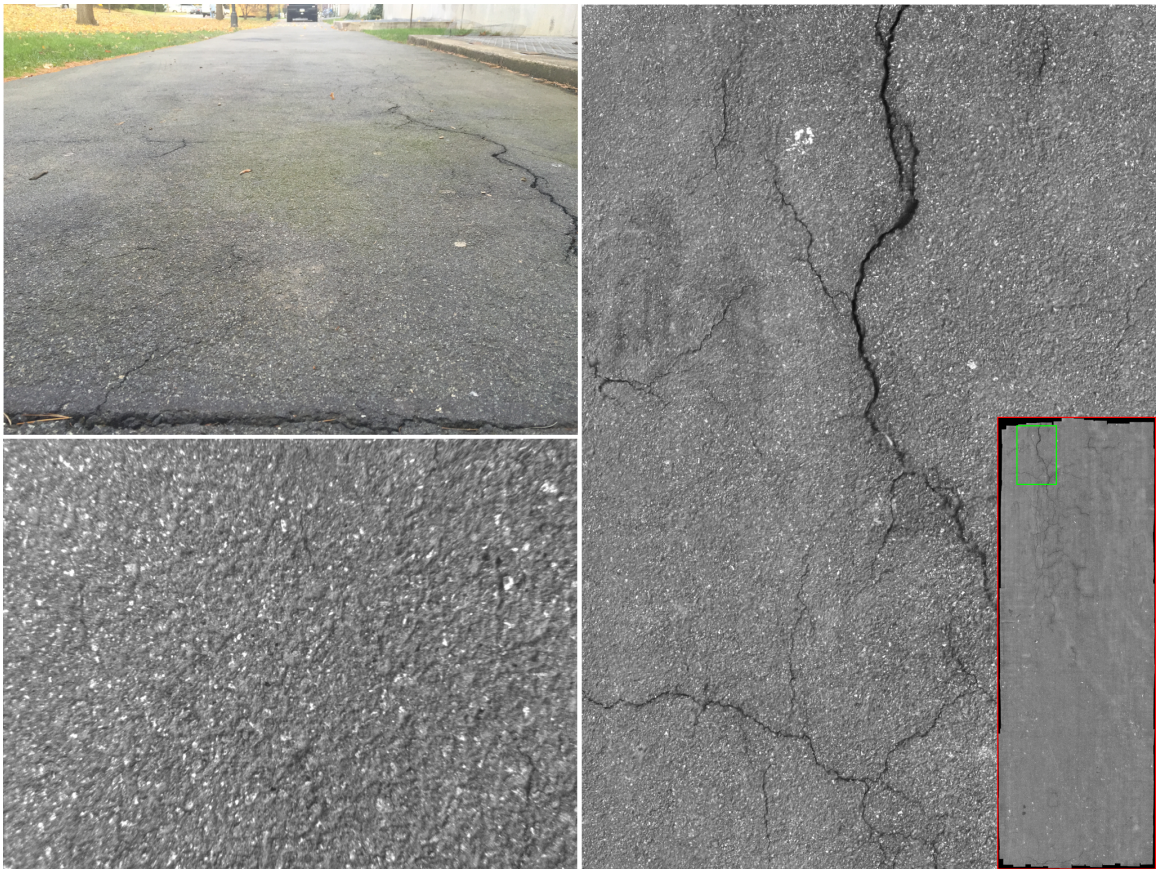


Figure A.5: Outdoor asphalt with fine aggregate, captured under ambient illumination (sunlight). This dataset contains 2215 images covering approximately 2.6×7.6 m.

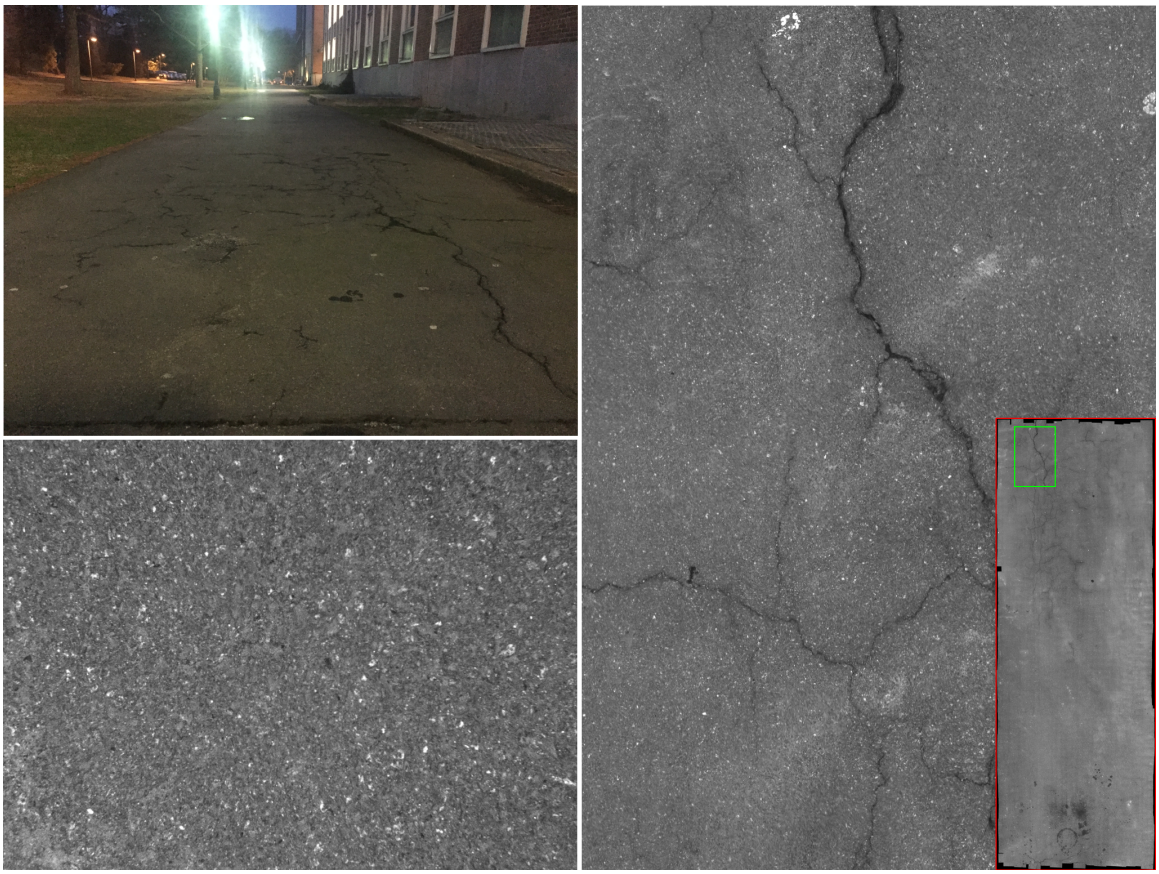


Figure A.6: Outdoor asphalt with fine aggregate captured at night under controlled LED illumination. This dataset contains 2118 images covering approximately the same area as above.

Figure A.7: Outdoor asphalt with coarse aggregate. This dataset contains 2061 images and the dimension is approximately 2.0×10.6 m.

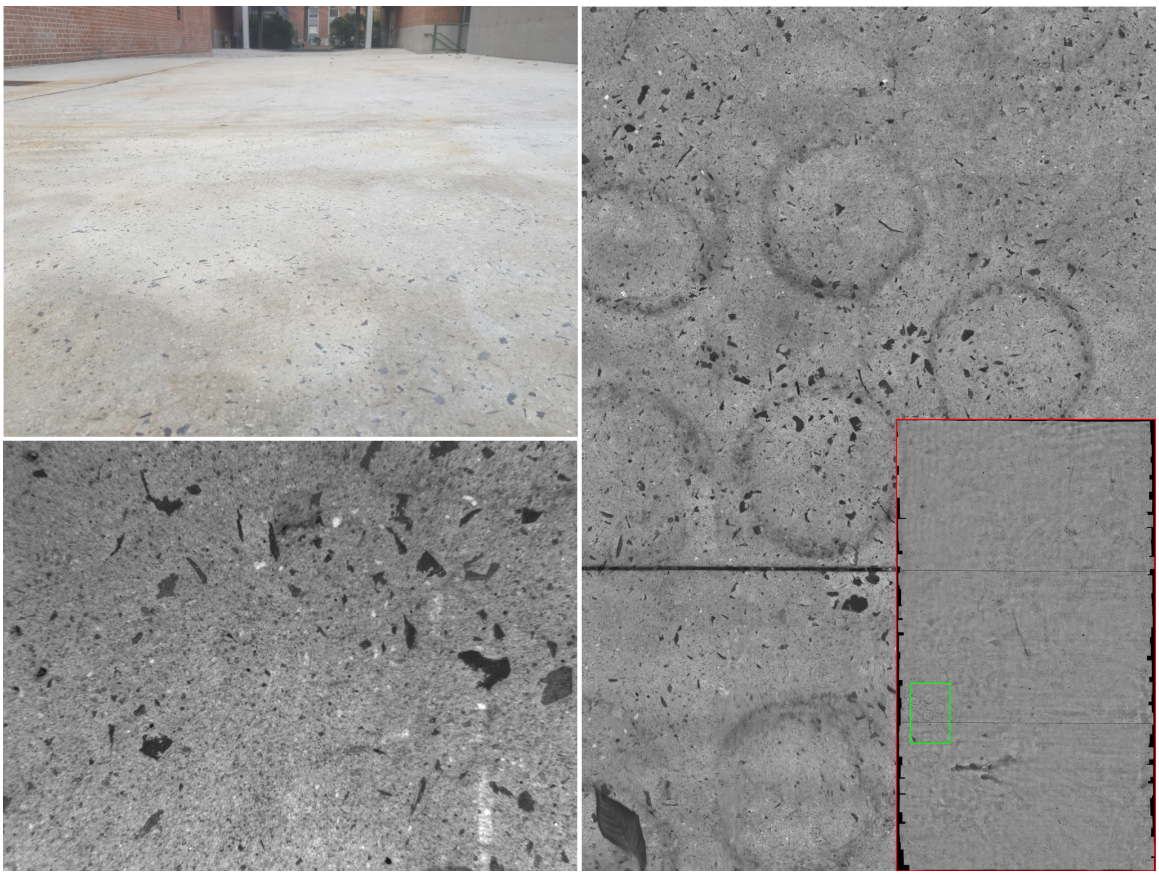


Figure A.8: Outdoor concrete. This dataset contains 3316 images and the dimension is approximately 7.6×4.3 m.

Bibliography

[1] Henrik Aanæs, Anders Lindbjerg Dahl, and Kim Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97(1):18–35, 2012.

[2] Sameer Agarwal, Keir Mierle, et al. Ceres solver. `http://ceres-solver.org/`, 2013.

[3] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M Seitz, and Richard Szeliski. Building rome in a day. In *Proc. ICCV*, pages 72–79. IEEE, 2009.

[4] Relja Arandjelović and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *Proc. CVPR*, pages 2911–2918. IEEE, 2012.

[5] Clemens Arth, Manfred Klopschitz, Gerhard Reitmayr, and Dieter Schmalstieg. Real-time self-localization from panoramic images on mobile devices. In *Proc. ISMAR*, pages 37–46, 2011.

[6] Yannis Avrithis and Giorgos Tolias. Hough pyramid matching: Speeded-up geometry re-ranking for large scale image retrieval. *International Journal of Computer Vision*, 107(1):1–19, March 2014.

[7] Georges Baatz, Kevin Köser, David Chen, Radek Grzeszczuk, and Marc Pollefeys. Handling urban location recognition as a 2d homothetic problem. In *Proc. ECCV*, pages 266–279, 2010.

[8] Soonmin Bae, Aseem Agarwala, and Frédo Durand. Computational rephotography. *ACM Trans. Graphics*, 29(3):24:1–24:15, July 2010.

[9] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *Proc. CVPR*, volume 4, page 6, 2017.

[10] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC*, volume 1, page 3, 2016.

[11] Vassileios Balntas, Lilian Tang, and Krystian Mikolajczyk. BOLD – binary online learned descriptor for efficient image matching. In *Proc. CVPR*, 2015.

[12] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Proc. ECCV*, pages 404–417, 2006.

[13] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: speeded up robust features. *Proc. ECCV*, pages 404–417, 2006.

[14] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, August 2007.

[15] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proc. ECCV*, pages 778–792. Springer, 2010.

[16] Song Cao and Noah Snavely. Minimal scene descriptions from structure from motion models. In *Proc. CVPR*, pages 461–468, 2014.

[17] William Clarkson, Tim Weyrich, Adam Finkelstein, Nadia Heninger, J Alex Halderman, and Edward W Felten. Fingerprinting blank paper using commodity scanners. In *Proc. IEEE Symposium on Security and Privacy*, pages 301–314, 2009.

[18] Matthew Cornick, Jeffrey Koechling, Byron Stanley, and Beijia Zhang. Localizing ground penetrating radar: a step toward robust autonomous ground vehicle localization. *Journal of Field Robotics*, 33(1):82–102, 2016.

[19] Kristin Dana, Bram van Ginneken, Shree Nayar, and Jan Koenderink. Reflectance and texture of real-world surfaces. *ACM Trans. Graphics*, 18(1):1–34, January 1999.

[20] Alexei Efros and Thomas Leung. Texture synthesis by non-parametric sampling. In *Proc. ICCV*, pages 1033–1038, 1999.

[21] Hui Fang, Ming Yang, Ruqing Yang, and Chunxiang Wang. Ground-texture-based localization for intelligent vehicles. *IEEE Trans. Intelligent Transportation Systems*, 10(3):463–468, September 2009.

[22] Wolfgang Förstner, Timo Dickscheid, and Falko Schindler. Detecting interpretable and accurate scale-invariant keypoints. In *Proc. ICCV*, pages 2256–2263. IEEE, 2009.

[23] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[24] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. MatchNet: unifying feature and metric learning for patch-based matching. In *Proc. CVPR*, pages 3279–3286, 2015.

[25] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proc. Alvey Vision Conference*, pages 147–151, 1988.

[26] Wilfried Hartmann, Michal Havlena, and Kaspar Schindler. Predicting matchability. In *Proc. CVPR*, pages 9–16, 2014.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. PAMI*, 37(9):1904–1916, 2015.

[28] Kun He, Yan Lu, and Stan Sclaroff. Local descriptors optimized for average precision. In *Proc. CVPR*, pages 596–605, 2018.

[29] David Heeger and James Bergen. Pyramid-based texture analysis/synthesis. In *Proc. ACM SIGGRAPH*, pages 229–238, 1995.

[30] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[32] Arnold Irschara, Christopher Zach, Jan-Michael Frahm, and Horst Bischof. From structure-from-motion point clouds to fast location recognition. In *Proc. CVPR*, pages 2599–2606, 2009.

[33] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Proc. NeurIPS*, pages 2017–2025, 2015.

[34] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. *Proc. ECCV*, pages 304–317, 2008.

[35] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Proc. CVPR*, pages 3304–3311. IEEE, 2010.

[36] Yan Ke and Rahul Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *Proc. CVPR*, volume 2, pages II–II. IEEE, 2004.

[37] Michel Keller, Zetao Chen, Fabiola Maffra, Patrik Schmuck, and Margarita Chli. Learning deep descriptors with scale-aware triplet networks. In *Proc. CVPR*. IEEE, 2018.

[38] Alonzo Kelly, Bryan Nagy, David Stager, and Ranjith Unnikrishnan. An infrastructure-free automated guided vehicle based on computer vision. *IEEE Robotics & Automation Magazine*, 14(3):24–34, September 2007.

[39] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proc. ICCV*, pages 2938–2946, 2015.

[40] BG Kumar, Gustavo Carneiro, Ian Reid, et al. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. In *Proc. CVPR*, pages 5385–5394, 2016.

[41] Bastian Leibe, Ales Leonardis, and Bernt Schiele. Combined object categorization and segmentation with an implicit shape model. In *ECCV Workshop on Statistical Learning in Computer Vision*, 2004.

[42] Karel Lenc and Andrea Vedaldi. Learning covariant feature detectors. In *Computer Vision – ECCV Workshops*, pages 100–117. Springer, 2016.

[43] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, June 2001.

[44] Yunpeng Li, Noah Snavely, Dan Huttenlocher, and Pascal Fua. Worldwide pose estimation using 3d point clouds. In *Proc. ECCV*, pages 15–29, 2012.

[45] Yunpeng Li, Noah Snavely, and Daniel Huttenlocher. Location recognition using prioritized feature matching. In *Proc. ECCV*, pages 791–804, 2010.

[46] Hyon Lim, Sudipta N Sinha, Michael F Cohen, Matt Uyttendaele, and H Jin Kim. Real-time monocular image-based 6-dof localization. *International Journal of Robotics Research*, 34(4-5):476–492, April 2015.

[47] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *Proc. CVPR*, pages 1183–1192, 2016.

[48] Siyu Liu. Localization using feature matching in near-random textures. Master's thesis, Princeton University, 2013.

[49] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[50] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[51] Zixin Luo, Tianwei Shen, Lei Zhou, Siyu Zhu, Runze Zhang, Yao Yao, Tian Fang, and Long Quan. Geodesc: Learning local descriptors by integrating geometry constraints. 2018.

[52] Simon Lynen, Torsten Sattler, Michael Bosse, Joel A Hesch, Marc Pollefeys, and Roland Siegwart. Get out of my lab: Large-scale, real-time visual-inertial localization. In *Robotics: Science and Systems XI*, 2015.

[53] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004.

[54] Sven Middelberg, Torsten Sattler, Ole Untzelmann, and Leif Kobbelt. Scalable 6-dof localization on mobile devices. In *Proc. ECCV*, pages 268–283, 2014.

[55] Krystian Mikolajczyk and Cordelia Schmid. An affine invariant interest point detector. In *Proc. ECCV*, pages 128–142. Springer, 2002.

[56] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004.

[57] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. PAMI*, 27(10):1615–1630, 2005.

[58] Andrej Mikulík, Michal Perdoch, Ondřej Chum, and Jiří Matas. Learning a fine vocabulary. In *Proc. ECCV*, pages 1–14, 2010.

[59] Anastasiia Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. In *Proc. NeurIPS*, pages 4829–4840, 2017.

[60] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISAPP*, pages 331–340, 2009.

[61] Arun Mukundan, Giorgos Tolias, and Ondrej Chum. Multiple-kernel local-patch descriptor. *arXiv preprint arXiv:1707.07825*, 2017.

[62] Rafael Munoz-Salinas. Aruco: A minimal library for augmented reality applications based on opencv. `https://www.uco.es/investiga/grupos/ava/node/26`, 2012.

[63] Raul Mur-Artal, JMM Montiel, and Juan D Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Trans. Robotics*, 31(5):1147–1163, October 2015.

[64] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: learning local features from images. In *Proc. NeurIPS*, pages 6237–6247, 2018.

[65] Adam Paszke, Soumith Chintala, Ronan Collobert, Koray Kavukcuoglu, Clement Farabet, Samy Bengio, Iain Melvin, Jason Weston, and Johnny Marietho z. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. `http://pytorch.org/`, 2017.

[66] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[67] James Philbin, Michael Isard, Josef Sivic, and Andrew Zisserman. Descriptor learning for efficient retrieval. In *Proc. ECCV*, pages 677–691, 2010.

[68] Srikumar Ramalingam, Sofien Bouaziz, Peter Sturm, and Matthew Brand. Geolocalization using skylines from omni-images. pages 23–30, 2009.

[69] Andrew Richardson and Edwin Olson. Learning convolutional filters for interest point detection. In *Proc. ICRA*, pages 631–637, 2013.

[70] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Proc. ECCV*, pages 430–443, 2006.

[71] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proc. ICCV*, pages 2564–2571. IEEE, 2011.

[72] Yong Rui, Thomas S Huang, and Shih-Fu Chang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of Visual Communication and Image Representation*, 10(1):39–62, 1999.

[73] Samuele Salti, Alessandro Lanza, and Luigi Di Stefano. Keypoints from symmetries by wave propagation. In *Proc. CVPR*, pages 2898–2905, 2013.

[74] Torsten Sattler. Out with the old? convolutional neural networks for feature matching and visual localization. In *56th Photogrammetric Week: Advancement in Photogrammetry, Remote Sensing and Geoinformatics*, 2017.

[75] Torsten Sattler, Michal Havlena, Filip Radenovic, Konrad Schindler, and Marc Pollefeys. Hyperpoints and fine vocabularies for large-scale location recognition. In *Proc. ICCV*, pages 2102–2110, 2015.

[76] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Fast image-based localization using direct 2d-to-3d matching. In *Proc. ICCV*, pages 667–674, 2011.

[77] Nikolay Savinov, Akihito Seki, Lubor Ladicky, Torsten Sattler, and Marc Pollefeys. Quad-networks: Unsupervised learning to rank for interest point detection. In *Proc. CVPR*, 2017.

[78] Miriam Schönbein and Andreas Geiger. Omnidirectional 3d reconstruction in augmented manhattan worlds. In *Proc. IROS*, pages 716–723, 2014.

[79] Johannes L Schönberger. Colmap, 2016.

[80] Johannes L Schönberger, True Price, Torsten Sattler, Jan-Michael Frahm, and Marc Pollefeys. A vote-and-verify strategy for fast spatial verification in image retrieval. In *Proc. ACCV*, 2016.

[81] Johannes Lutz Schönberger, Hans Hardmeier, Torsten Sattler, and Marc Pollefeys. Comparative evaluation of hand-crafted and learned local features. In *Proc. CVPR*, 2017.

[82] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proc. ICCV*, 2015.

[83] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proc. ICCV*, pages 118–126, 2015.

[84] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Trans. PAMI*, 36(8):1573–1585, 2014.

[85] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graphics*, 25(3):835–846, July 2006.

[86] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3D. *ACM Trans. Graphics*, 25(3):835–846, 2006.

[87] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[88] Christoph Strecha, Alex Bronstein, Michael Bronstein, and Pascal Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Trans. PAMI*, 34(1):66–78, 2012.

[89] Linus Svarm, Olof Enqvist, Fredrik Kahl, and Magnus Oskarsson. City-scale localization for cameras with known vertical direction. *IEEE Trans. PAMI*, 2016.

[90] Yurun Tian, Bin Fan, Fuchao Wu, et al. L2-net: Deep learning of discriminative patch descriptor in euclidean space. In *Proc. CVPR*, volume 2, 2017.

[91] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Trans. PAMI*, 32(5):815–830, May 2010.

[92] Giorgos Tolias, Yannis Avrithis, and Hervé Jégou. To aggregate or not to aggregate: Selective match kernels for image search. In *Proc. ICCV*, pages 1401–1408, 2013.

[93] Tomasz Trzcinski, Mario Christoudias, Pascal Fua, and Vincent Lepetit. Boosting binary keypoint descriptors. In *Proc. CVPR*, pages 2874–2881. IEEE, 2013.

[94] Tomasz Trzcinski and Vincent Lepetit. Efficient discriminative projections for compact binary descriptors. In *Proc. ECCV*, pages 228–242. Springer, 2012.

[95] Tinne Tuytelaars and Cordelia Schmid. Vector quantizing feature space with a regular lattice. In *Proc. ICCV*, pages 1–8. IEEE, 2007.

[96] Yannick Verdie, Kwang Yi, Pascal Fua, and Vincent Lepetit. Tilde: A temporally invariant learned detector. In *Proc. CVPR*, pages 5279–5288, 2015.

[97] Felix Von Hundelshausen and Rahul Sukthankar. D-nets: Beyond patch-based image descriptors. In *Proc. CVPR*, 2012.

[98] Jiayun Wang, Sanping Zhou, Jinjun Wang, and Qiqi Hou. Deep ranking model by large adaptive margin learning for person re-identification. *Pattern Recognition*, 74:241–252, 2018.

[99] Andreas Wendel, Arnold Irschara, and Horst Bischof. Natural landmark-based monocular localization for mavs. In *Proc. ICRA*, pages 5792–5799, 2011.

[100] Simon Winder, Gang Hua, and Matthew Brown. Picking the best daisy. In *Proc. CVPR*, pages 178–185. IEEE, 2009.

[101] Simon AJ Winder and Matthew Brown. Learning local image descriptors. In *Proc. CVPR*, pages 1–8. IEEE, 2007.

[102] Changchang Wu. Siftgpu: A gpu implementation of scale invariant feature transform (sift). `http://cs.unc.edu/~ccwu/siftgpu/`, 2011.

[103] Changchang Wu. Visualsfm: A visual structure from motion system. `http://ccwu.me/vsfm/`, 2011.

[104] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.

[105] Xiaomeng Wu and Kunio Kashino. Adaptive dither voting for robust spatial verification. In *Proc. ICCV*, pages 1877–1885, 2015.

[106] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. *Proc. ECCV*, pages 467–483, 2016.

[107] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proc. CVPR*, pages 4353–4361. IEEE, 2015.

[108] Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[109] Bernhard Zeisl, Torsten Sattler, and Marc Pollefeys. Camera pose voting for large-scale image-based localization. In *Proc. ICCV*, pages 2704–2712, 2015.

[110] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3DMatch: learning local geometric descriptors from RGB-D reconstructions. In *Proc. CVPR*, pages 1802–1811, 2017.

[111] Linguang Zhang, Adam Finkelstein, and Szymon Rusinkiewicz. High-precision localization using ground texture. *CoRR*, abs/1710.10687, 2017.

[112] Linguang Zhang and Szymon Rusinkiewicz. Learning to detect features in texture images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) spotlight presentation*, June 2018.

[113] Xu Zhang, Felix X Yu, Svebor Karaman, and Shih-Fu Chang. Learning discriminative and transformation covariant local feature detectors. In *Proc. CVPR*, pages 6818–6826, 2017.

[114] Xu Zhang, Felix X. Yu, Sanjiv Kumar, and Shih-Fu Chang. Learning spread-out local feature descriptors. In *Proc. ICCV*, Oct 2017.

[115] Barbara Zitova and Jan Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21(11):977–1000, 2003.