Mémoire d'Habilitation à Diriger les Recherches

Statistical Methods for Keypoint and Object Detection and Description

Vincent LEPETIT

défendu le 8 décembre 2015 à l'Université Joseph Fourier de Grenoble

Université Joseph Fourier, Grenoble

À ma famille, et à Sophie qui m'apporte un bonheur que je ne croyais pas possible

Contents

Chapter 1

Introduction

The application of Machine Learning techniques to Computer Vision problems has shown to be very successful [237, 64, 122], at least for some classes of problems. For example, 2D object detection can be naturally cast as a binary classification problem, and image category recognition as a multi-class classification problem. This success, and the popularity of Machine Learning methods for Computer Vision have probably several causes:

Statistical methods are often much more reliable, because they are optimized to perform well on many different samples;

They also require much less hand-tuning, which are often required in Computer Vision otherwise. Typically a few meta-parameters are left to be fixed, or even none with cross-validation;

They can also run very fast, because they can move some of the computation load from run-time to a pre-processing stage by selecting during training an efficient set of operations to perform at run-time. For example, this observation was the starting point of our work on feature point matching [133];

They can also often adapt easily to different modalities. For example, in [252]—presented in Chapter 7— we developed a method for object detection that can take either color images, or depth maps, or both, as input. The same code is used in each of the three cases, only the nature of the training data changes.

Nevertheless, some Computer Vision problems have received comparatively little attention from people armed with Machine Learning techniques, for example:

- image description is typically done with hand-designed methods. The well-known SIFT descriptor [140] and Histograms of Oriented Gradients (HOG) [46] were handcrafted and have been broadly used in Computer Vision. [140] states that the motivation for using histograms of image gradients came from the functions of the simple and complex cells in the Hubel and Wiesel theory for the primary visual cortex [106], but often, the reasoning is less clear, for example in the Local Binary Pattern method. Statistical methods have been introduced only recently, and we present ours in Chapter 4.
- methods for feature extraction are also often suboptimal. For example, the Harris interest point detector, which is rigorously derived in [195], relies on the strong brightness-constancy assumption, and provides no guarantee that the detected corners correspond to well localized 3D points. We developed a statistical method to learn to extract interest points that are resilient to drastic imaging variations. This method is described in Chapter 5.
- 3D problems, such as camera registration or 3D object detection and tracking, are mostly left untouched by the Machine Learning community. The geometric aspects of these problems have been well understood for decades [62, 88]. What is often missing is a reliable way to extract and exploit image information required to solve these problems. In [252], presented in Chapter 7, we introduced a method to learn object descriptors suitable for 3D object detection.

In addition, applying Machine Learning to these problems is not as straightforward as for 2D object detection or image classification. To apply statistical methods from Machine Learning to a Computer Vision problem, one has to solve several aspects:

The problem needs to be formalized in a suitable way, and define what exactly should be predicted. For example, in [232], to learn to detect interest points in images, we introduced an approach in which we predict a value for each pixel location, so that the values have local maximums at the desired locations for feature points;

An optimization problem whose solution is a reliable prediction method. Also in [232], we use constraints across training images to ensure that the predicted values remain similar even when the illumination conditions vary.

A training set of annotated samples needs to be created. To keep using the example of [232], we used images captured by webcams over a year to have samples of the same 3D points under many different lighting conditions. We also had to design an algorithm to identify image locations to learn to detect. In [168, 252], we rendered synthetic color views of 3D objects or their depth maps to augment the training set and increase the performance. Actually, the need for a training set is starting to be perceived as the limiting factor in many other problems as well, and many researchers are turning to non-supervised, semi-supervised, or other alternative approaches.

Over the last decade, together with brilliant young researchers, either PhD candidates or post-docs, at CVLab, EPFL, headed by Prof. Pascal Fua and later in my group at ICG, TU Graz, we have been revisiting with a toolbox of Machine Learning techniques keypoint detection and description, and 3D object detection, for which only handcrafted approaches existed. This consistently yields an increase of accuracy and reliability. This manuscript presents some of these works.

Of course, we were not alone to work in this direction. Simon Winder and Matthew Brown introduced local descriptor learning [250], Piotr Dollar and Pablo Arbelaez worked on contour detection learning [55, 11], Pablo and Saurabh Gupta [85] worked on Machine Learning for 3D object detection, Jamie Shotton on 3D registration [26, 196], just to name a few.

The next chapters are mainly based on the following publications:

Chapter 2 M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition Using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448–461, 2010.

Chapter 3 M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, 2012.

Chapter 4 T. Trzcinski, M. Christoudias, and V. Lepetit. Learning Image Descriptors with Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

Chapter 5 Y. Verdie, K. M. Yi, P. Fua, and V. Lepetit. TILDE: A Temporally Invariant Learned DEtector. In *Conference on Computer Vision and Pattern Recognition*, 2015.

Chapter 6 S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient Response Maps for Real-Time Detection of Textureless Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, May 2012.

Chapter 7 P. Wohlhart and V. Lepetit. Learning Descriptors for Object Recognition and 3D Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*, 2015.

In several places, I revised the original text to incorporate my current point of view, whenever it is different from the one I had when the first version was written. These updates are marked with gray bars in the left margin.

The last chapter presents the directions I intend to pursue in the next years.

Chapter 2

Fast Keypoint Recognition using Random Ferns

2.1 Introduction

In [130], we showed that casting the feature point matching problem as a more generic classification problem leads to solutions that are much less computationally demanding. We relied on an offline training phase during which multiple views of the patches to be matched are used to train Randomized Trees [8] to recognize them based on a few pairwise intensity comparisons. This yields both fast run-time performance and robustness to viewpoint and lighting changes.

During the PhD thesis of Mustafa Özuysal, we developed a novel classifier that we refer to as *ferns* to classify the patches. Each "fern" consists of a small set of binary tests and returns the probability that a patch belongs to any one of the classes that have been learned during training. These responses are then combined in a Naive Bayesian way. As in [130], we train our classifier by synthesizing many views of the keypoints extracted from a training image as they would appear under different perspective or scale.

The advantage of the ferns over the trees is that they are faster and simpler to implement, while being just as reliable. The code that implements patch evaluation can be written in ten lines of C++ code, which highlights the simplicity of the resulting implementation.

The binary tests we use as image features are the same as the ones we used in [130] and are picked completely at random, which puts our approach firmly in the camp of techniques that rely on randomization to achieve good performance [7]. We will show that this is particularly effective for applications such as real-time 3D object detection and *Simultaneous Localization and Mapping* (SLAM) that require scale and perspective invariance, involve a very large number of classes, but can tolerate significant error rates since we use robust statistical methods to exploit the information provided by the correspondences. Furthermore, our approach is particularly easy to implement, does not overfit, does not require *ad hoc* patch normalization, and allows fast and incremental training.

2.2 Related Work

Due to its robustness to partial occlusions and computational efficiency, recognition of image patches extracted around detected keypoints is crucial for many vision problems. As a result, two main classes of approaches have been developed to achieve robustness to perspective and lighting changes.

The first family relies on local descriptors designed to be invariant, or at least robust, to specific classes of deformations [190, 140]. These approaches usually rely on the fine scale and rotation estimates provided by the keypoint detector. Among these, the SIFT vector [140], computed from local histograms of gradients, has been shown to work remarkably well and we will use it as a benchmark for our own approach. It has also been shown that keypoints can be used as visual words [201] for image retrieval in very large image databases [158]. Keypoints are labeled by hierarchical k-means [23] clustering based on their SIFT descriptors. This makes it possible to use very many visual words. However, performance is measured in terms of the number of correctly retrieved documents

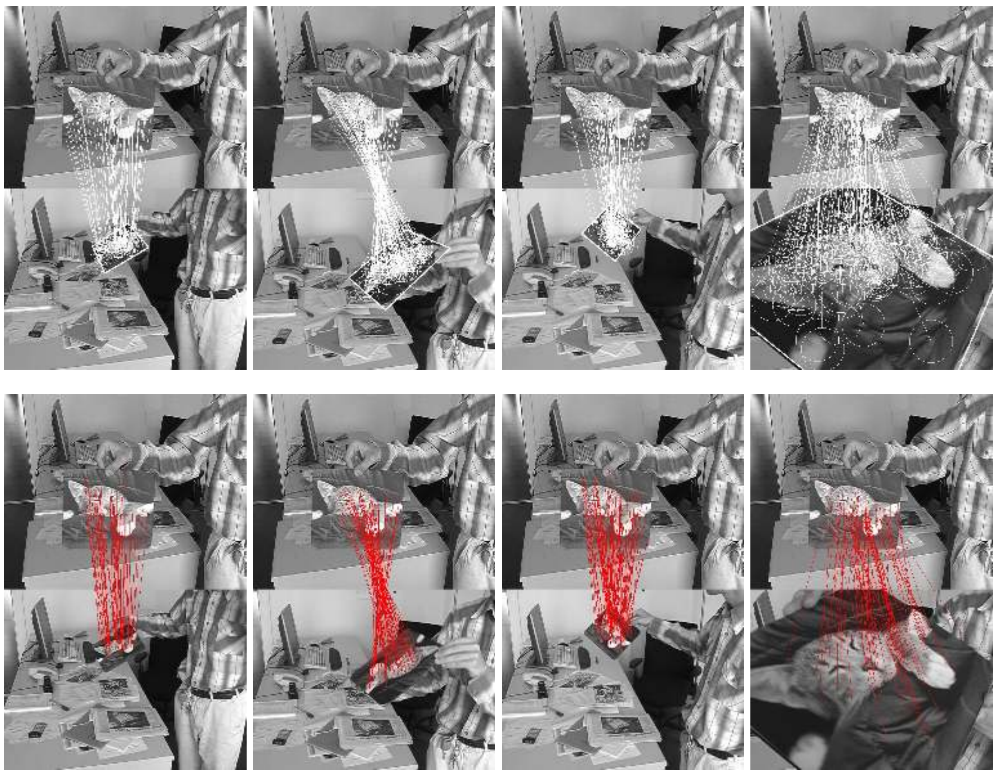


Figure 2.1: Matching a mouse pad in a 1074-frame sequence against a reference image. The reference image appears at the top and the input image from the video sequence at the bottom. : **Top row.** Matches obtained using ferns in a few frames. **Bottom row.** Matches obtained using SIFT in the same frames.

rather than the number of correctly classified keypoints, which is the important criterion for applications such as pose estimation or SLAM.

A second class relies on statistical learning techniques to compute a probabilistic model of the patch. The one-shot approach of [63] uses PCA and Gaussian Mixture Models but does not account for perspective distortion. Since the set of possible patches around an image feature under changing perspective and lightning conditions can be seen as a class, we showed that this problem can be overcome by training a set of Randomized Trees [8] to recognize feature points independently of pose [130]. This is done using a database of patches obtained by warping those found in a training image by randomly chosen homographies.

This approach is fast and effective to achieve the kind of object detection depicted by Figs 2.1 and 2.2. Note that unlike in traditional classification problems, a close-to-perfect method is not required, because the output of the classifier can be filtered by a robust estimator. However the classifier should be able to handle many classes— typically more than 200— simultaneously without compromising performance or speed. We will demonstrate that the approach we propose is even faster than the trees [130, 189], more scalable, and just as reliable.

Local Binary Patterns [165], also rely on binary feature statistics by describing the underlying texture in terms of histograms of binary features over all pixels of a target region. While such a description is appropriate for texture classification, it is not directly suitable for keypoint characterization since histograms are built in this way, it will lose the spatial information between the features. By contrast, we compute the statistics of the binary features over example patches seen from different viewpoints and use independence assumptions between groups of features, hence using many more features centered on the keypoint location, to improve the recognition rate.

The Real-Time SLAM method of [249] also extends the RT's into lists of features (similar to our Ferns), but the full posterior distribution over binary features are replaced by a single bit. This design choice is aimed at significantly reducing the memory requirements while correctly matching the sparse set of visible landmarks at a time. For maximum performance, we model the full joint probability. Memory requirements can be tackled by using fixed point representations that require fewer bits than the standard floating point representation.

Trees and Ferns have recently been used for image classification , as a replacement for a multi-way Support Vector Machine [25]. The binary features are computed on shape and texture descriptors, hence gaining invariance

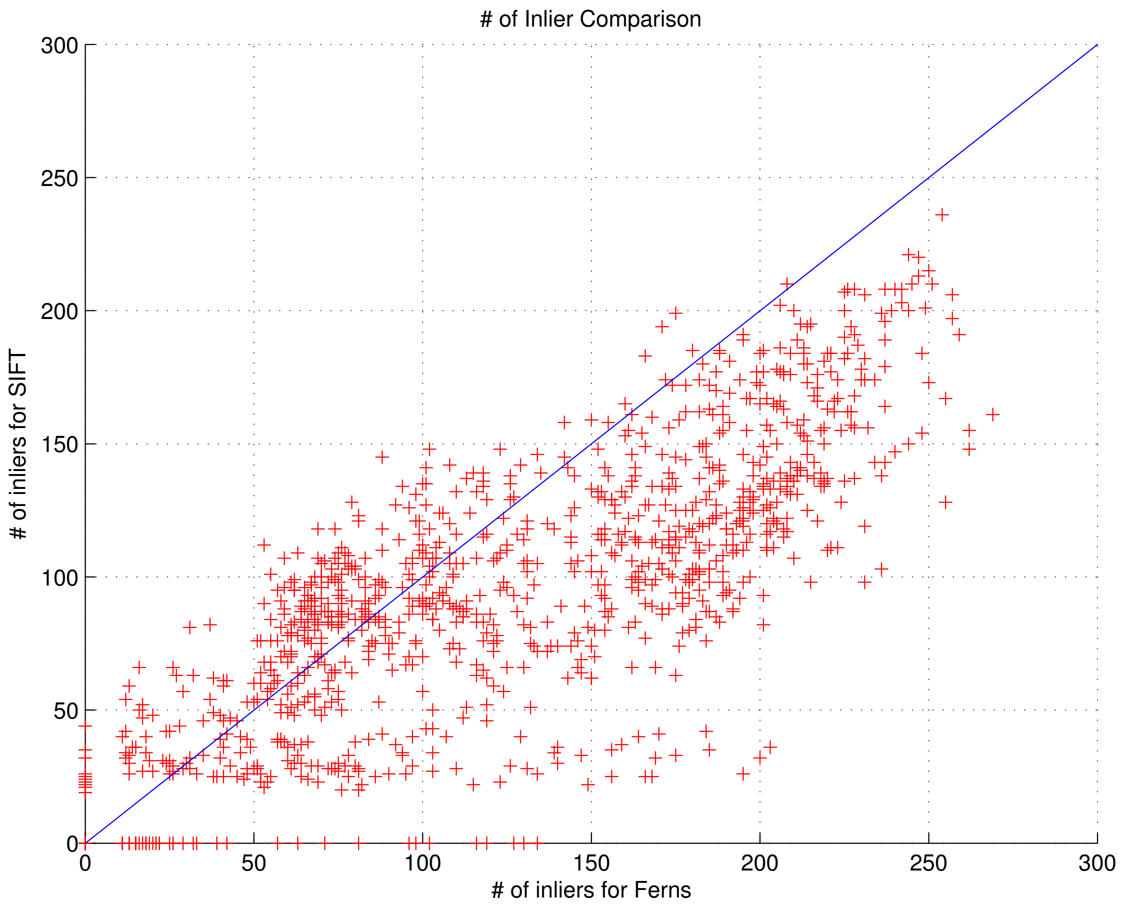


Figure 2.2: Scatter plot showing the number of inliers for each frame for the experiment in Fig. 2.1. The values on the axes give the number of inliers for Ferns and SIFT. Most of the time, the Ferns match at least as many points as SIFT and often even more, as can be seen from the fact that most of the points lay below the diagonal.

to local deformations. The distributions are computed over different instances of the same class, but unlike our approach the posteriors from different Trees and Ferns are combined by averaging. The results match our own observations that using either Fern or Tree structures leads to similar classification performance.

2.3 A Semi-Naive Bayesian Approach to Patch Recognition

As discussed in Section 2.2, image patches can be recognized on the basis of very simple and randomly chosen binary tests that are grouped into decision trees and recursively partition the space of all possible patches [130]. In essence, we treat the set of possible appearances of a keypoint as classes and the Randomized Trees (RTs) embody a probability distribution over these classes. In practice, no single tree is discriminative enough when there are many classes. However, using a number of trees and averaging their votes yields good results because each one partitions the feature space in a different way.

In this section, we will argue that, when the tests are chosen randomly, the power of the approach derives not from the tree structure itself but from the fact that combining groups of binary tests allows improved classification rates. Therefore, replacing the trees by our non-hierarchical ferns and pooling their answers in a Naive Bayesian manner yields better results and scalability in terms of number of classes. As a result, we can combine many more features, which is key to improved recognition rates.

We first show that our non-hierarchical Ferns fit nicely into a Naive Bayesian framework and then explain the training protocol which is similar to the one used for RTs.

2.3.1 Formulation of Feature Combination

As discussed in Section 2.2 we treat the set of all possible appearances of the image patch surrounding a keypoint as a class. Therefore, given the patch surrounding a keypoint detected in an image, our task is to assign it to the most likely class. Let $c_i, i = 1, \ldots, H$ be the set of classes and let $f_j, j = 1, \ldots, N$ be the set of binary features that will be calculated over the patch we are trying to classify. Formally, we are looking for

$$\hat{c_i} = \underset{c_i}{\text{argmax}}\, P(C = c_i \mid f_1, f_2, \ldots, f_N) \,,$$

where C is a random variable that represents the class. Bayes' Formula yields

$$P(C = c_i \mid f_1, f_2, \ldots, f_N) = \frac{P(f_1, f_2, \ldots, f_N \mid C = c_i) P(C = c_i)}{P(f_1, f_2, \ldots, f_N)} \,.$$

Assuming a uniform prior $P(C)$, since the denominator is simply a scaling factor that it is independent from the class, our problem reduces to finding

$$\hat{c_i} = \underset{c_i}{\text{argmax}}\, P(f_1, f_2, \ldots, f_N \mid C = c_i) \,. \tag{2.1}$$

In our implementation, the value of each binary feature f_j only depends on the intensities of two pixel locations $\mathbf{d}_{j,1}$ and $\mathbf{d}_{j,2}$ of the image patch. We therefore write

$$f_j = \begin{cases} 1 \text{ if } I(\mathbf{d}_{j,1}) < I(\mathbf{d}_{j,2}) \\ 0 \text{ otherwise} \end{cases} \,,$$

where I represents the image patch. Since these features are very simple, we require many $(N \approx 300)$ for accurate classification. Therefore a complete representation of the joint probability in Eq. (2.1) is not feasible since it would require estimating and storing 2^N entries for each class. One way to compress the representation is to assume independence between features. An extreme version is to assume complete independence, that is,

$$P(f_1, f_2, \ldots, f_N \mid C = c_i) = \prod_{j=1}^{N} P(f_j \mid C = c_i) \,.$$

However this completely ignores the correlation between features. To make the problem tractable while accounting for these dependencies, a good compromise is to partition our features into M groups of size $S = \frac{N}{M}$. These groups are what we define as *Ferns* and we compute the joint probability for features in each Fern. The conditional probability becomes

$$P(f_1, f_2, \ldots, f_N \mid C = c_i) = \prod_{k=1}^{M} P(F_k \mid C = c_i) \,, \tag{2.2}$$

where $F_k = \{f_{\sigma(k,1)}, f_{\sigma(k,2)}, \ldots, f_{\sigma(k,S)}\}, k = 1, \ldots, M$ represents the k^{th} fern and $\sigma(k, j)$ is a random permutation function with range $1, \ldots, N$. Hence, we follow a Semi-Naive Bayesian [257] approach by modelling only some of the dependencies between features. The viability of such an approach has been shown by [102] in the context of image retrieval applications.

This formulation yields a tractable problem that involves $M \times 2^S$ parameters, with M between 30-50. In practice, as will be shown in Section 2.4, $S = 11$ yields good results. $M \times 2^S$ is therefore in the order of 80000, which is much smaller than 2^N with $N \approx 450$ that the full joint probability representation would require. Our formulation is also flexible since performance/memory trade-offs can be made by changing the number of Ferns and their sizes.

Note that we use randomization in feature selection but also in grouping. An alternative approach would involve selecting feature groups to be as independent from each other as possible. This is routinely done by Semi-Naive Bayesian classifiers based on a criteria such as the mutual information between features [43]. However, in practice, we have not found this to be necessary to achieve good performance. We have therefore chosen not to use such a strategy to preserve the simplicity and efficiency of our training scheme and to allow for incremental training.

2.3.2 Training

We assume that at least one image of the object to be detected is available for training. We call any such image as a *model* image. Training starts by selecting a subset of the keypoints detected on these model images. This is done by deforming the images many times, applying the keypoint detector, and keeping track of the number of times the same keypoint is detected. The keypoints that are found most often are assumed to be the most stable and retained. These stable keypoints are assigned a unique class number.

The training set for each class is formed by generating 10000 sample images with randomly picked affine deformations by sampling the deformation parameters from a uniform distribution, adding Gaussian noise to each sample image, and smoothing with a Gaussian filter of size 7×7. This increases the robustness of the resulting classifier to run-time noise, especially when there are features that compare two pixels on a uniform area.

The training phase estimates the class conditional probabilities $P(F_m \mid C = c_i)$ for each Fern F_m and class c_i, as described in Eq. 2.2. For each Fern F_m we write these terms as:

$$p_{k,c_i} = P(F_m = k \mid C = c_i) \ , \tag{2.3}$$

where we simplify our notations by considering F_m to be equal to k if the base 2 number formed by the binary features of F_m taken in sequence is equal to k. With this convention, Ferns can take $K = 2^S$ values and, for each one, we need to estimate the $p_{k,c_i}, k = 1, 2, \ldots, K$ under the constraint

$$\sum_{k=1}^{K} p_{k,c_i} = 1 \ .$$

The simplest approach would be to assign the maximum likelihood estimate to these parameters from the training samples. For parameter p_{k,c_i} it is

$$p_{k,c_i} = \frac{N_{k,c_i}}{N_{c_i}} \ ,$$

where N_{k,c_i} is the number of training samples of class c_i that evaluates to Fern value k and N_{c_i} is the total number of samples for class c_i. These parameters can therefore be estimated for each Fern independently.

In practice however, this simple scheme yields poor results because if no training sample for class c_i evaluates to k, which can easily happen when the number of samples is not infinitely large, both N_{k,c_i} and p_{k,c_i} will be zero. Since we multiply the p_{k,c_j} for all Ferns, it implies that, if the Fern evaluates to k, the corresponding patch can *never* be associated to class c_i, no matter the response of the other Ferns. This makes the Ferns far too selective because the fact that $p_{k,c_i} = 0$ may simply be an artifact of the necessarily limited size of the training set. To overcome this problem we take p_{k,c_i} to be

$$p_{k,c_i} = \frac{N_{k,c_i} + N_r}{N_{c_i} + K \times N_r} \ ,$$

where N_r represents a regularization term, which behaves as a uniform Dirichlet prior [21] over feature values. If a sample with a specific Fern value is not encountered during training, this scheme will still assign a non-zero value to the corresponding probability. As illustrated by Fig. 2.3, we have found our estimator to be insensitive to the exact value of N_r and we use $N_r = 1$ in all our experiments. However, having N_r be strictly greater than zero is essential. This tallies with the observation that combining classifiers in a Naive Bayesian fashion can be unreliable if improperly done [119].

In effect, our training scheme marginalizes over the pose space since the class conditional probabilities $P(F_m \mid C = c_i)$ depend on the camera poses relative to the object. By densely sampling the pose space and summing over all samples, we marginalize over these pose parameters. Hence at run-time, the statistics can be used in a pose independent manner, which is key to real-time performance. Furthermore, the training algorithm itself is very efficient since it only requires storing the N_{k,c_i} counts for each fern while discarding the training samples immediately after use, which means that we can use arbitrarily many if need be.

2.4 Comparison with Randomized Trees

As shown in Figs 2.4 and 2.5, Ferns can be considered as simplified trees. Whether or not this simplification degrades their classification performance hinges on whether our randomly chosen binary features are still appropriate

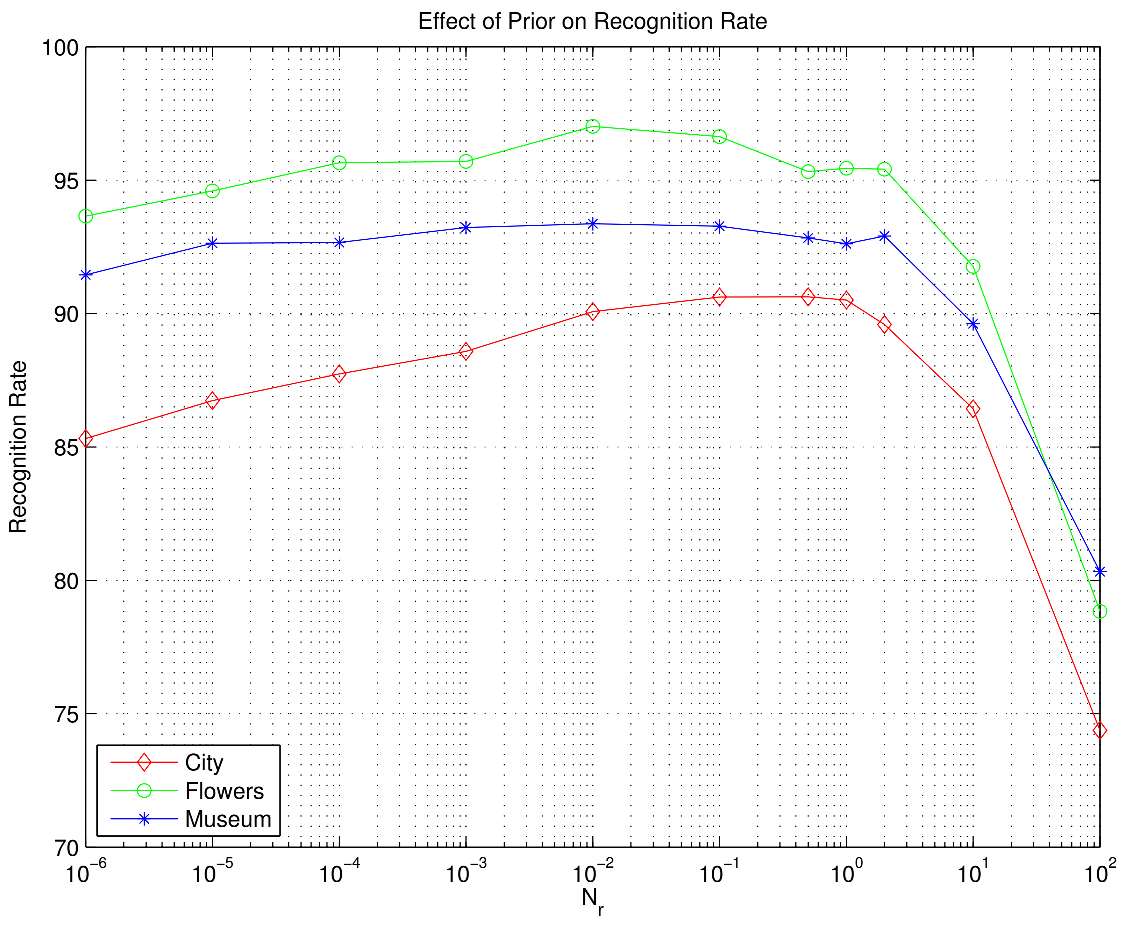


Figure 2.3: Recognition rate as a function of $\log(N_r)$ using the three test images of Section 2.4. The recognition rate remains relatively constant for $0.001 < N_r < 2$. For $N_r < 0.001$ it begins a slow decline, which ends in a sudden drop to about 50% when $N_r = 0$. The rate also drops when N_r is too large because too strong a prior decreases the effect of the actual training data, which is around 10000 samples for this experiment.

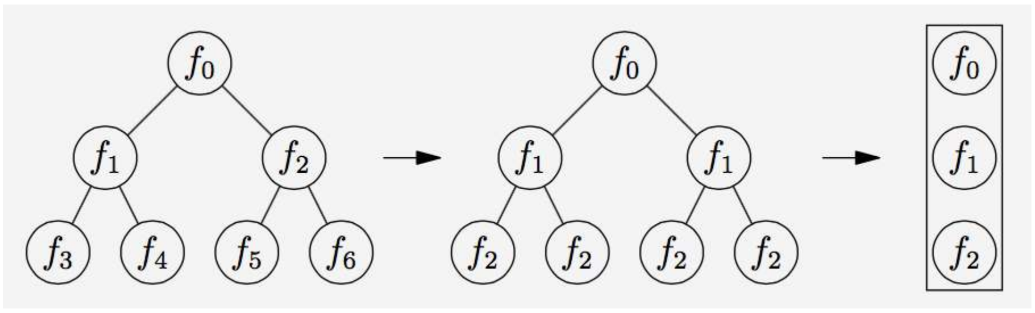


Figure 2.4: Ferns vs Trees. A tree can be transformed into a Fern by performing the following steps. First, we constrain the tree to systematically perform the same test across any given hierarchy level, which results in the same feature being evaluated independently of the path taken to get to a particular node. Second, we do away with the hiearchical structure and simply store the feature values at each level. This means applying a sequence of tests to the patch, which is what Ferns do.

in this context. In this section, we will show that they are indeed. In fact, because our Naive Bayesian scheme outperforms the averaging of posteriors used to combine the output of the decision trees [130], the Ferns are both simpler and more powerful.

To compare RTs and Ferns, we experimented with the three images of Fig. 2.6. We warp each image by repeatedly applying random affine deformations and detect Harris corners in the deformed images. We then select

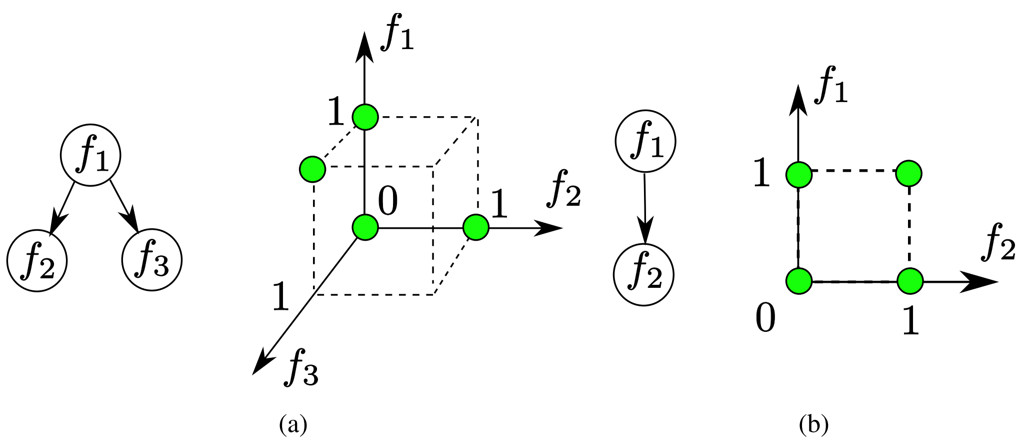


Figure 2.5: The feature spaces of Trees and Ferns. Although the space of tree features seems much higher dimensional, it is not because only a subset of features can be evaluated. (a) For the simple tree on the left only the 4 combination of feature values denoted by green circles are possible. (b) The even simpler Fern on the left also yields 4 possible combinations of feature values but a much simpler structure. As shown below, this simplicity does not entail any performance loss.

Figure 2.6: The recognition rate experiments are performed on three images that show different texture and structures. Image size is 640×480 pixels.

the most stable 250 keypoints per image based on how many times they are detected in the deformed versions to use in the following experiments and assign a unique class id to each of them. The classification is done using patches that are 32×32 pixels in size.

Ferns differ from trees in two important respects: The probabilities are multiplied in a Naive-Bayesian way instead of being averaged and the hierarchical structure is replaced by a flat one. To disentangle the influence of these changes, we consider four different scenarios:

- Using Randomized Trees and averaging of class posterior distributions, as in [130],
- Using Randomized Trees and combining class conditional distributions in a Naive-Bayesian way,
- Using Ferns and averaging of class posteriors,
- Using Ferns and combining class conditional distributions in a Naive-Bayesian way, as we advocate in this chapter.

The trees are of depth 11 and each Fern has 11 features, yielding the same number of parameters for the estimated distributions. Also the number of features evaluated per patch is equal in all cases.

The training set is obtained by randomly deforming images of Fig. 2.6. To perform these experiments, we represent affine image deformations as 2×2 matrices of the form

$$R_\theta R_{-\phi} \mathrm{diag}(\lambda_1, \lambda_2) R_\phi \quad ,$$

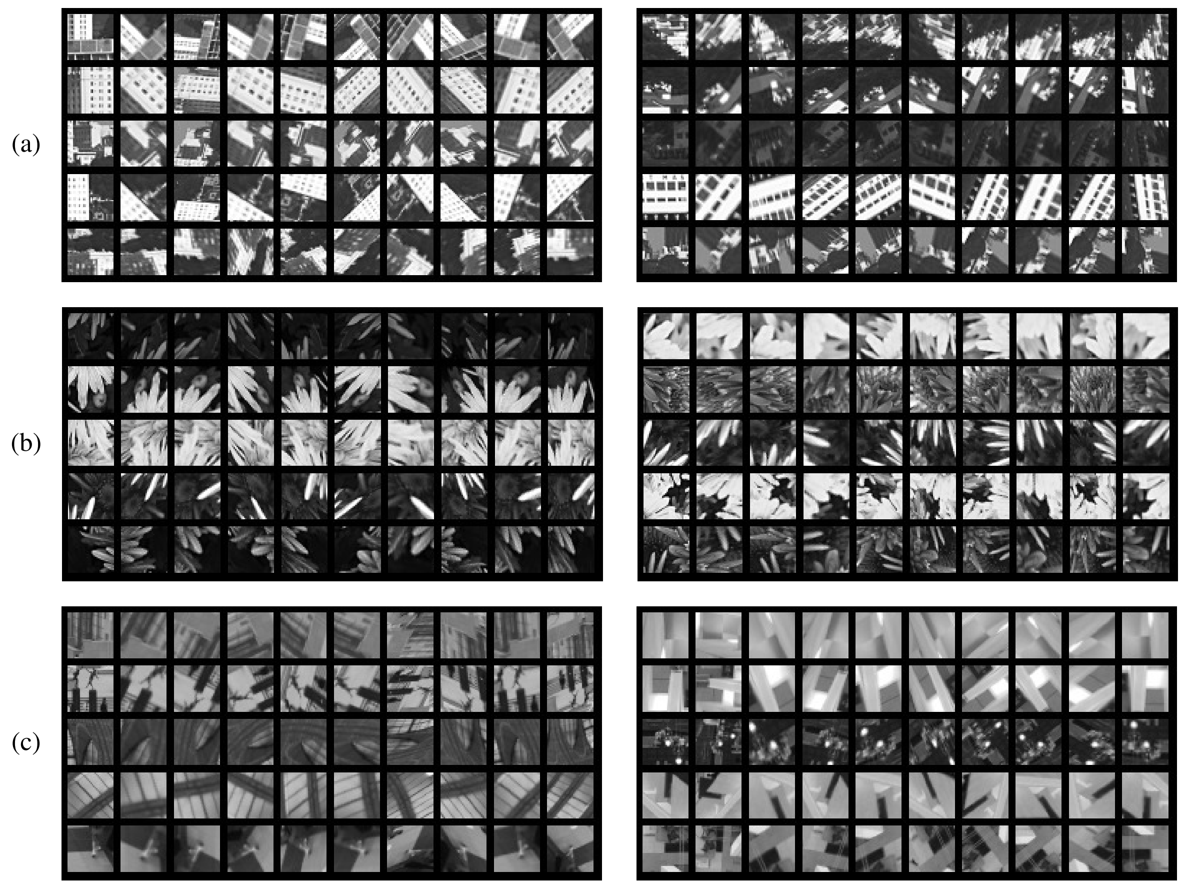


Figure 2.7: Warped patches from the images of Fig. 2.6 show the range of affine deformations we considered. In each line, the left most patch is the original one and the others are deformed versions of it. (a) Sample patches from the *City* image. (b) Sample patches from the *Flowers* image. (c) Sample patches from the *Museum* image.

Table 2.1: Variance of the recognition rate.

Number of Structures	10	15	20	25	30	35	40	45	50
Fern-Naive	0.5880	0.2398	0.1116	0.0624	0.0481	0.0702	0.0597	0.0754	0.0511
Fern-Avg	0.3333	0.2138	0.2869	0.2360	0.3522	0.3930	0.3469	0.4087	0.2973
Tree-Naive	0.3684	0.1861	0.1255	0.0663	0.0572	0.0379	0.0397	0.0186	0.0095
Tree-Avg	0.3009	0.2156	0.1284	0.1719	0.1631	0.1694	0.1497	0.1683	0.1954

where $\mathrm{diag}(\lambda_1, \lambda_2)$ is a diagonal 2×2 matrix and R_γ represents a rotation of angle γ. Both to train and to test our ferns, we warped the original images using such deformations computed by randomly choosing θ and ϕ in the $[0 : 2\pi]$ range and λ_1 and λ_2 in the $[0.6 : 1.5]$ range. Fig. 2.7 depicts patches surrounding individual interest points first in the original images and then in the warped ones. We used 30 random affine deformations per degree of rotation to produce 10800 images. As explained in Section 2.3.2, we then added Gaussian noise with zero mean and a large variance—25 for gray levels ranging from 0 to 255—to these warped images to increase the robustness of the resulting ferns. Gaussian smoothing with a mask of 7×7 is applied to both training and test images.

The test set is obtained by generating a separate set of 1000 images in the same affine deformation range and adding noise. Note that we simply transform the original keypoint locations, therefore we ignore the keypoint detector's repeatability in the tests and measure only the recognition performance. In Fig. 2.8, we plot the results as a function of the number of trees or Ferns being used.

We first note that using either flat Fern or hierarchical tree structures does not affect the recognition rate, which

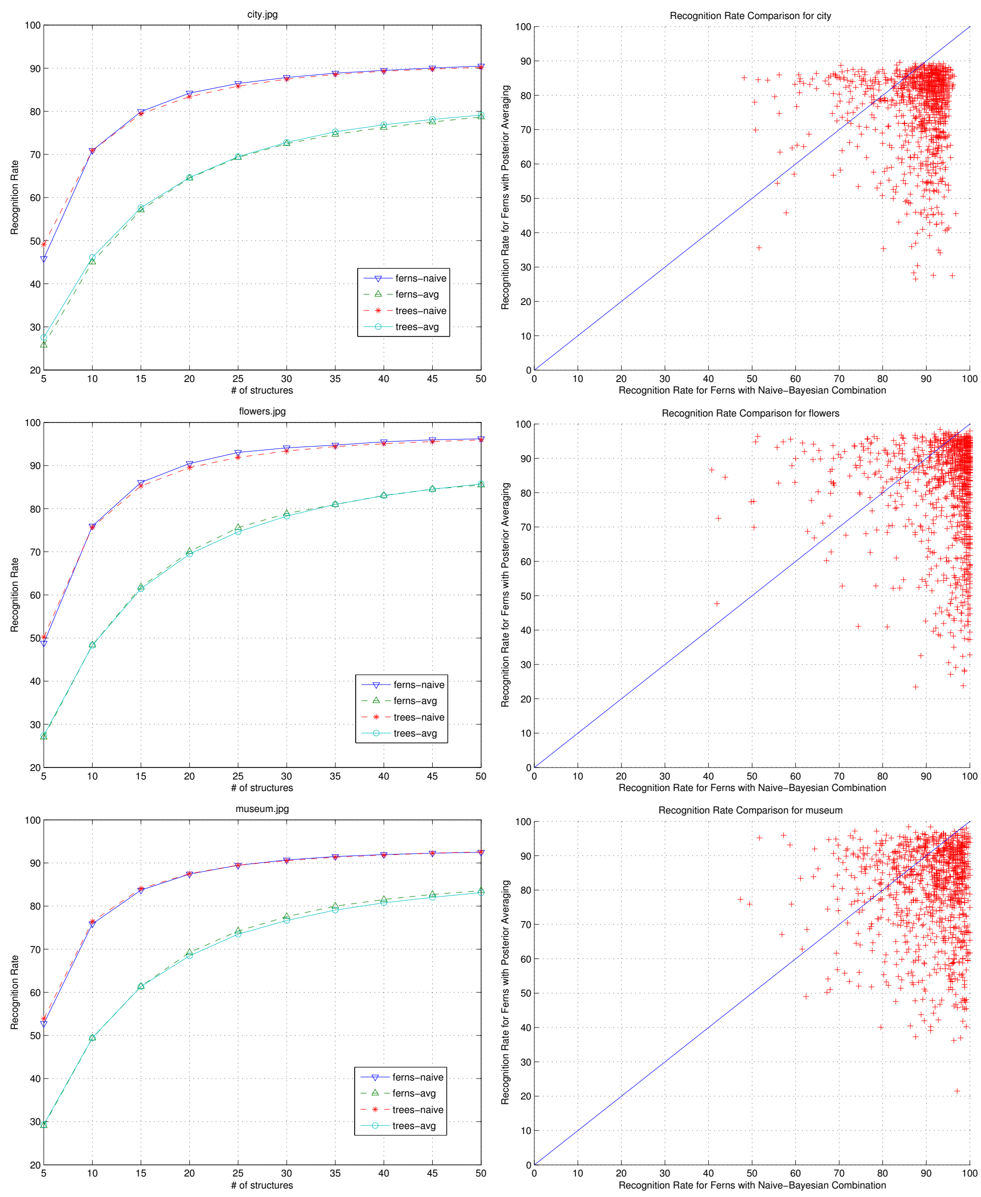


Figure 2.8: **Left Column.** The average percentage of correctly classified image patches over many trials is shown as the number of Trees or Ferns is changed. Using the Naive Bayesian assumption gives much better rates at reduced number of structures, while the Fern and tree structures are interchangeable. **Right Column.** The scatter plots show the recognition rate over individual trials with 50 Ferns. The recognition rates for the Naive-Bayesian combination and posterior averaging are given in the $\mathbf{x}$ and $\mathbf{y}$ axes, respectively. The Naive-Bayesian combination of features usually performs better, as evidenced by the fact that most points are below the diagonal, and only very rarely produce recognition rates below 80%. By contrast the averaging produces rates below 60%.

was to be expected as the features are taken completely at random. By contrast the Naive-Bayesian combination strategy outperforms the averaging of posteriors and achieves a higher recognition rate even when using relatively few structures. Furthermore as the scatter plots of Fig. 2.8 show, for the Naive-Bayesian combination the recognition rate on individual deformed images never falls below an acceptable rate. Since the features are taken randomly, the recognition rate changes and the variance of the recognition rate is given as Table 2.1. As more Ferns or Trees are used the variance decreases and more rapidly for the naive combination. If the number of Ferns or Trees are below 10, the recognition rate starts to change more erratically and entropy based optimization of feature selection becomes a necessity.

To test the behavior of the methods as the number of classes is increased, we have trained classifiers for matching up to 1500 classes. Fig. 2.9 shows that the performance of the Naive-Bayesian combination does not

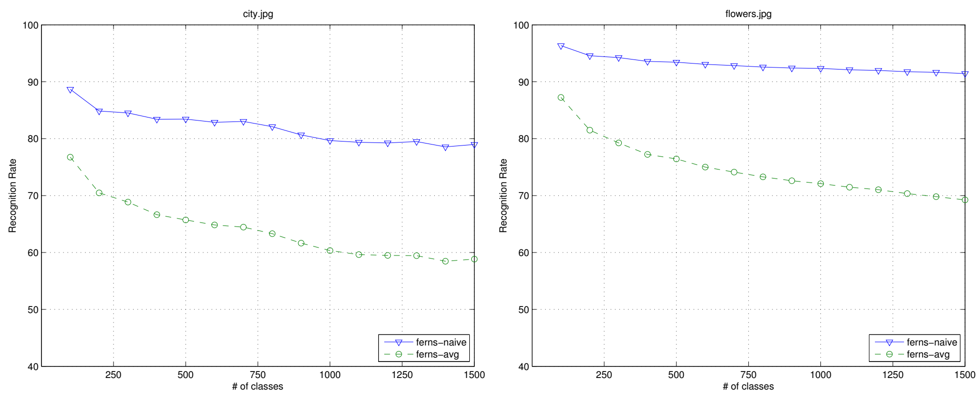
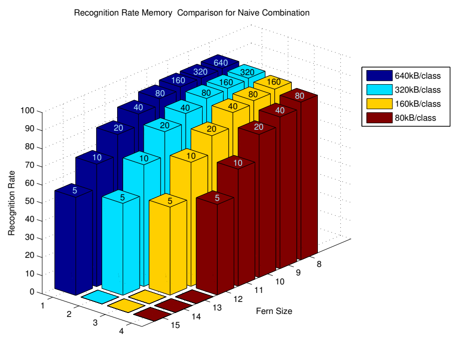


Figure 2.9: Recognition rate as a function of the number of classes. While the naive combination produces a very slow decrease in performance, posterior averaging exhibits a much sharper drop. The tests are performed on the high resolution versions of the *City* and *Flowers* data, respectively.


(a)

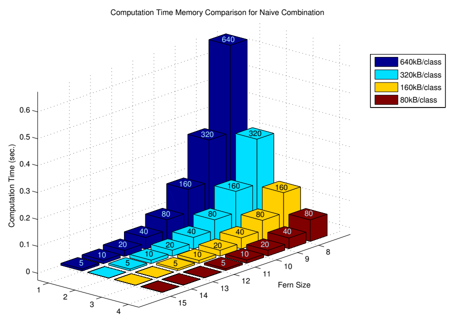

(b)

Figure 2.10: Recognition rate (a) and computation time (b) as a function of the amount of memory available and the size of the Ferns being used. The number of ferns used is indicated on the top of each bar and the y-axis shows the Fern size. The color of the bar represents the required memory amount, which is computed for distributions stored with single precision floating numbers. Note that while using many small ferns achieves higher recognition rates, it also entails a higher computational cost.

degrade rapidly and scales much better than averaging posteriors. For both methods, the required amounts of memory and computation times increase linearly with the number of classes, since we assign a separate class for each keypoint.

So far, we have used 11 features for each Fern, and we now discuss the influence of this number on recognition performance, and memory requirements.

Increasing the Fern size by one doubles the number of parameters hence the memory required to store the distributions. It also implies that more training samples should be used to estimate the increased number of parameters. It has however negligible effect on the run-time speed and larger Ferns can therefore handle more variation at the cost of training time and memory but without much of a slow-down.

By contrast adding more Ferns to the classifier requires only a linear increase in memory but also in computation time. Since the training samples for other Ferns can be reused it only has a negligible effect on training time. As shown in Fig. 2.10, for a given amount of memory the best recognition rate is obtained by using many relatively small Ferns. However this comes at the expense of run-time speed and when sufficient memory is available, a Fern size of 11 represents a good compromise, which is why we have used this value in the experiments.

Finally we evaluate the behavior of the classifier as a function of the number of training samples. Initially, we use 180 training images that we generate by warping the images of Fig. 2.6 by random scaling parameters and deformation angle ϕ, while the rotation angle θ is uniformly sampled at every two degrees. We then increase the number of training samples by 180 at each step. The graphs depicted by Fig. 2.11 show that the Naive-Bayesian combination performs consistently better than the averaging, even when only a small number of training samples

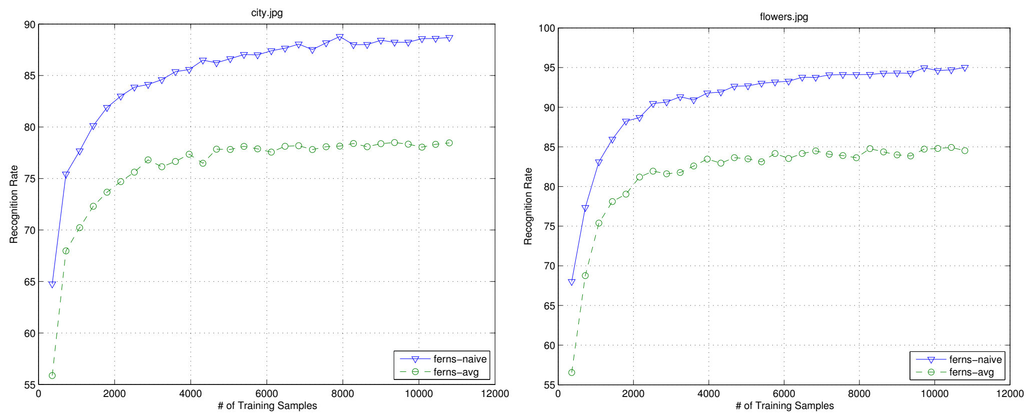


Figure 2.11: Recognition rate as a function of the number of training samples for each one of the three images of Fig. 2.6. As more training samples are used the recognition rate increases and it does so faster for the Naive-Bayesian combination of Ferns.

are used.

2.5 Experiments

We evaluate the performance of Fern based classification for both planar and fully three dimensional object detection and SLAM based camera tracking. We also compare our approach against SIFT [140], which is among the most reliable descriptors for patch matching.

2.5.1 Ferns vs SIFT to detect planar objects

We used the 1074-frame video depicted by Fig. 2.1 to compare Ferns against SIFT for planar object detection. It shows a mouse pad undergoing motions involving a large range of rotations, scalings, and perspective deformations against a cluttered background. We used as a reference an image in which the mouse pad is seen frontally. We match the keypoints extracted from each input image against those found in the reference image using either Ferns or SIFT and then eliminate outliers by computing a homography using RANSAC.

The SIFT keypoints and the corresponding descriptors are computed using the publicly available code kindly provided by David Lowe. The keypoint detection is based on the Difference of Gaussians over several scales and for each keypoint dominant orientations are precomputed. By contrast the Ferns rely on a simpler keypoint detector that computes the maxima of Laplacian on three scales, which provides neither dominant orientation information nor a finely estimated scale. We retain the 400 strongest keypoints in the reference, and 1000 keypoints in the input images for the two methods.

We train 20 Ferns of size 14 to establish matches with the keypoints on the video frame by selecting the most probable class. In parallel, we match the SIFT descriptors for the keypoints on the reference image against the keypoints on the input image by selecting the one which has the nearest SIFT descriptor. Given the matches between the reference and input image, we use a robust estimation followed by non-linear refinement to estimate a homography. We then take all matches with reprojection error less than 10 pixels to be inliers. Fig. 2.1 shows that the Ferns can match as many points as SIFT and sometimes even more.

It is difficult to perform a completely fair speed comparison between our Ferns and SIFT for several reasons. SIFT reuses intermediate data from the keypoint extraction to compute canonic scale and orientations and the descriptors, while ferns can rely on a low-cost keypoint extraction. On the other hand, the distributed SIFT C code is not optimized, and the Best-Bin-First KD-tree of [19] is not used to speed up the nearest-neighbor search.

However, it is relatively easy to see that our approach requires much less computation. Performing the individual tests of Section 2.3 requires very little time and most of the time is spent computing the sums of the posterior probabilities. The classification of a keypoint requires $H \times M$ additions, with H the number of classes, and M the number of Ferns. By contrast, SIFT uses $128H$ additions and as many multiplications when the Best-Bin-First

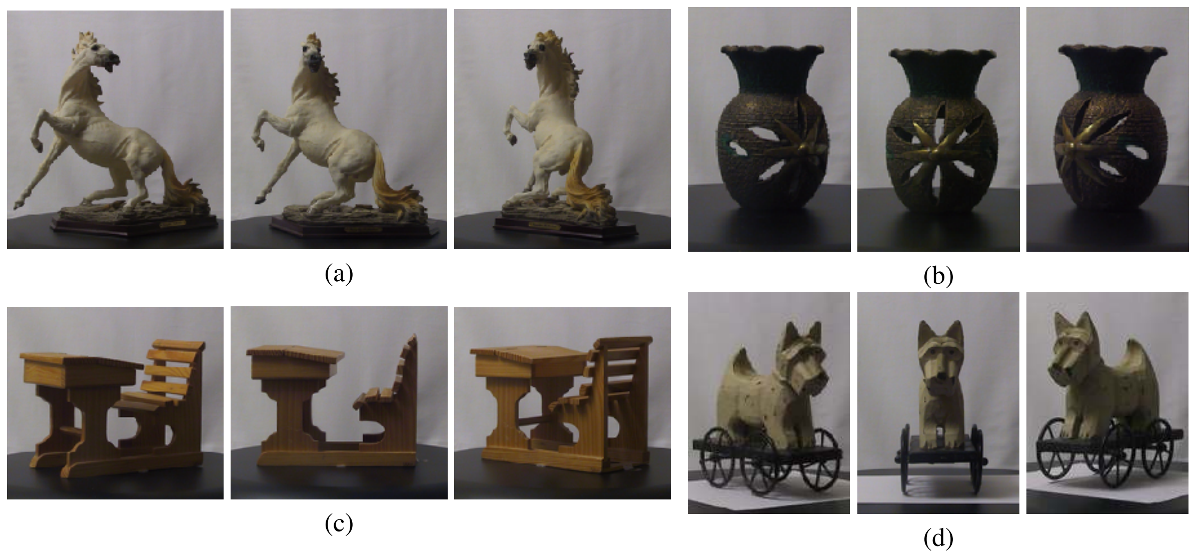


Figure 2.12: When detecting a 3D object viewpoint change is more challenging due to self-occlusions and non-trivial lighting effects. The images are taken from a database presented in [154] and cover a total range of 70° of camera rotation. They are cropped around the object, while we used the original images in the experiments. (a) *Horse* dataset. (b) *Vase* dataset. (c) *Desk* dataset. (d) *Dog* dataset.

KD-tree is not used. This represents an obvious advantage of our method at run-time since M can be much less than 128 and is taken to be 20 in practice, while selecting a large number of features for each fern and using tens of thousands of training samples.

The major gain actually comes from the fact that Ferns do not require descriptors. This is significant because computing the SIFT descriptors, which is the most difficult part to optimize, takes about 1ms on a MacBook Pro laptop without including the time required to convolve the image. By contrast, Ferns take $13.5\ 10^{-3}$ milliseconds to classify one keypoint into 200 classes on the same machine. Moreover, ferns still run nicely with a primitive keypoint extractor, such as the one we used in our experiments. When 300 keypoints are extracted and matched against 200 classes, our implementation on the MacBook Pro laptop requires 20ms per frame for both keypoint extraction and recognition in 640×480 images, and four fifths of this time are devoted to keypoint extraction. This corresponds to a theoretical 50Hz frame rate if one does ignore the time required for frame acquisition. Training takes less than five minutes.

Of course, the ability to classify keypoints fast and work with a simple keypoint detector comes at the cost of requiring a training stage, which is usually off-line. By contrast, SIFT does not require training and for some applications, this is clearly an advantage. However for other applications Ferns offer greater flexibility by allowing us to precisely state the kind of invariance we require through the choice of the training samples. Ferns also let us incrementally update the classifiers as more training samples become available as demonstrated by the SLAM application of Section 2.5.4. This flexibility is key to the ability to carry out offline computations and significantly simplify and speed-up the run-time operation.

2.5.2 Ferns vs SIFT to detect 3D objects

So far we have considered that the keypoints lie on a planar object and evaluated the robustness of Ferns with respect to perspective effects. This simplifies training as a single view is sufficient and the known 2D geometry can be used to compute ground truth correspondences. However most objects have truly three dimensional appearance, which implies that self occlusions and complex illuminations effects have to be taken into account to correctly evaluate the performance of any keypoint matching algorithm.

Recently, an extensive comparison of different keypoint detection and matching algorithms on a large database of 3D objects has been published [154]. It was performed on images taken by a stereo camera pair of objects rotating on a turntable. Fig. 2.12 shows such images spanning a 70° camera rotation range. We used this image database to evaluate the performance of Ferns for a variety of 3D objects. We compare our results against the

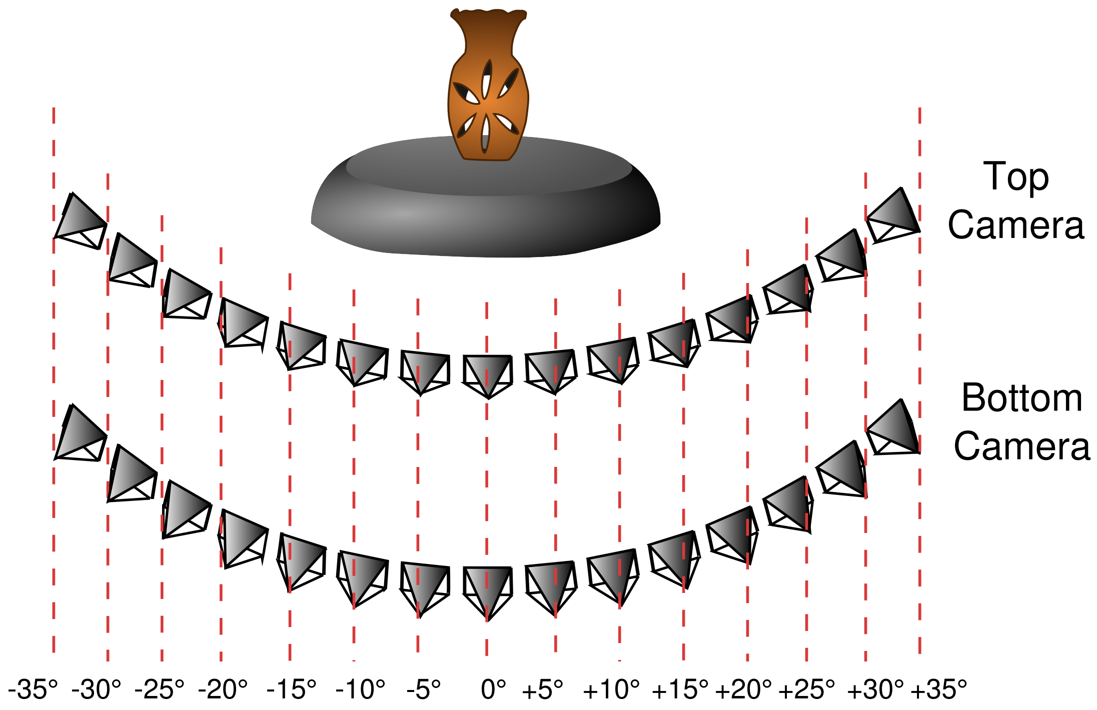


Figure 2.13: Generating ground truth data for 3D object detection. Each test object contains two sequences of images taken by the *Top* and *Bottom* cameras while the object rotates on the turntable. The camera geometry has been calibrated using a checkerboard calibration pattern. We use 15 consecutive camera views for evaluation purposes, because it is easy to obtain high quality calibration for this range of rotation.

SIFT detector/descriptor pair which has been found to perform very well on this database. The keypoints and the descriptors are computed using the same software as before.

As in [130], we obtained the ground truth by using purely geometric methods, which is possible because the cameras and the turn table are calibrated. The initial correspondences are obtained by using the trifocal geometry between the top/bottom cameras in the center view and every other camera as illustrated by Fig. 2.13. We then reconstruct the 3D points for each such correspondence in the bottom/center camera coordinate frame and use these to form the initial tracks that span the $-35\,°/+35\,°$ rotation range around a central view. Since the database images are separated by $5\,°$, the tracks span 15 images each for the top and bottom cameras. We eliminate very short tracks and remaining tracks are extended by projecting the 3D point to each image and searching for a keypoint in the vicinity of the projection. Finally to increase robustness against spurious tracks formed by outliers, we eliminate tracks covering less then 30% of the views and the remaining tracks form the ground truth for the evaluation, which is almost free of outliers. Sample ground truth data is depicted by Fig. 2.14, which shows the complex variations in patch appearance induced by the 3D structure of the objects.

The training is done using views separated by $10\,°$, skipping every other frame in the ground truth data. We then use the views we skipped for testing purposes.. This geometry based sampling is shown in Fig. 2.14. Sampling based on geometry creates uneven number of training and test samples for different keypoints as there are gaps in the tracks. The sampling could have been done differently to balance the number of test and training samples for each keypoint. However our approach to sampling closely mimics what happens in practice when training data comes from sparse views and the classifier must account for unequal numbers of training samples.

We train the Ferns in virtually the same way we do in the planar case. Each track is assigned a class number and the training images are deformed by applying random affine deformations. We then use all of them to estimate the probability distributions, as discussed in Section 2.3. 1000 random affine deformations per training image are used to train 50 Ferns of size 11. Ferns classify the test patches by selecting the track with the maximum probability. For SIFT, each test example is classified by selecting the track number of the keypoint in the training set with the nearest SIFT descriptor.

In our tests, we learn the appearance and the geometry of a 3D object from several views and then detect it in new ones by matching keypoints. Hence the learned geometry can be used to eliminate outliers while estimating

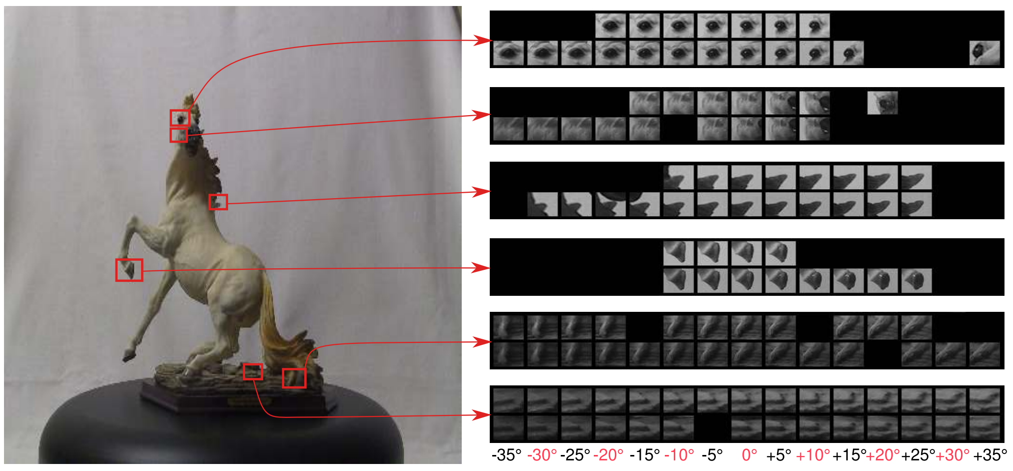


Figure 2.14: Samples from the ground truth for the *Horse* dataset. The image on the left shows the center view. The six keypoint tracks on the right show the content variation for each patch as the camera center rotates around the turntable center. Each track contains two lines that correspond to the *Top* and *Bottom* cameras, respectively. Black areas denote frames for which the keypoint detector did not respond. The views produced by a rotation that is a multiple of 10 ° are used for training and are denoted by red labels. The others are used for testing.

the camera pose using the P3P algorithm [73] together with a robust matching strategy such as RANSAC [67]. Unlike [154], we therefore do not use the ratio test on descriptor distances or a similar heuristics to reject matches, as this might reject correct correspondences. The additional computational burden can easily be overcome by using the classification score for RANSAC sampling as presented by [44].

We compare the recognition rates of both methods on objects with different kinds of texture. Fig. 2.15 shows the recognition rate on each test image together with the average over all frames. The Ferns perform as well as nearest neighbor matching with SIFT for a whole range of objects with very different textures.

Note that, when using Ferns, there is almost no run-time speed penalty for using multiple frames, since as more training frames are added we can increase the size of our Ferns. As discussed in Section 2.4 this requires more memory but does not slow down the algorithm in any meaningful way. By contrast, using more frames for nearest neighbor SIFT matching linearly slows down the matching, although a clever and approximate implementation might mitigate the problem.

In theory it should be possible to improve the performance of SIFT-based approach by replacing nearest neighbor matching with a more sophisticated technique such as K-Nearest Neighbors with voting. However, this would further slow down the algorithm. Our purpose is to show that the Fern based classifier can naturally integrate data from multiple images without the need for a more complex training phase or any handicap in the run-time performance, reaching the performance of standard SIFT matching.

2.5.3 Panorama and 3D Scene Annotation

With the recent proliferation of mobile devices with significant processing power, there has been a surge of interest in building real-world applications that can automatically annotate the photos and provide useful information about places of interest. These applications test keypoint matching algorithms to their limits because they must operate under constantly changing lighting conditions and potentially changing scene texture, both of which reduce the number of reliable keypoints. We have tested Ferns on two such applications, annotation of panorama scenes and parts of a historical building with 3–D structure. Both applications run smoothly at frame rate using a standard laptop and an of the shelf web camera. By applying standard optimizations for embedded hardware, we have ported this implementation onto a mobile device that runs at a few frames per second. More recently, Ferns have been successfully integrated into commercially available mobile phones to run at frame rates by taking into account specific limitations of the hardware and integrating detection with frame-to-frame tracking [238].

For the panorama application, we trained Ferns using an annotated panorama image stitched from multiple images. At run-time given an input image and after having established correspondences between the panorama

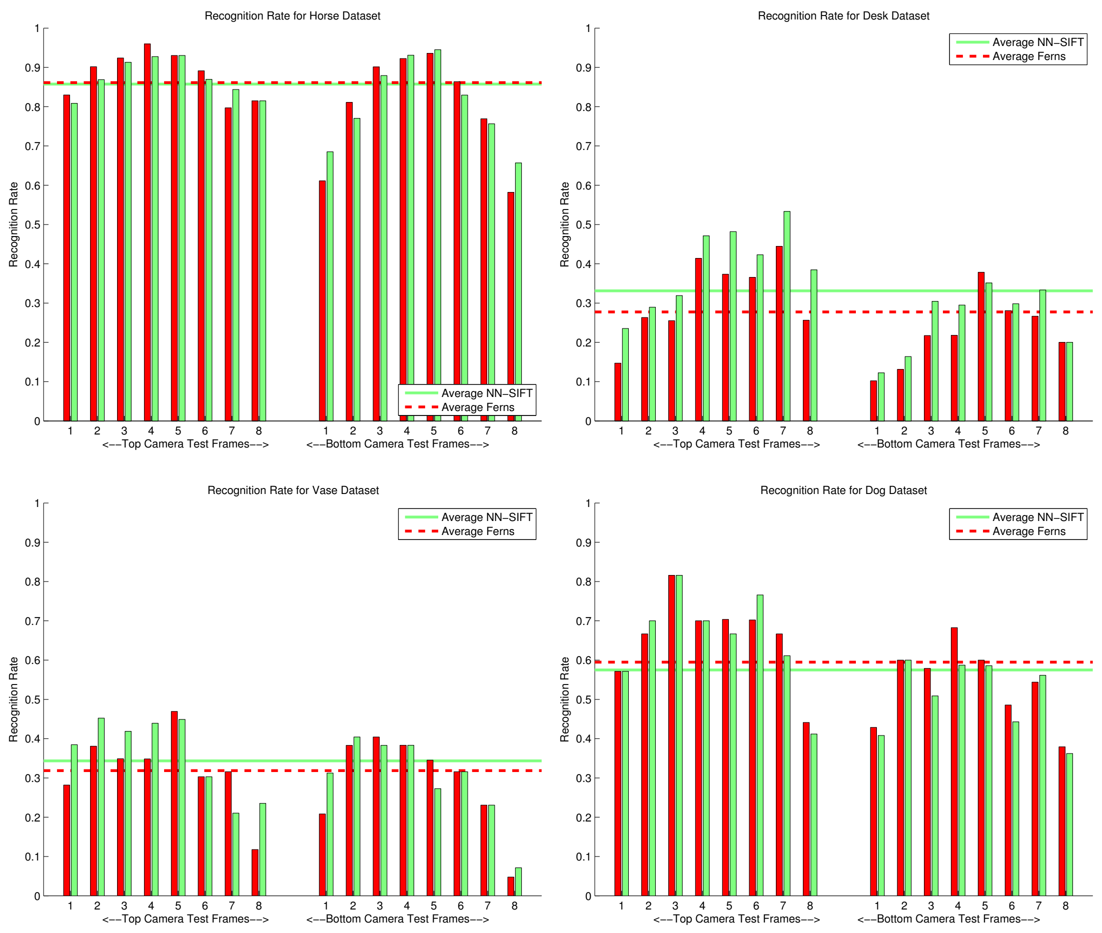


Figure 2.15: Recognition rates for 3D objects. Each pair of bars correspond to a test frame. The red bar on the left represents the rate for Ferns and the light green bar on the right the rate for Nearest Neighbor SIFT matching. The weighted averages over all frames also appear as dashed line for Ferns and solid line for NN-SIFT. The weights we use are the number of keypoints per frame.

and the test image, we compute a 2–D homography and use it to eliminate outliers and to transfer the annotation from the training image to the input image as shown in Fig. 2.16. We successfully run a number of tests under different weather conditions and different times of day.

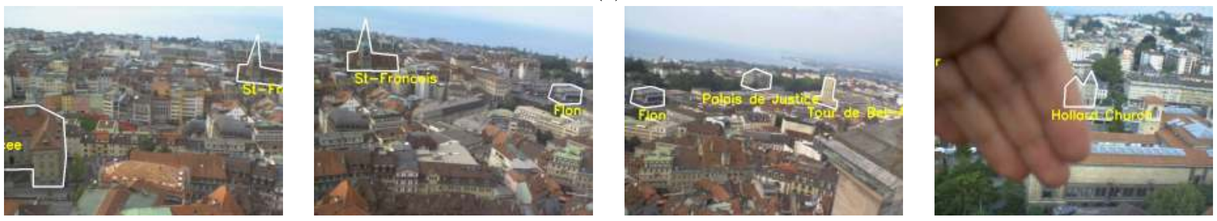
Annotating a 3–D object requires training using multiple images from different viewpoints, which is easy to do in our framework as discussed in the previous subsection. We also built a 3–D model for the object using standard structure from motion algorithms to register the training images followed by dense reconstruction [209, 208]. The resulting fine mesh is too detailed to be used so it is approximated by a coarse one containing much less detail. Despite its rough structure, this 3–D model allows annotation of important parts of the object and the correct reprojection of this information onto images taken from arbitrary viewpoints as depicted by Fig. 2.16.

2.5.4 SLAM using Ferns

In this section we demonstrate that Ferns can increase the robustness of a Visual SLAM system. Their role is twofold. We first use them to bootstrap the system by localizing the camera with respect to a known planar pattern in the scene. Second, we incrementally train a second set of Ferns to recognize new landmarks reconstructed by the system. They make the system robust against severe disturbances such as complete occlusion or strong shaking of the camera, as evidenced by the smoothness of the recovered camera trajectory in Fig. 2.18 (a).

For simplicity, we use a FastSLAM [151, 152] approach with a single particle to model the distribution over the camera trajectory. This is therefore a simplified version of FastSLAM, but the Ferns are sufficiently powerful to make it robust.

(a)



(b)

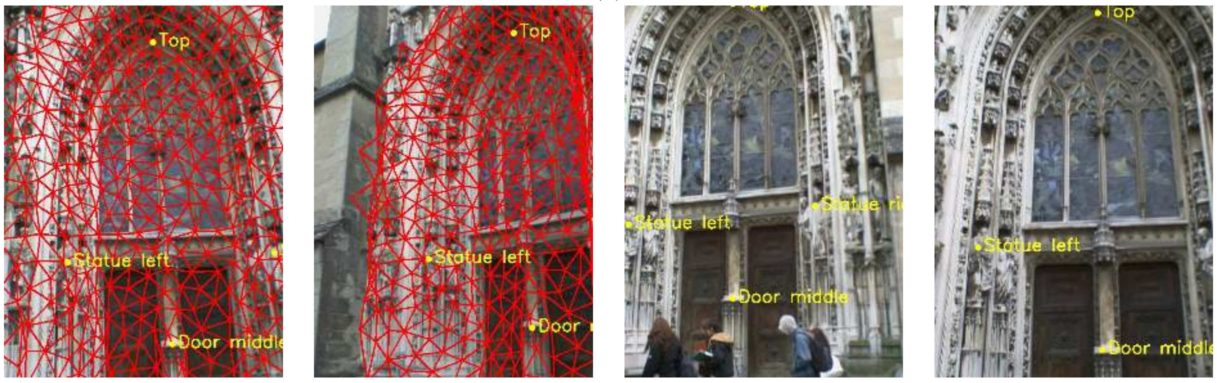


Figure 2.16: Automated image annotation. (a) We match an input image against a panorama. Despite occlusions, changes in weather conditions and lighting, the Ferns return enough matches for reliable annotation of major landmarks in the city. (b) We match an image to an annotated 3–D model, overlaid on the two leftmost images. This lets us display annotations at the right place.

As discussed above, we use two different sets of Ferns. We will refer to the first set as “Offline Ferns”, that we trained offline to recognize keypoints on the planar pattern. As shown on Fig. 2.17 (a), we make sure it is visible in the first frame of the sequence to bootstrap the system and replace the four fiducial markers used by many other systems. This increases the flexibility of our system since we can use any pattern we want provided that it is textured enough. The second set of Ferns, the “Online Ferns”, are incrementally trained to recognize the 3D landmarks the SLAM system discovers and reconstructs.

The incremental training takes place over several frames, typically between 20 and 50, where the corresponding landmark was matched. For each, we add a small number of random views of the observed patch to the Ferns. In the case where a landmark is observed for the first time, this number is substantially higher, around 100. The comparatively small number of views in the beginning works because motion between frames are relatively small and by the time the viewing angle has changed significantly the training will be complete. Of course, not all of the patches need to be re-trained, but only the new ones. This is achieved by undoing the normalization step, adding the new observations and then re-normalizing the posteriors. This incremental training is computationally costly and future research will aim at reducing the cost of training [32].

Our algorithm goes through the following steps:

1) Initialize the camera pose and some landmarks by detecting a known pattern using the Offline Ferns.

2) Detect keypoints and match them using the Offline and Online Ferns against the known landmarks.

3) Estimate the camera pose from these correspondences using a P3P algorithm and RANSAC [67]. The estimated pose is refined via a non-linear optimization.

4) Refine the location estimates of the inlier landmarks using an Extended Kalman filter.

5) Create new landmarks. Choose a number of detected keypoints that *do not* belong to any landmark in the map and initialize the new landmarks with a large uncertainty along the line of sight and a much smaller uncertainty in the camera’s lateral directions.

6) Retrain Ferns with good matches from 2) and the new landmarks.

7) Loop to step 2.

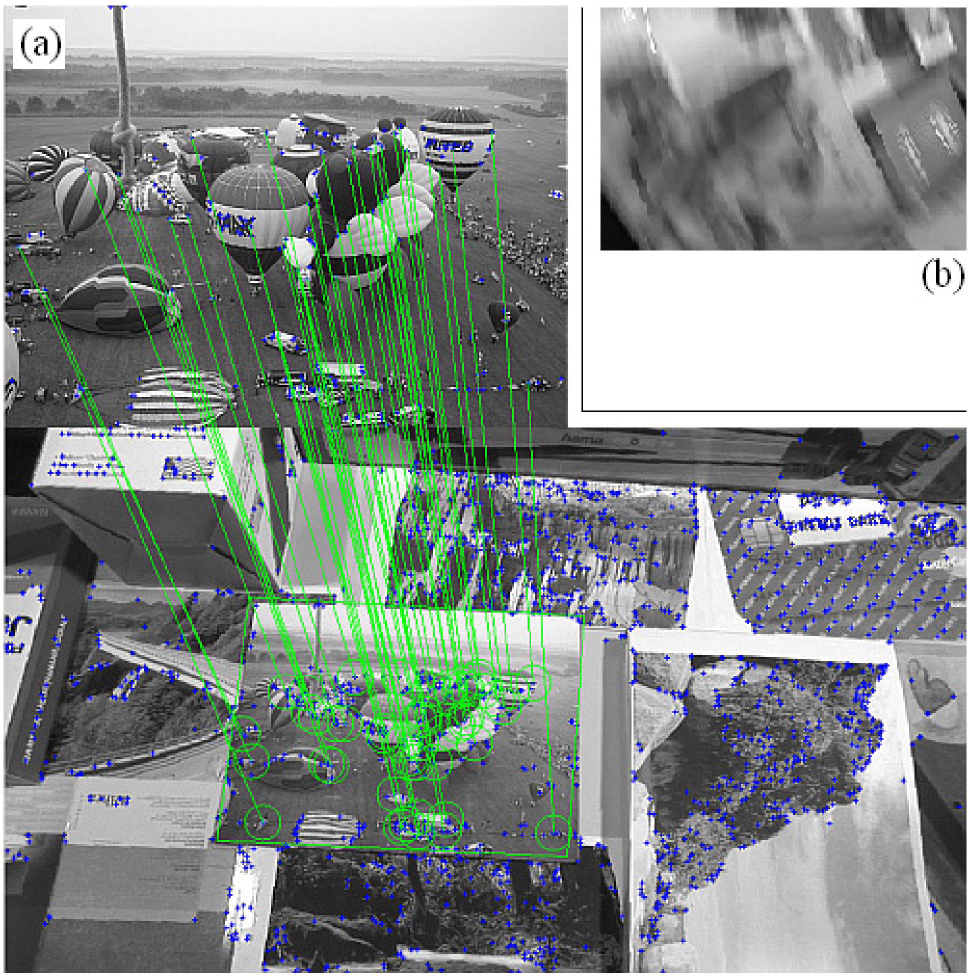


Figure 2.17: Ferns applied to SLAM. (a) Initialization: The pattern at the top is detected in the image yielding initial landmarks and pose. (b) A later image from the sequence, which shows both strong camera shaking and motion blur. When the image is that bad no correspondences can be established and the tracking is lost. Nevertheless, the system automatically recovers when the image quality improves thanks to the Ferns.

With this system we demonstrate that both smooth tracking and recovery from complete failure can be naturally integrated by employing Ferns for the matching task.

The reconstructed trajectory in Fig. 2.18 shows only tiny jags in the order of a few millimeters and appears as smooth as a trajectory that was estimated in a filtered approach to SLAM, such as MonoSLAM [49, 48]. This is especially noteworthy as the camera's state is re-estimated from scratch *in every frame* and there is no such thing as a motion model.[1] At the same time, this is a strong indication for an overall correct operation, since an incorrect map induces an unstable state estimation and vice versa. In total the system mapped 724 landmarks and ran stable over all 2026 frames of the sequence.

Recently, [249] presented a system that is also capable of recovering from complete failure. They achieved robustness with a hybrid combination between template matching and a modified version of Randomized Trees. However, their map typically contains one order of magnitude fewer landmarks and there has been no indication that the modified Trees will still be capable of handling a larger number of interest-points.

We also validated our system quantitatively. First, we checked the relative accuracy for reconstructed pairs of 3D points and we found an error between 3.5 to 8.8% on their Euclidean distances. Second, the absolute accuracy was assessed by choosing two world planes parallel to the ground plane on top of two boxes in the scene on which some points were reconstructed. Their z-coordinates deviated on average 7–10 mm. Given that the camera is at roughly 0.6 to 1.4 m from the points under consideration, this represents very good accuracy.

[1]Other systems commonly use a motion model to predict the feature location in the next frame and accordingly restrict the search area for template matching.

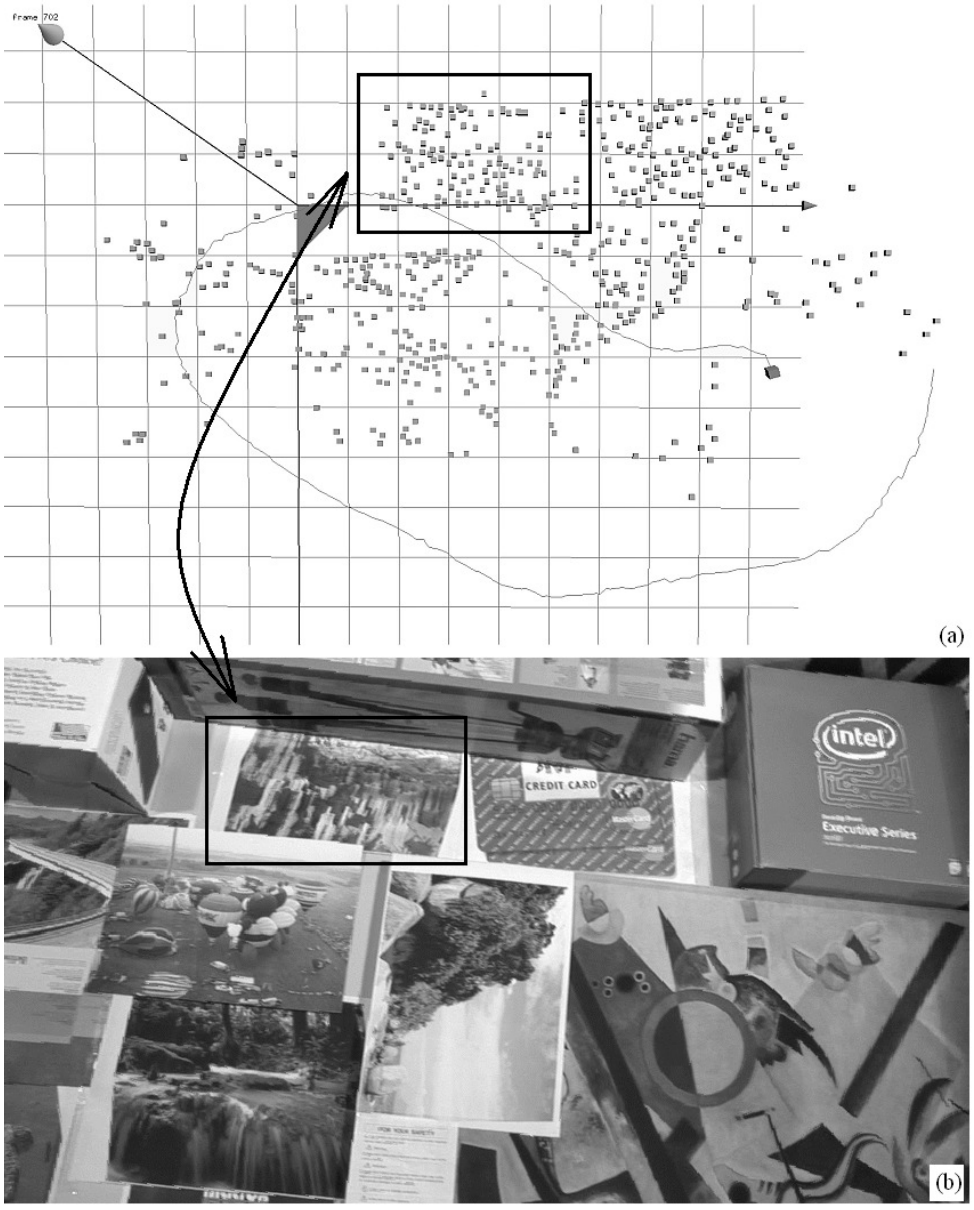


Figure 2.18: Ferns applied to SLAM. (a) View from top on the xy-plane of the 3D space in which the trajectory and landmarks are reconstructed. The landmarks are shown as cubes centered around the current mean value of the corresponding Kalman filter. Note how smooth the trajectory is, given the difficulties the algorithm had to face. (b) Qualitative visual comparison: One group of 3D points and the corresponding group of 2D points is marked. Comparing this group to the surroundings confirmes a basic structural correctness of the reconstruction.

2.6 Conclusion

We have presented a powerful method for image patch recognition that performs well even in the presence of severe perspective distortion. The "semi-naive" structure of ferns yields a scalable, simple, and fast implementation to what is one of the most critical step in many Computer Vision tasks. Furthermore the Ferns naturally allow trade offs between computational complexity and discriminative power. As computers become more powerful, we can add more Ferns to improve the performance. Conversely, we can adapt them to low computational power such those on hand-held systems by reducing the number of ferns. This has actually been done in recent work [238] to achieve real-time performance on a mobile phone.

A key component of our approach is the Naive-Bayesian combination of classifiers that clearly outperforms the averaging of probabilities we used in earlier work [130]. To the best of our knowledge, a clear theoretical argument motivating the superiority of Naive-Bayesian (NB) techniques does not exist.

There is however strong empirical evidence that they are effective in our specific case. This can be attributed to two different causes. First, unlike mixture models, the product models can represent much sharper distributions [99]. Indeed, when averaging is used to combine distributions, the resulting mixture has higher variance than the individual components. In other words, if a single Fern strongly rejects a keypoint class, it can counter the combined effect of all the other Ferns that gives a weak positive response. This increases the necessity of larger amounts of training data and the help of a prior regularization term as discussed in Section 2.3. Second, the classification task, which just picks a single class, will not be adversely affected by the approximation errors in the joint distribution as long as the maximum probability is assigned to the correct class [71, 56]. We have shown that such a naive combination strategy is a worthwhile alternative when the specific problem is not overly sensitive to the implied independence assumptions.

Since the publication of the paper corresponding to this chapter [168], several works used ferns or took inspiration from them for other problems. This includes image classification [25], face alignment [38], object detection and tracking [118], object pose recovery [53], deformable object tracking [76], interactive learning for object detection [233], object category 3D detection [234], action recognition [167], specular textureless object detection [177], character detection in the wild [242], hand pose recognition [123], and even gene selection [125].

Even if we did not realize it when we developed the ferns, they are actually closely related to Local Sensitive Hashing (LSH) [10]. LSH is a very efficient method for approximate nearest neighbor search in a set of data vectors. It applies random projections and thresholding to the query and data vectors to compute binary strings that are called “hash keys”. During a pre-processing stage, the data vectors are assigned to different “buckets”, which are arrays that contain the data vectors with the same hash key value. Given a query, LSH computes the values of the hash keys for it, and restricts the search for nearest neighbors to the data vectors in the corresponding buckets. While there is no guarantee that the nearest neighbor will always be found by LSH, it works remarkably well and it is very efficient.

Our groups of binary features clearly correspond to the hash keys of LSH. Our binary features can even be written as a projection and thresholding operation (with +1 and -1 weights applied to two coordinates and a threshold of 0). Our synthetic training samples correspond to the data vectors, and instead of arranging the data vectors into buckets, we store the probabilities for the all the classes of samples. The advantage of ferns compared to LSH is that we compute a probability for each class. The drawback is that the complexity of ferns in terms of memory and computation times grows linearly with the number of classes. Even if ferns are very efficient, that prevents to use them for problems with more than a thousand classes.

This is because of this limit that we turned back to descriptors for the PhD thesis of Michael Calonder. We wanted however to keep somehow the simplicity and efficiency of the ferns. We described the resulting approach in the next chapter.

Chapter 3

BRIEF: Computing a Local Binary Descriptor Very Fast

3.1 Introduction

As explained at the end of the previous chapter, matching feature points with the ferns classifier is very efficient as long as the number of reference points is not higher than a few thousands. Above, because of the linear complexity in terms of speed and memory, the ferns are not well adapted anymore. Moreover, they require a learning stage, which is not possible for all applications.

For the PhD thesis of Michael Calonder, we therefore wanted to develop a scalable method. Local descriptors are a scalable solution, because the best approximate nearest neighbors methods have a very low complexity. This is because of this limit that with Michael, we turned back to local descriptors. However, existing local descriptors were mostly hand-designed, and we wanted to introduce in the field the efficiency and simplicity of the ferns.

One obvious way to speed up matching and reduce memory consumption when working with local descriptors is to consider short descriptors. They can be achieved by applying dimensionality reduction, such as PCA [149] or LDA [104], to an original descriptor such as SIFT [140] or SURF [17]. For example, it was shown in [229, 251, 33] that floating point values of the descriptor vector could be quantized using very few bits per number without performance loss. An even more drastic dimensionality reduction can be achieved by using hash functions that reduce SIFT descriptors to binary strings, as done in [192, 207] These strings represent binary descriptors whose similarity can be measured by the Hamming distance.

However, these approaches to dimensionality reduction require first computing the full descriptor before further processing can take place, and this is time consuming especially for real-time applications. Moreover, descriptors such as SIFT and SURF are hand-crafted and were difficult to design. With BRIEF, we showed that this whole computation can be avoided by *directly* computing binary strings from image patches. The individual bits are obtained by comparing the intensities of pairs of points, as for the ferns in the previous chapter, but without requiring a training phase.

BRIEF is not based on Machine Learning per se. However, it was inspired by our previous work: Strings of random binary features proved to be powerful for classification, and it was logical to expect that longer strings would make a reliable descriptor. It also makes it very easy to implement, and its only meta-parameter is the length of the string. Moreover, it motivated our next work on descriptor learning, which is presented in the next chapter.

Our experiments show that only 256 bits, or even 128 bits, often suffice to obtain very good matching results. BRIEF is therefore very efficient both to compute and to store in memory. Furthermore, comparing strings can be done by computing the Hamming distance, which can be done extremely fast on modern CPUs that often provide a specific instruction to perform a XOR or bit count operation, as is the case in the latest SSE [109] and NEON [12] instruction sets.

This means that BRIEF easily outperforms other fast descriptors such as SURF and U-SURF in terms of speed, as will be shown in the Results section. Furthermore, it also outperforms them in terms of recognition rate in many cases, as we will demonstrate on well known benchmark datasets.

It is fair to say that BRIEF started a popular trend, and several authors proposed their own binary descriptors. We discuss these other descriptors in the conclusion to this chapter.

3.2 Related Work

The SIFT descriptor [140] is highly discriminant but, being a 128-vector, relatively slow to compute and match. This substantially affects real-time applications such as SLAM that keep track of many points as well as algorithms that require to store very large numbers of descriptors, for example for large-scale 3D reconstruction purposes.

This has been addressed by developing descriptors such as SURF [17] that are faster to compute and match while preserving the discriminative power of SIFT. Like SIFT, SURF relies on local gradient histograms but uses integral images to speed up the computation. Different parameter settings are possible but, since a vector of 64 dimensions already yields good recognition performance, that version has become a *de facto* standard. In the Results section, we will therefore compare BRIEF to both SIFT and SURF.

SURF is faster than SIFT but, since the descriptor is a 64-vector of floating points values, its memory footprint is still 256 bytes. This becomes significant when millions of descriptors need to be stored. There are three main classes of approaches towards reducing this number.

The first one involves dimensionality reduction techniques such as Principal Component Analysis (PCA) or Linear Discriminant Embedding (LDE). PCA is very easy to use and can reduce the descriptor size at no loss in recognition performance [149]. By contrast, LDE requires labeled training data, in the form of descriptors that should be matched together, which is more difficult to obtain. It can improve performance [104] but can also overfit and degrade performance.

Another way to shorten a descriptor is to quantize its floating-point coordinates into integers coded on fewer bits. In [229], it is shown that the SIFT descriptor could be quantized using only 4 bits per coordinate. The same method could most probably be applied to SURF as well, and makes these approaches competitive with BRIEF in terms of memory need. Quantization is a simple operation that not only yields a memory gain but also faster matching because the distance between short vectors can then be computed very efficiently on modern CPUs. In [30], it is shown that for appropriate parameter settings of the DAISY descriptor [221], PCA and quantization can be combined to reduce its size to 60 bits. Quantization can also be achieved by matching the descriptors to a small number of pre-computed centroids as done in source coding to obtain very good results [113]. However, the Hamming distance cannot be used for matching after quantization because the bits cannot be processed independently, which is something that does not happen with BRIEF.

A third and more radical way to shorten a descriptor is to binarize it. For example, [192] drew its inspiration from Locality Sensitive Hashing (LSH) [108] to turn floating-point vectors into binary strings. This is done by thresholding the vectors after multiplication by an appropriate matrix. Similarity between descriptors is then measured by the Hamming distance between the corresponding binary strings. This is very fast because the Hamming distance can be computed very efficiently via a bitwise XOR operation followed by a bit count. The same algorithm was applied to the GIST descriptor to obtain a binary description of an entire image [222]. A similar binarization method was also used in [112] with a random rotation matrix on quantized SIFT vectors to speed up matching within Voronoi cells. We also developed a method to estimate a matrix and a set of thresholds that optimize the matching performances when applied to SIFT vectors [207]. Another way to binarize the GIST descriptor is to use non-linear Neighborhood Component Analysis [222, 184], which seems more powerful but probably slower at run-time.

While all three classes of shortening techniques provide satisfactory results, relying on them remains inefficient in the sense that first computing a long descriptor then shortening it involves a substantial amount of time-consuming computation. By contrast, the approach we advocate in this chapter directly builds short descriptors by comparing the intensities of pairs of points without ever creating a long one. Such intensity comparisons were used in [168] for classification purposes and were shown to be very powerful in spite of their extreme simplicity. Nevertheless, the present approach is very different from [168] and [219] because it does *not* involve any form of online or offline training.

Finally, BRIEF echoes some of the ideas that were proposed in two older methods. The first one is the *Census transform* [254] that was designed to produce a descriptor robust to illumination changes. This descriptor is non-parametric and local to some neighborhood around a given pixel and essentially consists of a pre-defined set of pixel intensity comparisons in a local neighborhood to a central pixel, which produces a sequence of binary values. The bit string is then directly used for matching. Based on this idea, a number of variants have been introduced such as [72, 206], for example.

The second method, developed by Ojala *et al.* [164] and called Locally Binary Patterns (LBP), builds a Census-like bit string where the neighborhoods are taken to be circles of fixed radius. Unlike Census, however, LBPs

	Rotational invariance	Scale invariance
SURF	good (sliding orientation window)	good (scale space search)
U-SURF	limited	good (scale space search)
SIFT	good (orientational HoG)	good (scale space search)
U-SIFT	limited	good (scale space search)
U-BRIEF	limited	fairly good
O-BRIEF	good (using external information)	fairly good
S-BRIEF	limited	good (using external information)
D-BRIEF	very good (template database)	good (template database)

Table 3.1: Invariance properties of SURF, SIFT, and the four different BRIEF variants.

usually translate the bit strings into its decimal representation and build an histogram of these decimal values. The concatenation of these histogram values have been found to result in stable descriptors. Extensions are numerous and include [239, 92, 256, 28, 146, 156].

3.3 Method

Our approach is inspired by earlier work [168] that showed that image patches could be effectively classified on the basis of a relatively small number of pairwise intensity comparisons. The results of these tests were used to train either randomized classification trees [130] or a Naive Bayesian classifier [168] to recognize patches seen from different viewpoints. Here, we do away with both the classifier and the trees, and simply create a bit string out of the test responses, which we compute after having smoothed the image patch.

More specifically, we define test τ on patch $\mathbf{p}$ of size $S \times S$ as

$$\tau(\mathbf{p};\mathbf{x},\mathbf{y}) := \begin{cases} 1 & \text{if } I(\mathbf{p},\mathbf{x}) < I(\mathbf{p},\mathbf{y}) \\ 0 & \text{otherwise} \end{cases} , \tag{3.1}$$

where $I(\mathbf{p},\mathbf{x})$ is the pixel intensity in a smoothed version of $\mathbf{p}$ at $\mathbf{x} = (u, v)^\top$. Choosing a set of n_d $(\mathbf{x},\mathbf{y})$-location pairs uniquely defines a set of binary tests. We take our BRIEF descriptor to be the n_d-dimensional bit string that corresponds to the decimal counterpart of

$$\sum_{1 \le i \le n_d} 2^{i-1}\, \tau(\mathbf{p};\mathbf{x}_i,\mathbf{y}_i) \ . \tag{3.2}$$

In this work we consider $n_d = 128$, 256, and 512 and will show in the Results section that these yield good compromises between speed, space, and accuracy.

The above procedure corresponds to the first BRIEF variant, which we call *upright* BRIEF (U-BRIEF). Naturally and by design, U-BRIEF has limited invariance to in-plane rotation which we will quantify later. However, if an orientation estimate for the feature point at hand is available, the tests can be pre-rotated accordingly and hence made rotation invariant. We refer to this as *oriented* BRIEF (O-BRIEF). In practice, the orientation can be estimated by a feature point detector or, when using a mobile device to capture the image, by the device's gravity sensor. Likewise, we can use the scale information provided by the feature detector to grow or shrink the tests accordingly, yielding the *scaled* BRIEF (S-BRIEF) variant of BRIEF. In the absence of externally supplied scale and orientation data, O-BRIEF and S-BRIEF are mostly of academic interest since they would have to rely on a potentially slow feature detector to provide the necessary information. Such is not the case of the fourth and final BRIEF variant, which we refer to as database BRIEF (D-BRIEF). The idea behind D-BRIEF is to achieve full invariance to rotation and scaling by building a database of templates, which are the U-BRIEF descriptors for rotated versions of patches to be recognized. This needs to be done only once and can be done in real-time. Matching a new patch against a database then means matching against all rotated and scaled versions. However, because computing the distance between binary strings is extremely fast, this remains much faster than using either SIFT or SURF and we demonstrate the viability of this approach in an application in section 3.5.5, running at 25 Hz or more. We summarize the different BRIEF variants and its competitors in table 3.1.

In the remainder of the chapter, we will add a postfix to BRIEF, BRIEF-k, where $k = n_d/8$ represents the number of *bytes* required to store the descriptor.

When creating such descriptors, the only choices that have to be made are those of the kernels used to smooth the patches before intensity differencing and the spatial arrangement of the $(\mathbf{x}, \mathbf{y})$-pairs. We discuss these in the remainder of this section.

To this end, we use the Wall dataset that we will describe in more detail in Section 3.5. It contains five image pairs, with the first image being the same in all pairs and the second image shot from a monotonically growing baseline, which makes matching increasingly more difficult. To compare the pertinence of the various potential choices, we use as a quality measure the *recognition rate* that will be precisely defined at the beginning of section 3.5. In short, for both images of an image pair that is to be matched and for a given number of corresponding keypoints between them, it quantifies how often the correct match can be established. In the case of BRIEF, the nearest neighbor is identified using the Hamming distance measure. The recognition rate can be computed reliably because the scene is planar and the homography between images is known and can therefore be used to check whether points truly correspond to each other or not.

3.3.1 Smoothing Kernels

By construction, the tests of Eq. 3.1 take only the information at single pixels into account and are therefore very noise-sensitive. By pre-smoothing the patch, this sensitivity can be reduced, thus increasing the stability and repeatability of the descriptors. It is for the same reason that images need to be smoothed before they can be meaningfully differentiated when looking for edges. This analogy applies because our intensity difference tests can be thought of as evaluating the sign of the derivatives within a patch.

Fig. 3.1 illustrates the effect on the recognition rate of increasing amounts of Gaussian smoothing. The variance of the Gaussian kernel has been varied in $[0, 2.75]$. The more difficult the matching, the more important smoothing becomes to achieving good performance. Furthermore, the recognition rates remain relatively constant in the 1 to 3 range and, in practice, we use a value of 2. For the corresponding discrete kernel window we found a size of 7×7 pixels be necessary and sufficient.

While the Gaussian kernel serves this purpose, its non-uniformity makes it fairly expensive to evaluate, compared to the much cheaper binary tests. We therefore experimented by a box filter. As can be seen in Fig. 3.1, no accuracy is lost when replacing the Gaussian filter by a box filter, which confirms a similar finding of [92]. The latter, however, is much faster to evaluate since integral images offer an effective way to computing box filter responses independently of the filter size using only three additions.

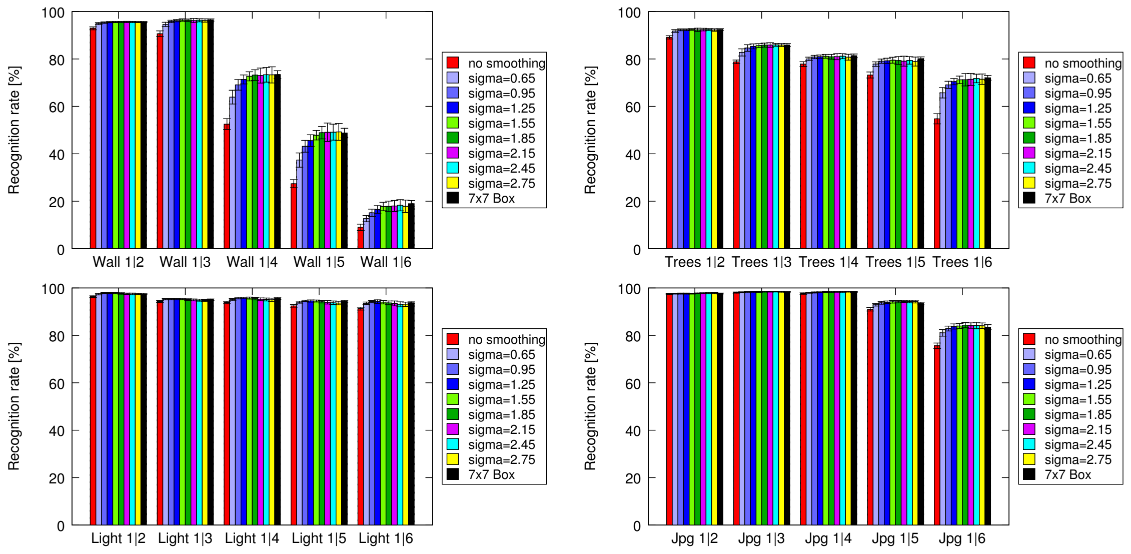


Figure 3.1: Recognition rates and their variances for different amounts of smoothing and different benchmark image sets. Each group of 10 bars represents the recognition rates in one specific image pair for increasing levels of smoothing. Especially for the hard-to-match pairs, which are those on the right side of the plot, smoothing is essential in slowing down the rate at which the recognition rate decays. Note that the first 9 bars corresponds to the Gaussian kernel where the rightmost bar of each group corresponds to using a box filter. It appears that using a box filter does not harm accuracy.

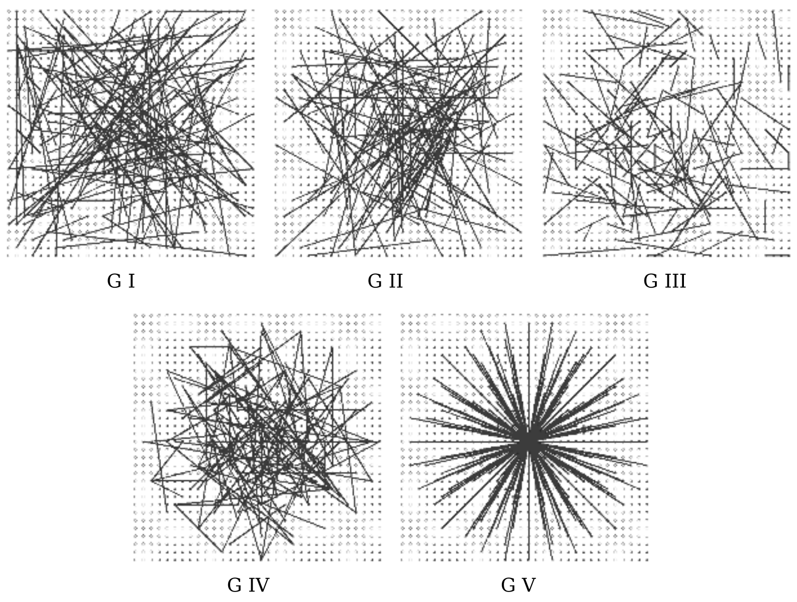


Figure 3.2: Different approaches to choosing the test locations. All except the righmost one are selected by random sampling. Showing 128 tests in every image.

3.3.2 Spatial Arrangement of the Binary Tests

Generating a length-n_d bit vector leaves many options for selecting the test locations $(\mathbf{x}_i, \mathbf{y}_i)$ of Eq. 3.1 in a patch of size $S \times S$. We experimented with the five sampling geometries depicted by Fig. 3.2. Assuming the origin of the patch coordinate system to be located at the patch center, they can be described as follows.

I) $(\mathbf{X}, \mathbf{Y}) \sim$ i.i.d. Uniform$(-\frac{S}{2}, \frac{S}{2})$: The $(\mathbf{x}_i, \mathbf{y}_i)$ locations are evenly distributed over the patch and tests can lie close to the patch border.

II) $(\mathbf{X}, \mathbf{Y}) \sim$ i.i.d. Gaussian$(0, \frac{1}{25}S^2)$: The tests are sampled from an isotropic Gaussian distribution. This is motivated by the fact that pixels at the patch center tend to be more stable under perspective distortion than those near the edges. Experimentally we found $\frac{S}{2} = \frac{5}{2}\sigma \Leftrightarrow \sigma^2 = \frac{1}{25}S^2$ to give best results in terms of recognition rate.

III) $\mathbf{X} \sim$ i.i.d. Gaussian$(0, \frac{1}{25}S^2)$, $\mathbf{Y} \sim$ i.i.d. Gaussian$(\mathbf{x}_i, \frac{1}{100}S^2)$: The sampling involves two steps. The first location $\mathbf{x}_i$ is sampled from a Gaussian centered around the origin, like the previous, while the second location is sampled from another Gaussian centered on $\mathbf{x}_i$. This forces the tests to be more local. Test locations outside the patch are clamped to the edge of the patch. Again, experimentally we found $\frac{S}{4} = \frac{5}{2}\sigma \Leftrightarrow \sigma^2 = \frac{1}{100}S^2$ for the second Gaussian performing best.

IV) The $(\mathbf{x}_i, \mathbf{y}_i)$ are randomly sampled from discrete locations of a coarse polar grid introducing a spatial quantization. This geometry is of interest because it reflects the behavior of BRIEF when the underlying tests are sampled from a more coarse grid than that of the pixels.

V) $\forall i : \mathbf{x}_i = (0, 0)^\top$ and $\mathbf{y}_i$ is takes all possible values on a coarse polar grid containing n_d points. Note that this geometry corresponds to that of the Census transform [254] discussed in section 3.2.

For each of these test geometries we compute the recognition rate on several benchmark image sets and show the results in Fig. 3.3.

Clearly, the symmetrical and regular G V strategy loses out against all random designs G I to G IV, with G II enjoying a small advantage over the other three in most cases. For this reason, in all further experiments presented in this chapter, it is the one we will use.

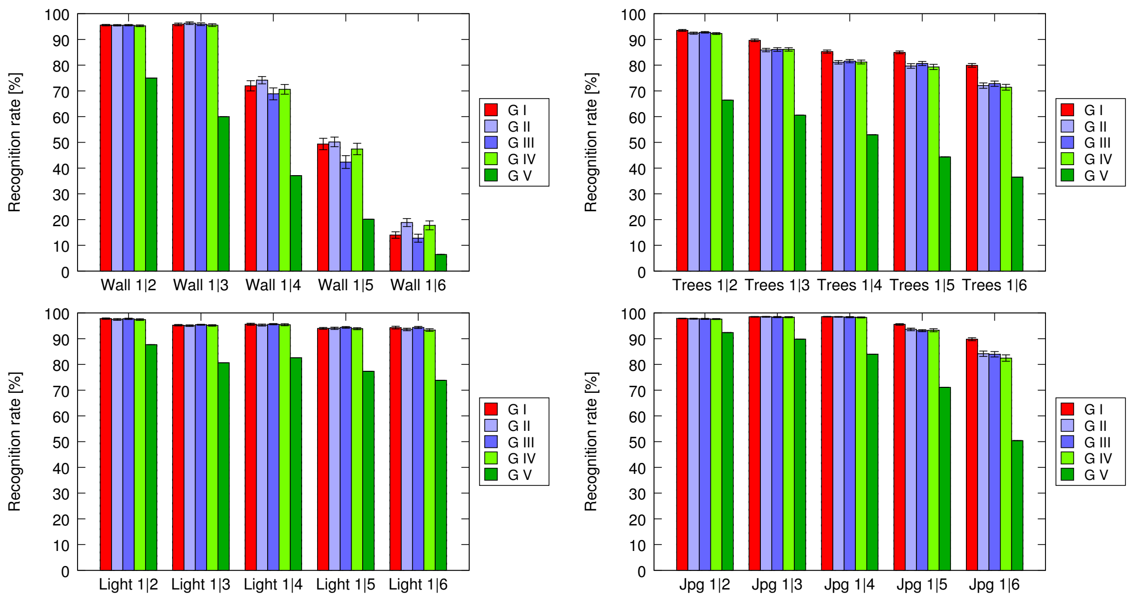


Figure 3.3: Recognition rates and their variances for the five different test geometries introduced in Section 3.3.2 and different benchmark image sets.

3.3.3 Distance Distributions

In this section, we take a closer look at the distribution of Hamming distances between our descriptors. To this end we extract about 4000 matching points from the five image pairs of the Wall sequence. For each image pair, Fig. 3.4 shows the normalized histograms, or distributions, of Hamming distances between corresponding points (in blue) and non-corresponding points (in red). The maximum possible Hamming distance being $32 \cdot 8 = 256$ bits, unsurprisingly, the distribution of distances for non-matching points is roughly Gaussian and centered around 128. As could also be expected, the blue curves are centered around a smaller value that increases with the baseline of the image pairs and, therefore, with the difficulty of the matching task.

Since establishing a match can be understood as classifying pairs of points as being a match or not, a classifier that relies on these Hamming distances will work best when their distributions are most separated. As we will see in section 3.5, this is of course what happens with recognition rates being higher in the first pairs of the Wall sequence than in the subsequent ones.

3.4 Analysis

In this section, we use again the Wall dataset of Fig. 3.10 to analyze the influence of the various design decisions we make when implementing the four variants of the BRIEF descriptor.

3.4.1 Descriptor Length

Since many practical problems involve matching a few hundred feature points, we first use $N = 512$ of them to compare the recognition rate of U-BRIEF using either 128, 256, or 512 tests, which we denote as U-BRIEF-16, U-BRIEF-32, and U-BRIEF-64. The main purpose here is to show the dependence of the recognition rate on the descriptor length which is why we only include the recognition rate of U-SURF-64 in the plots. Recall that U-SURF returns 64 floating point numbers requiring 256 bytes of storage–this is 4 to 16 times more than BRIEF.

In Fig. 3.5 we use Wall to plot the recognition rate as a function of the number of tests. We clearly see a saturation effect beyond 200 tests for the more easy cases and an improvement up to 512 for the others. The motivates the choice for the default descriptor BRIEF-32.

We would like to convince the reader that this behavior is *not* an artifact arising from the fairly low number of feature points used for testing and that BRIEF scales reasonably with the number of features to match. To this

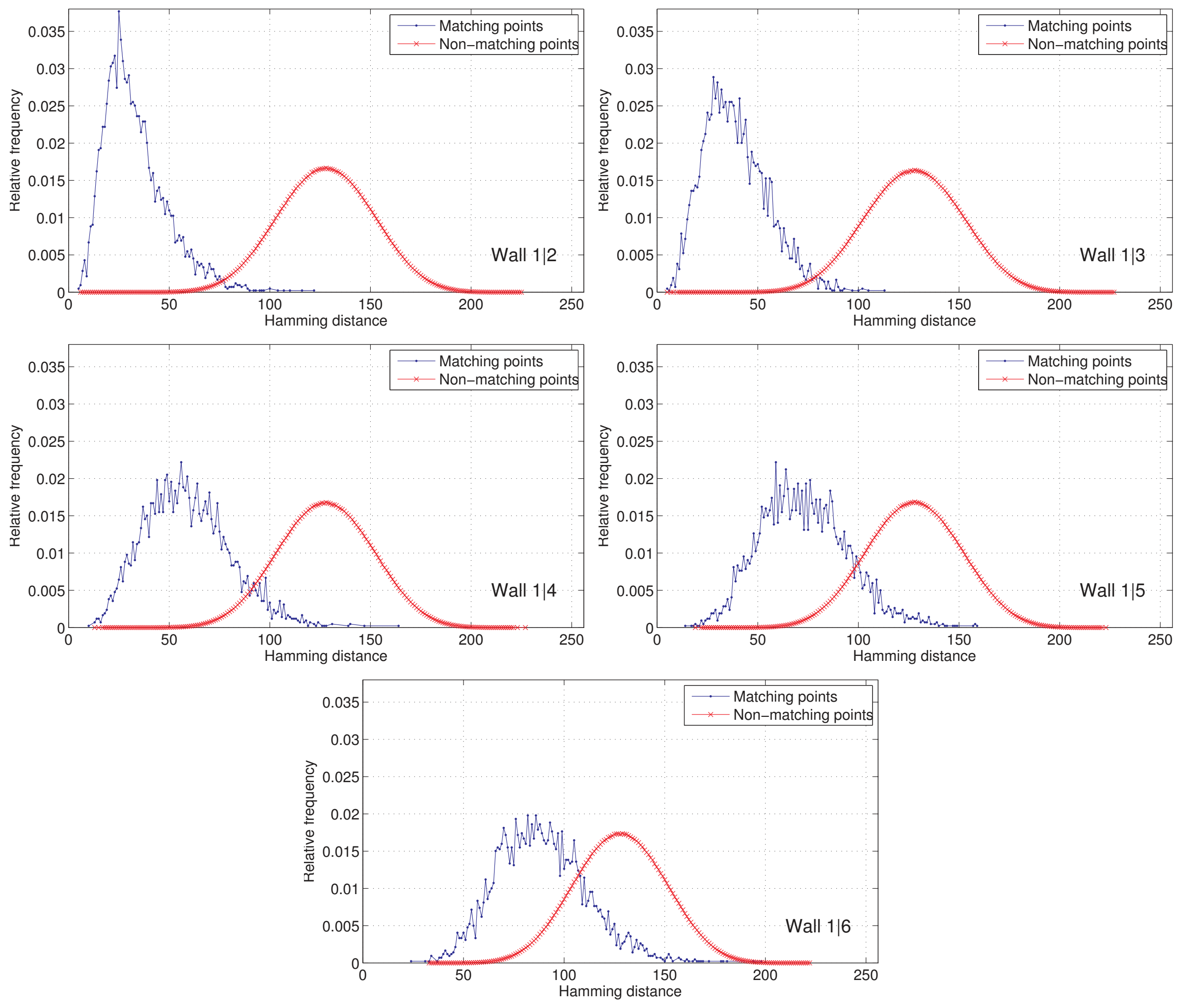


Figure 3.4: Distributions of Hamming distances for matching pairs of points (thin blue lines) and for non-matching pairs (thick red lines) in each of the five image pairs of the Wall dataset. They are most separated for the first image pairs, whose baseline is smaller, ultimately resulting in higher recognition rates.

end we repeat the testing for values of N ranging from 512 to 4096, adding new feature points by decreasing the detection threshold. We found similar recognition rates, as shown in in Fig. 3.6. Note how the rates drop with increasing N for *all* the descriptors, as expected; however, the rankings remain unchanged. This behavior changes when N becomes even larger, as discussed in Section 3.5.C.

3.4.2 Orientation Sensitivity

U-BRIEF is not designed to be rotationally invariant. Nevertheless, as shown by our results on the six test data sets, it tolerates small amounts of rotation. To quantify this tolerance, we take the first image of the Wall sequence with $N = 512$ points and match these against points in a rotated version of itself, where the rotation angle is varied between 0 and 180 degrees.

Figure 3.7 compares the recognition rate of SURF (●), three versions of BRIEF-32 (×, ◇, △), and U-SURF (○). Since the latter does not correct for orientation either, its behavior is very similar or even slightly worse than that of U-BRIEF: Up to 15 degrees there is little degradation, however, this is followed by a precipitous drop. SURF, which attempts to compensate for orientation changes, does better for large rotations but worse for small ones, highlighting once again that orientation invariance comes at a cost. The typical shape of the SURF-64 curve is a known artifact arising from approximating the Gaussian for scale-space analysis, degrading the performance under in-plane rotation for odd multiples of $\pi/4$ [121, 137].

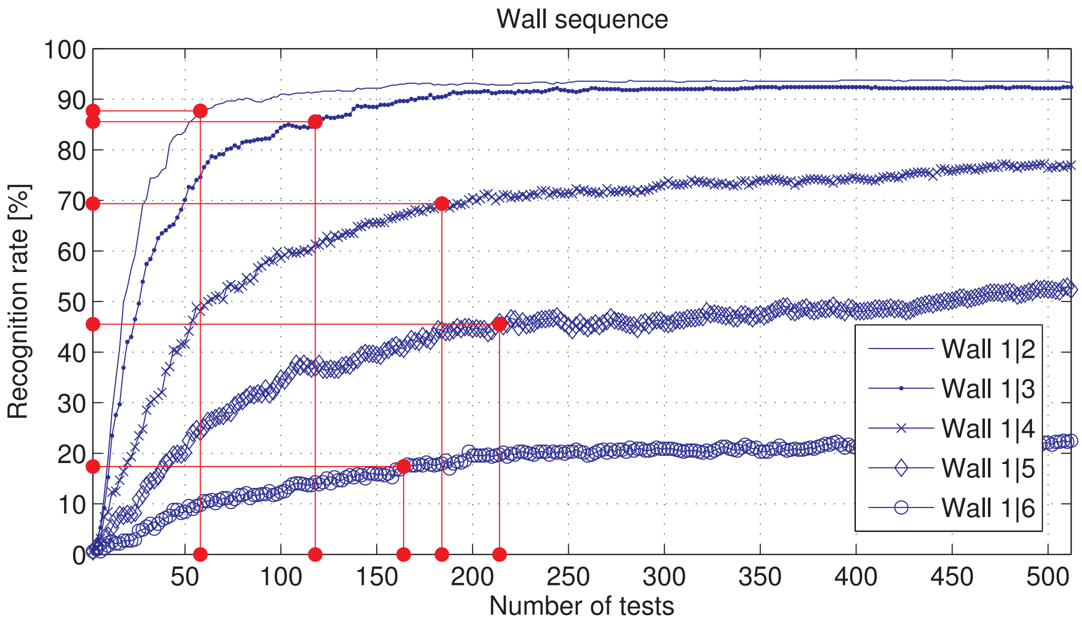


Figure 3.5: Varying the number of tests n_d of U-BRIEF. The plot shows the recognition rate as a function of n_d on Wall. The vertical and horizontal lines indicate the number of tests required to achieve the same recognition rate as U-SURF on respective image pairs. In other words, U-BRIEF requires only 58, 118, 184, 214, and 164 bits for Wall 1|2, . . ., 1|6, respectively, which compares favorably to U-SURF's $64 \cdot 4 \cdot 8 = 2048$ bits (assuming 4 bytes/float).

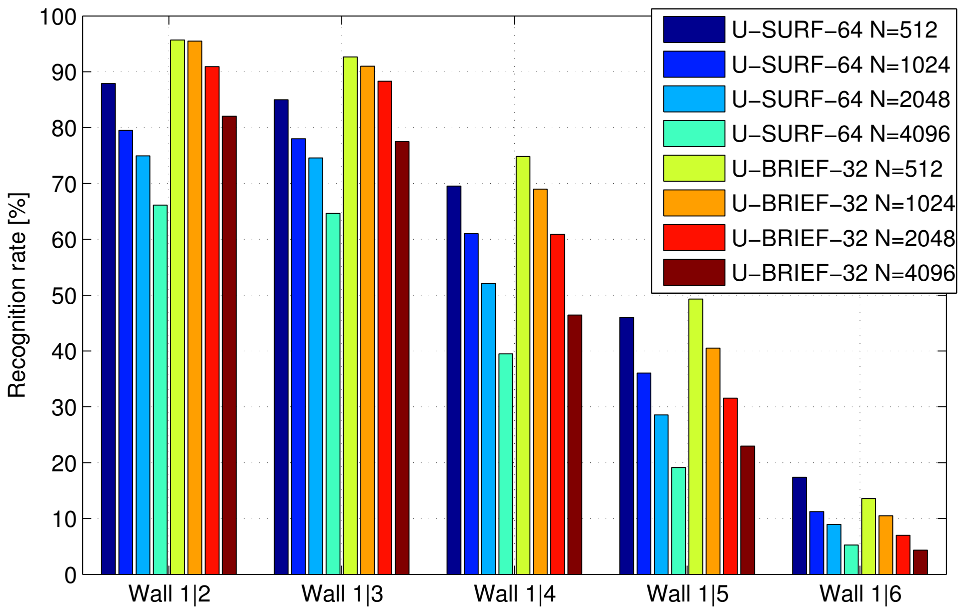


Figure 3.6: Scalability. For each of the five image pairs of Wall, we plot on the left side four sets of rates for N being 512, 1024, 2048, and 4096 when using U-SURF-64, and four equivalent sets when using U-BRIEF-32. Note that the recognition rate consistently decreases with increasing N. However, for the same N, BRIEF outperforms SURF, except in the last image pair where the recognition rate is equally low for both.

To complete the experiment, we plot two more curves. The first corresponds to O-BRIEF which is identical to U-BRIEF except that the tests are rotated according to the orientation estimation of SURF. O-BRIEF-32 is not meant to represent a practical approach but to demonstrate that the response to in-plane rotations is more a function of the quality of the orientation estimator rather than of the descriptor itself, as evidenced by the fact that the O-BRIEF-32 and SURF curves are almost perfectly superposed.

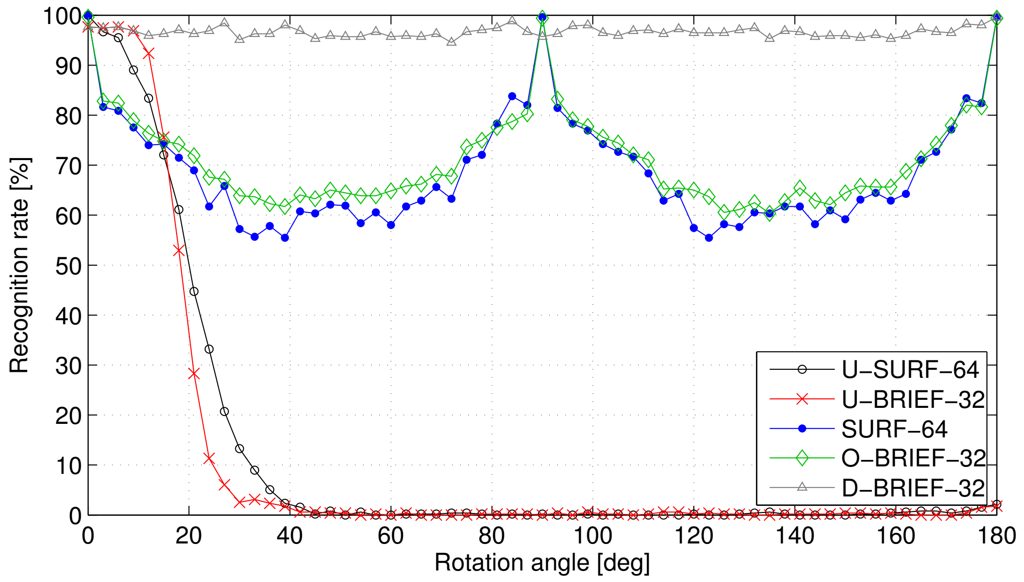


Figure 3.7: Recognition rate when matching the first image of the Wall dataset against a rotated version of itself, as a function of the rotation angle.

The second curve is labeled D-BRIEF-32. It is applicable to problems where full rotational invariance is required while no other source for orientation correction is available–a case seen less and less often. As discussed at the beginning of section 3.3, D-BRIEF exploits BRIEF's characteristic speed by pre-computing a database of U-BRIEF descriptors for several orientations, matching incoming frames against the entire database and picking the set of feature points with the probably highest number of correct matches. Doing so is practically viable and lets applications process incoming frames at 30 Hz. As a welcome side-effect, D-BRIEF also avoids the characteristic accuracy loss observed in scale space-based methods based on approximations of Gaussians as can be seen in Fig. 3.7.

3.4.3 Sensitivity to Scaling

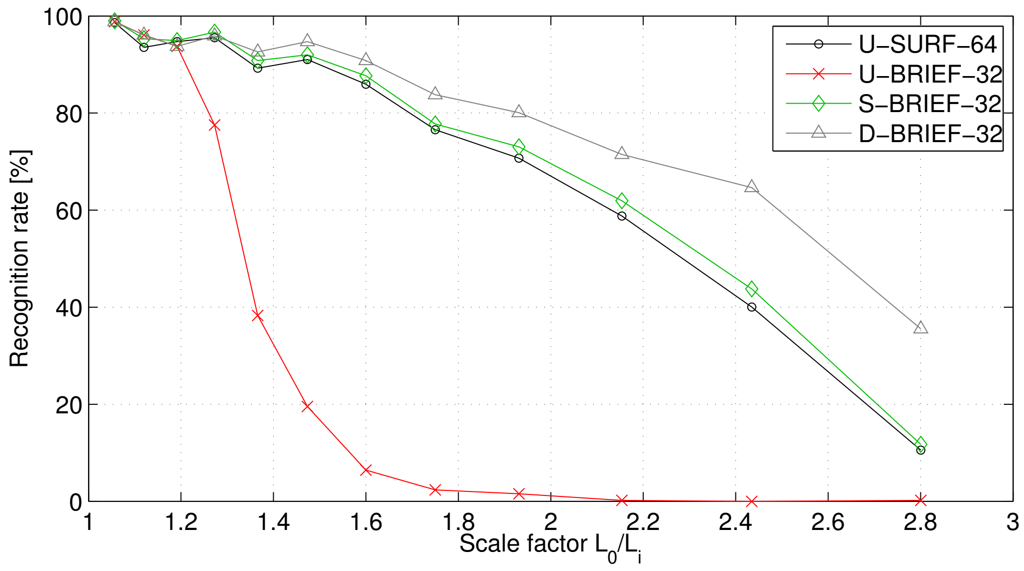


Figure 3.8: Recognition rate under (down)scaling using the first image from Wall. L_0: original image width, L_i: width of matched image i.

By default, U- and O-BRIEF are not taking scale information into account, although the smoothing provides a fair amount of robustness to scale changes as confirmed in the experiments below. We assess the extent of this tolerance in Figure 3.8 and plot the recognition rate as a function of the image scaling factor. The first two curves in the plot, U-SURF-64 and U-BRIEF-32, show that U-BRIEF is inherently robust to small and intermediate scale changes, at least to some limited extent. Larger changes result in a notable loss in accuracy.

Scale information, if available from a detector, can be readily integrated into BRIEF, which we demonstrate with the third curve in the plot, labeled S-BRIEF-32. The scale of a keypoint is used to adjust both the area which the tests cover and the smoothing kernel size before the tests are applied. Apparently, this results in a descriptor that behaves similarly to SURF under scale changes where potentially using a larger-than-standard kernel size for smoothing comes at no cost, thanks to integral images used for computing the box filter response.

The D-BRIEF curve in Fig. 3.8 describes the case where no scale information is available. As for rotational invariance, we can resort to pre-computing a database of scaled versions of the original image. The scales are chosen such that they cover the expected range of observable ones, which is in the present case $[1.0, 0.60, 0.43, 0.33]$. In a more general setup this interval may be centered around 1.0 to account for larger and smaller scales. However, since a scaling factor of 0.33 is smaller than what practical applications typically demand, that scale can be removed and another, larger one added. Therefore, 4 or even 3 scales might suffice for most applications.

Alternatively, a point detector that also returns a scale estimate can be used to achieve such scale-invariant behavior and in practice, a faster detector such as CenSurE [1] would be given preference over the still fairly slow Fast Hessian detector underlying SURF, let alone SIFT's DoG detector. In any case, changing the underlying point detector does not influence BRIEF's behavior, as we are going to show in the next section.

3.4.4 Robustness to Underlying Feature Detector

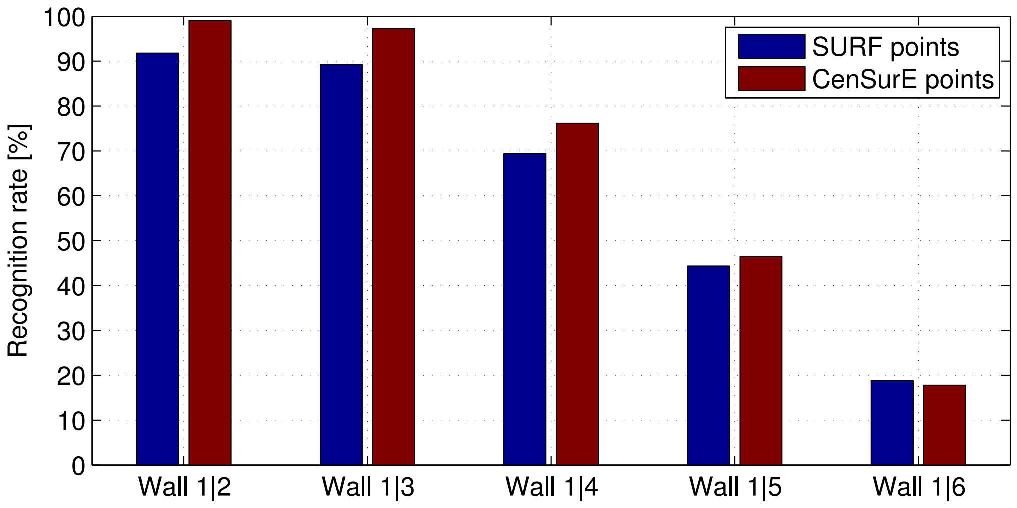


Figure 3.9: Using CenSurE feature points instead of SURF ones. U-BRIEF works slightly better with CenSurE feature points rather than with those from SURF.

To perform the experiments described above, we used SURF keypoints so that we could run both SURF and BRIEF on the same points. This choice was motivated by the fact that SURF requires an orientation and a scale and U-SURF a scale, which the SURF detector provides.

However, in practice, using the SURF detector in conjunction with BRIEF would negate part of the considerable speed advantage that BRIEF enjoys over SURF. It would make much more sense to use a fast detector such as CenSurE [1]. To test the validity of this approach, we therefore re-computed our recognition rates on the Wall sequence using CenSurE features[1] instead of SURF keypoints. As can be seen in Figure 3.9, U-BRIEF works even slightly better for CenSurE points than for SURF points.

		{U,O,S}-BRIEF-X (CPU)			U-SURF-64	
		X=16	X=32	X=64	CPU	GPU
Detection		1.61	1.61	1.61	160	6.93
Description	Gaussian	6.59	6.81	7.40	101	3.90
	Uniform	6.05	6.07	6.30		
	Uniform-Integral†	1.74	2.69	4.65		
	Uniform-Integral‡	1.01	1.98	3.91		
Matching	n^2-Matching w/o `POPCNT`	2.62	5.03	9.56	28.3	6.80
	n^2-Matching w/ `POPCNT`	0.433	0.833	1.58		

Table 3.2: CPU wall clock time in [ms] for {U,O,S}-BRIEF of length 16, 32, or 64 and U-SURF-64 on 512 points, median over 10 trials. Values for matching using D-BRIEF could be obtained by multiplying the given value by the number of orientations and/or scales used. The four row of numbers in the *Description* part of the table correspond to four different methods to smooth the patches before performing the binary tests. The two rows of numbers in the *Matching* part of the table correspond to whether or not the POPCNT instruction is part of the instruction set of the CPU being used. Neither the kernel choice or the presence of the POPCNT instruction affect SURF. †Integral image pre-computed. ‡Integral image re-used from a detector such as CenSurE.

3.4.5 Speed Estimation

In a practical setting where either speed matters or computational resources are limited, not only should a descriptor exhibit the highest possible recognition rate but also be as computationally frugal as possible. Table 3.2 gives timing results in milliseconds measured on a 2.66 GHz/Linux x86-64 laptop for 512 keypoints. We give individual times for the three main computational steps depending on various implementation choices. We also provide SURF timings on both CPU and GPU for comparison purposes.

1. **Detecting feature points.** For scale-invariant methods such as SURF or SIFT, the first step can involve a costly scale-space search for local extrema. In the case of BRIEF, any fast detector such as CenSurE [1] or FAST [180, 181] can be used. BRIEF is therefore at an advantage there.

2. **Computing descriptors.** When computing the BRIEF descriptor, the most time-consuming part of the computation is smoothing the image-patches. In Table 3.2, we provide times when using Gaussian smoothing, simple box filtering, and box filtering using integral images. The latter is of course much faster. Furthermore, as discussed in Section 3.3.1 and shown in Fig. 3.1, it does not result in any matching performance loss.

   In the specific case of the BRIEF-32 descriptor, we observe a 38-fold speed-up over U-SURF, without having to resort to a GPU which may not exist on a low-power device.

3. **Matching descriptors.** This involves performing nearest neighbor search in descriptor space. For benchmarking purposes, we used the simplest possible matching strategy by computing distances of every descriptor against every other. Even though the computation time is quadratic in terms of the number of points being matched, BRIEF is fast enough to so that it remains practical for real-time purposes over a broad range. Furthermore, for larger point sets, it would be easy to incorporate a more elaborate matching strategy [238].

   More technically, matching BRIEF's bit vectors mainly involves computing Hamming distances. The distance between two BRIEF vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ is computed as $\text{POPCNT}(\mathbf{b}_1 \text{ XOR } \mathbf{b}_2)$ where POPCNT is returning the number of bits set to 1. Older CPUs require POPCNT to be implemented in native C++ and BRIEF descriptors can be matched about 6 times faster than their SURF counterparts, as shown in Tab. 3.2. Furthermore, since POPCNT is part of the SSE4.2 [109] instruction set that newer CPUs, such as the Intel Core i7, support, the corresponding speed-up factor of BRIEF over SURF becomes 34. Note that recent ARM processors, which are typically used in low-power devices, also have such an instruction, called VCNT [12].

[1]Center Surrounded Extrema [1], or CenSurE for short, has been implemented in OpenCV 2.0 with some improvements and hence received a new name: *Star detector*, hinting at the shape of the bi-level filters.

In short, for all three steps involved in matching keypoints, BRIEF is significantly faster than the already fast SURF without any performance loss and without requiring a GPU. This is summarized graphically in Fig. 3.17.

3.5 Results

In this section, we compare our method against several competing approaches. Chief among them is the latest OpenCV implementation of the SURF descriptor [17], which has become a *de facto* standard for fast-to-compute descriptors. We use the standard SURF-64 version, which returns a 64-dimensional floating point vector and requires 256 bytes of storage. Because {U,S}-BRIEF, unlike SURF, do not correct for orientation, we also compare against U-SURF-64 [17], where the U also stands for *upright* and means that orientation is ignored [17].

As far as D-BRIEF is concerned, because its database of rotated patches has to be tailored to a particular application, we demonstrate it for real-time Augmented Reality purposes.

3.5.1 Datasets

We compare the methods on real-world data, using the six publicly available test image sequences depicted by Fig. 3.10. They are designed to test robustness to typical image disturbances occurring in real-world scenarios. They include

- viewpoint changes: Wall[2], Graffiti[2], Fountain[3], Liberty[4];
- compression artifacts: Jpg[2];
- illumination changes: Light[2];
- image blur Trees[2].

For all sequences except Libertywe consider five image pairs by matching the first image to the remaining five. The five pairs in Wall and Fountain are sorted in order of increasing baseline so that pair 1|6 is much harder to match than pair 1|2, which negatively affects the performance of *all* the descriptors considered here. The pairs from Jpg, Light, and Treesare similarly sorted in order of increasing difficulty.

The Wall and Graffiti scenes being planar, the images are related by homographies that we use to compute the ground truth. Both sequences involve substantial out-of-plane rotation and scale changes. Although the Jpg, Light, and Trees scenes are non-planar, the ground truth can also be represented accurately enough by a homography that is close to identity since there is almost no change in viewpoint.

By contrast, the Fountain scene is fully three-dimensional and contains substantial perspective distortion. As this precludes using a homography to encode the ground truth, we use instead accurate laser scan data, which is also available, to compute the flow field for each image pair.

These six datasets are representative of the challenges that an algorithm designed to match image pairs might face. However, they only involve relatively small numbers of keypoints to be matched in each individual image. To test the behavior of our algorithm on the much larger libraries of keypoints that applications such as image retrieval might involve, we used the publicly available Liberty dataset depicted by Fig. 3.11. It contains over 400k scale and rotation normalized patches sampled from a 3D reconstruction of the New York Statue of Liberty. The 64×64 patches are sampled around interest points detected using Difference of Gaussians and the correspondences between patches are found using multi-view stereo algorithm. The dataset created this way represent substantial perspective distortion of the 3D statue seen under various lighting and viewing conditions.

[2]`http://www.robots.ox.ac.uk/~vgg/research/affine` [150]
[3]`http://cvlab.epfl.ch/~strecha/multiview` [211]
[4]`http://cvlab.epfl.ch/~brown/patchdata/patchdata.html` [30]

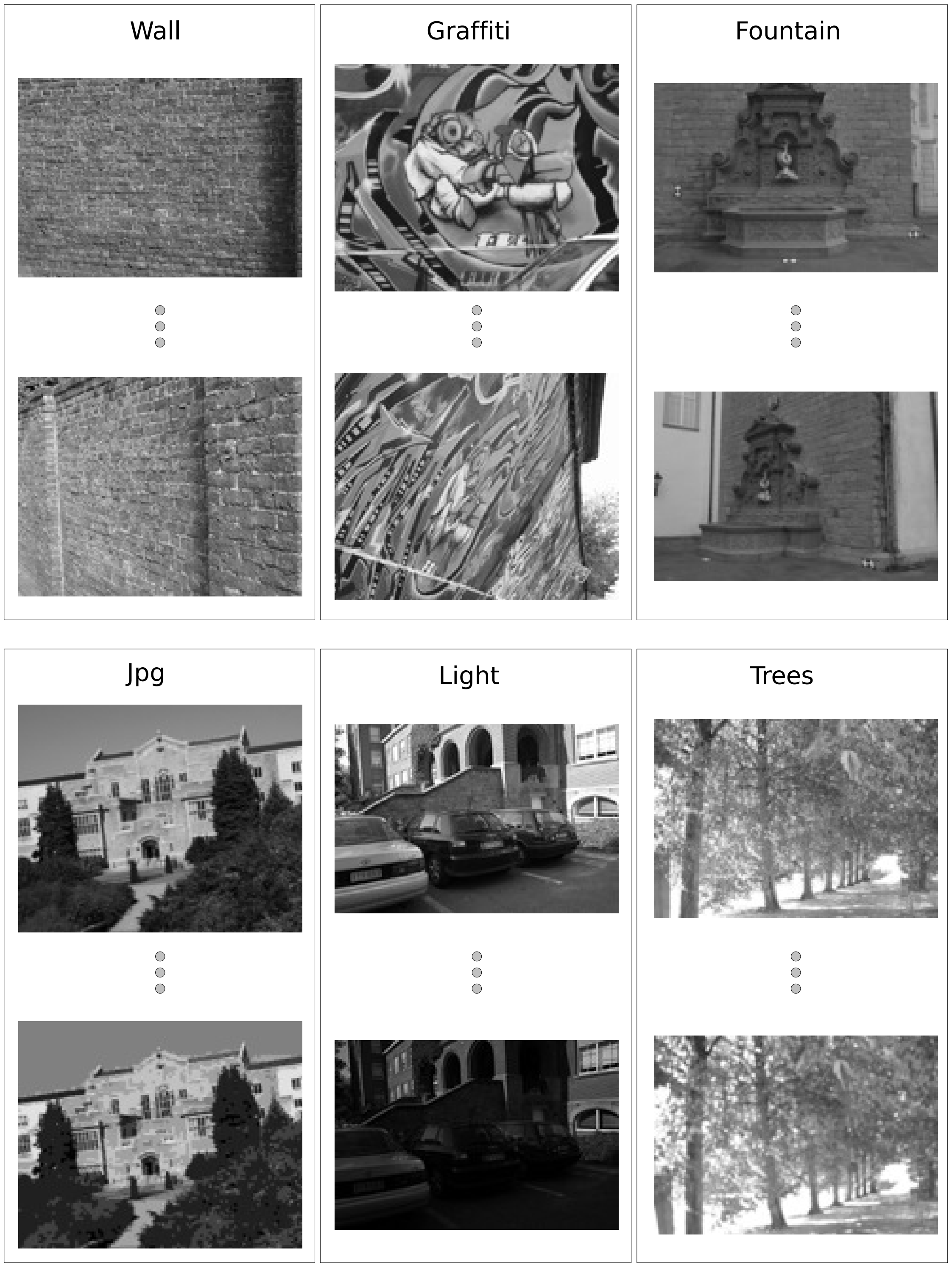


Figure 3.10: Test image sequences. The left column shows the first image, the right column the last image. Each sequence contains 6 images and hence we consider 5 image pairs per sequence by matching the first one against all subsequent images. This way, except for Graffiti, the pairs are sorted in ascending difficulty. More details in the text.

3.5.2 Comparing Performance for Image Matching Purposes

This section finally compares the BRIEF-32 descriptor against SIFT and SURF. For SIFT we employ the `siftpp` implementation by Vedaldi [231] that computes the standard 128-vector of real numbers, requiring 512 bytes of

Figure 3.11: Some of the images patches from the Liberty dataset used in our evaluation.

memory. For SURF we rely on the latest OpenCV implementation [27] where we use the standard SURF-64 version. The default parameter settings are used.

Although BRIEF can be computed on arbitrary kinds of feature points, this comparison is based on SIFT or SURF points, depending on which of the two we are actually comparing BRIEF to. While doing so removes the direct comparability of SIFT's and SURF's recognition rates, it is vital for a fair comparison to BRIEF. In practical applications, however, we would rather employ FAST [180] or CenSurE [1] features for computational efficiency. While FAST is faster to compute than CenSurE, the latter provides BRIEF with integral images that can be exploited for additional speed, see Tab. 3.2.

We measure the performance of the descriptors using Nearest Neighbor (NN) correctness. We refer to this measure as *recognition rate* ρ. It simply computes the fraction of descriptors that are NN in feature space while belonging to the same real-world point, and hence being a true match. Given an image pair, this is done as follows.

- Pick N feature points from the first image, infer the N corresponding points in the second image using the known geometric relation between them, and compute the $2N$ associated descriptors using the method under consideration.
- For each point in the first set, find the NN in the second one and call it a match.
- Count the number of correct matches n_c and compute $\rho := n_c/N$.

Note that not detecting feature points in the second image but using geometry instead prevents repeatability issues in the detector from influencing our results. Since we apply the same procedure to all methods, none is favored over any other.

The recognition rate is of great importance when the feature vectors are matched using exhaustive NN search. This is feasible in real-time when the number of features is not exceeding a few hundred or maybe one thousand as typical for SLAM or object detection applications. Even though approximate NN schemes exist, exact search is done whenever possible because the accuracy of such schemes quickly deteriorates in high-dimensional spaces.

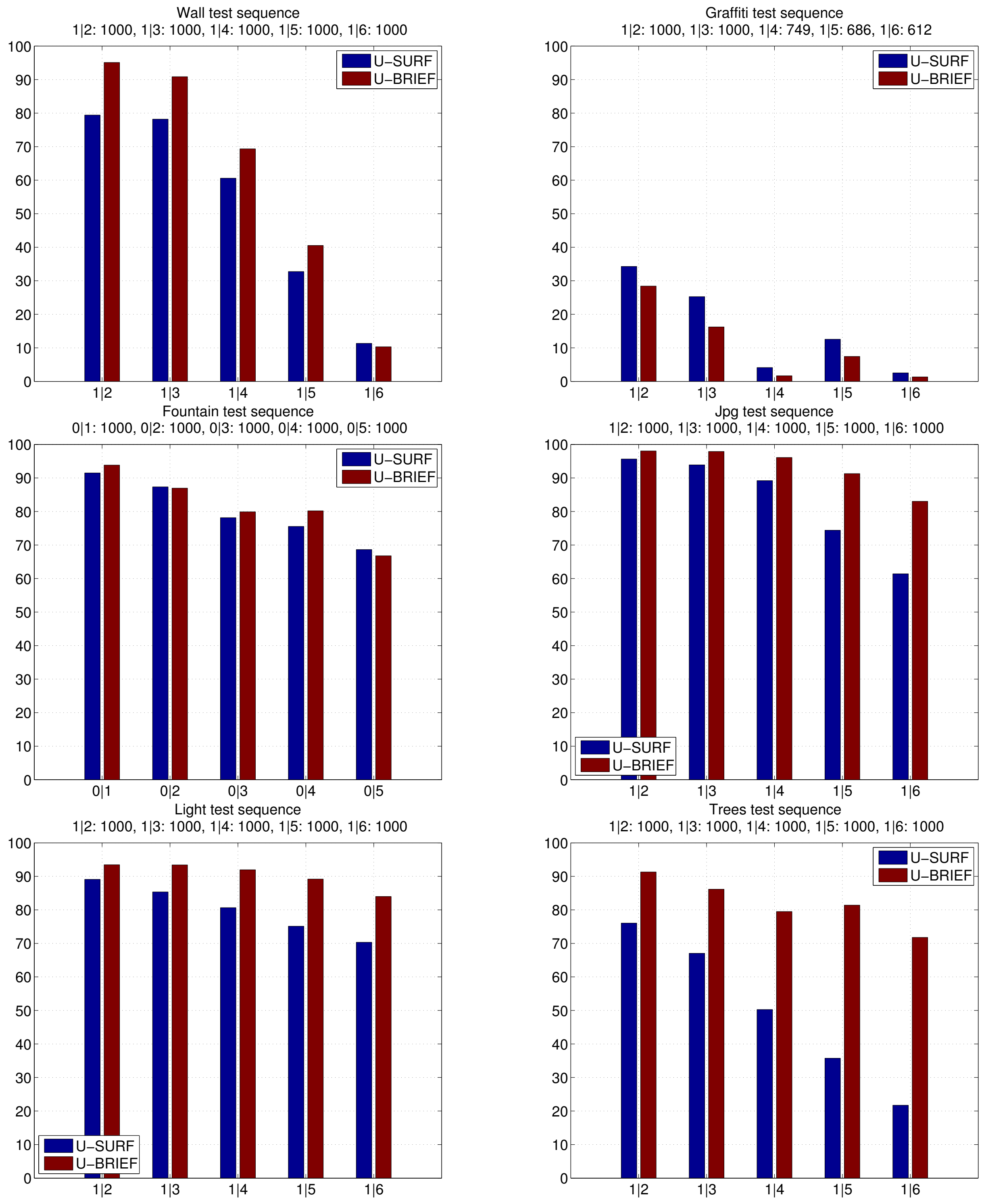


Figure 3.12: Recognition rate vs. image pairs of all sequences. Comparing to U-SURF.

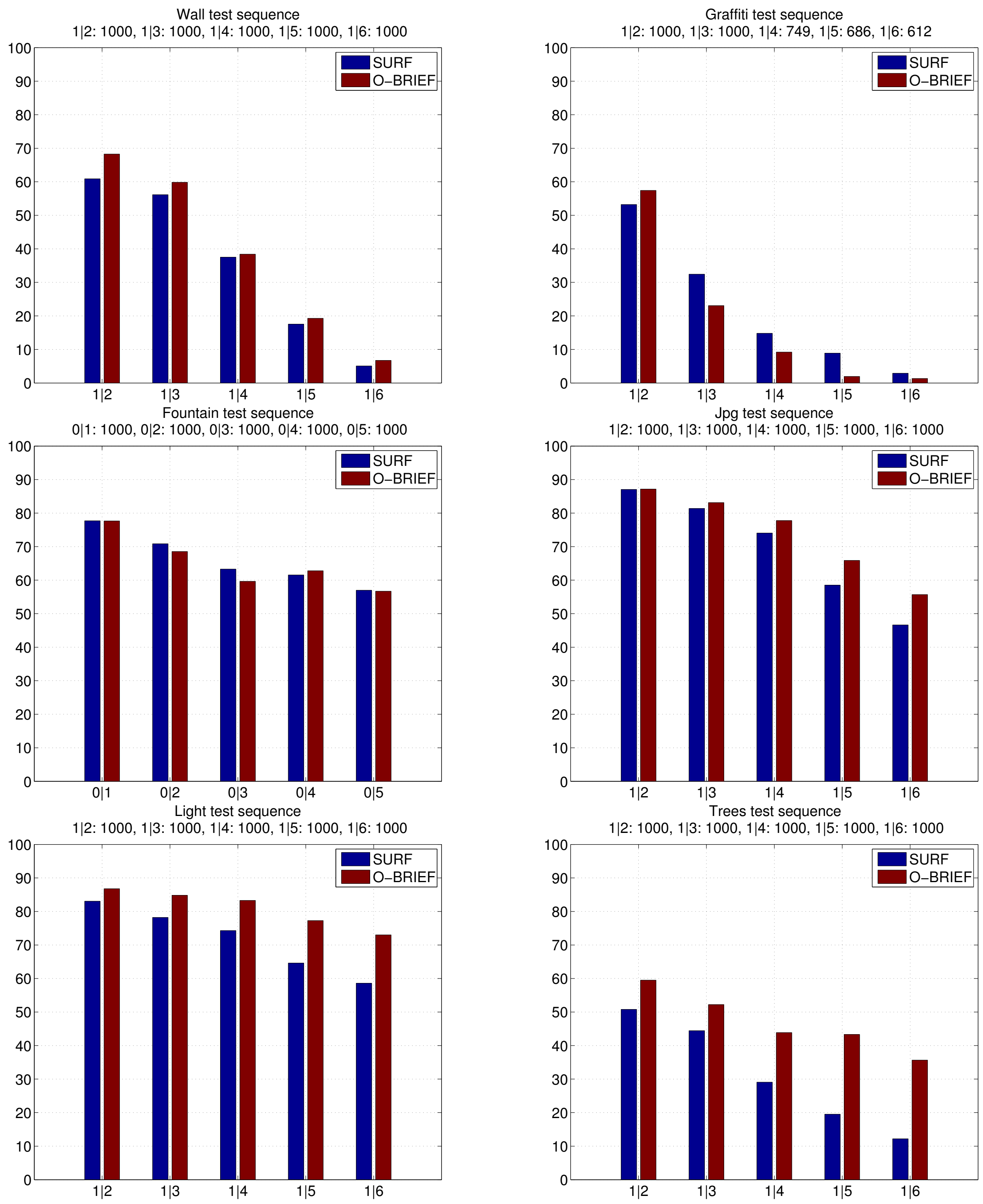


Figure 3.13: Recognition rate vs. image pairs of all sequences. Comparing to SURF.

In the following we show a large number of graphs to compare the different variants of BRIEF to SURF and SIFT. More specifically we use 6 image sequences, each containing 5 image pairs, to compare

- U-SURF and U-BRIEF that do not correct for orientation, and
- SIFT, SURF, and O-BRIEF that do using the same orientation estimate.

This means that a thorough comparison encompasses $2 \times 6 \times 2 = 24$ recognition rate plots. The factor 2 from SIFT and SURF cannot be dropped because—although the two are conceptually very similar—they are not working with

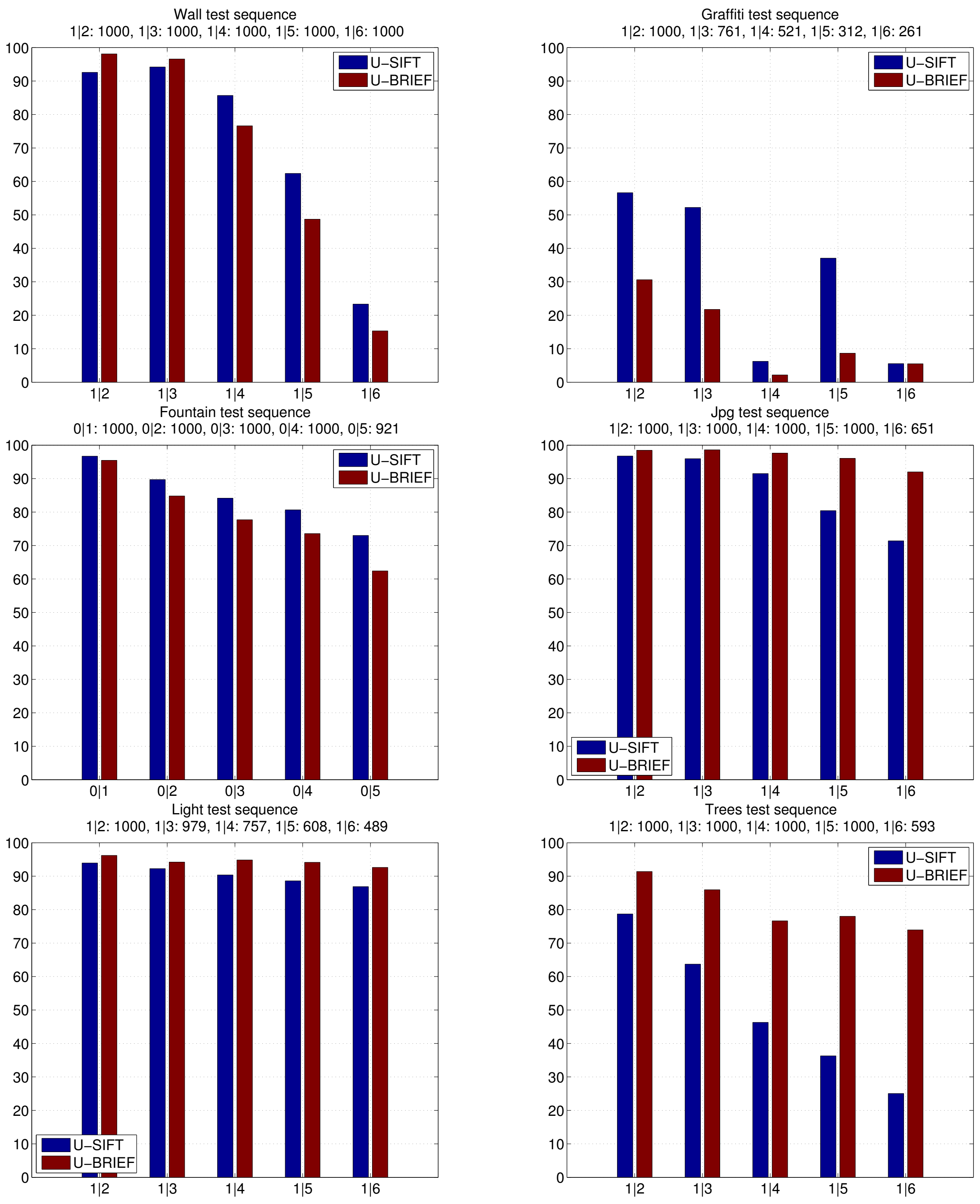


Figure 3.14: Recognition rate vs. image pairs of all sequences. Comparing to U-SIFT.

the same set of feature points and forcing either to use the other's will distort the results. In some cases we use points extracted by SIFT and in others by SURF.

In the title of each plot we give the name of the test sequence and the number of points that were matched for each image pair. When the detected points in a pair give rise to more than 1000 ground truth-compatible matches, 1000 of them are selected at random; otherwise the maximum available number of matches is used. Keeping the number of points constant for all evaluations makes comparisons easier.

Figs. 3.12 through 3.15 show the recognition rate of BRIEF together with those of SURF and SIFT for com-

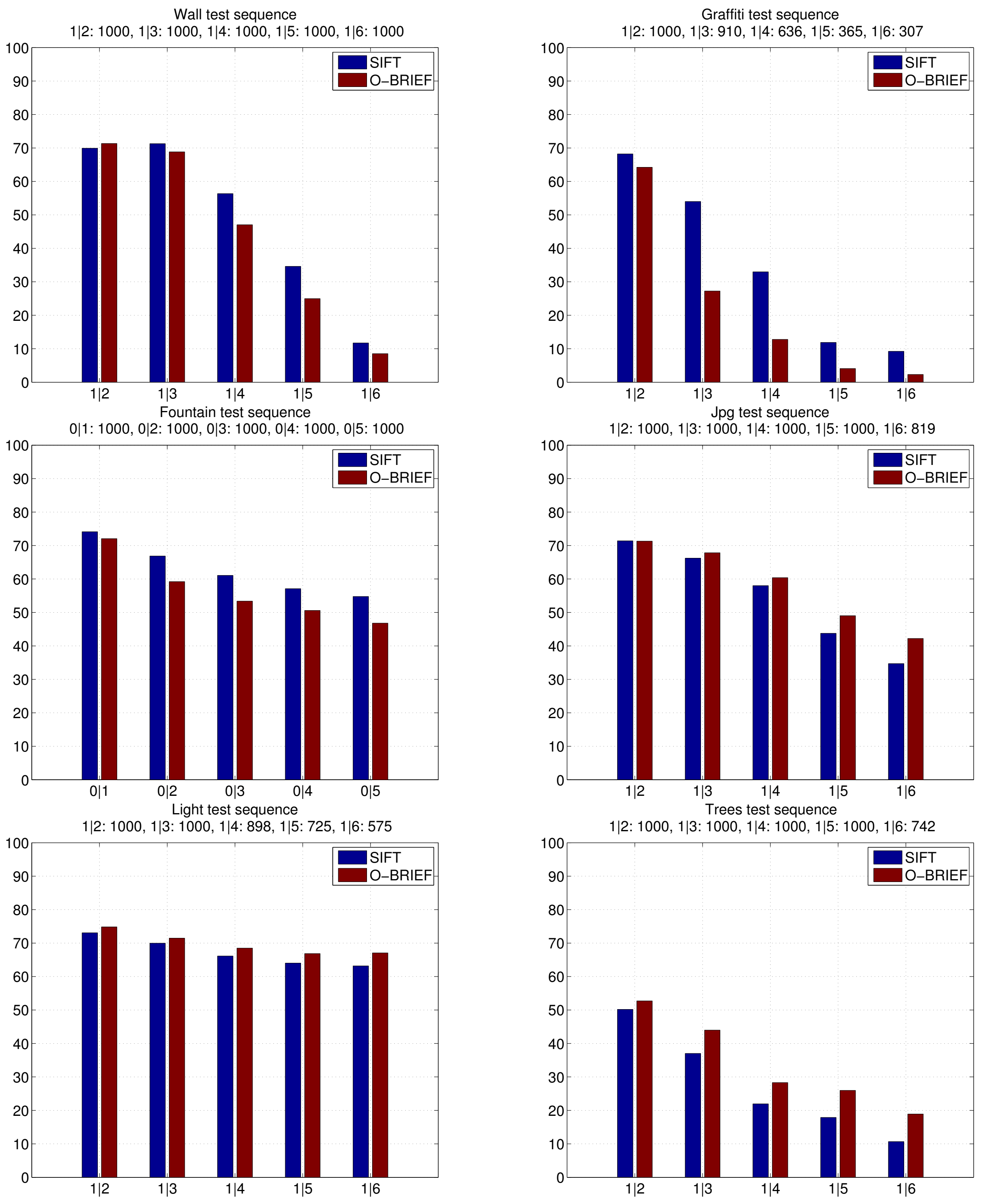


Figure 3.15: Recognition rate vs. image pairs of all sequences. Comparing to SIFT.

parison. Each of the four figures contains six graphs, one for each image sequence. Based on these plots, we make the following observations:

- On all sequences except on Graffiti, U-BRIEF outperforms U-SURF, as can be seen in Fig. 3.12.
- Using orientation correction, O-BRIEF also outperforms SURF except on Graffiti. O-BRIEF does slightly better than U-BRIEF on Graffiti but not much (Figure 3.13).

- SURF and SIFT are both substantially more sensitive to image blur than BRIEF is, and slightly more sensitive to compression and illumination artifacts (Figures 3.12 to 3.15).
- O-BRIEF works better with SURF's orientation assignment than with that of SIFT features (Figures 3.13 and 3.15).
- SIFT is more robust to viewpoint changes than both BRIEF and SURF.

In summary, the rough ordering 'SIFT $\gtrsim$ BRIEF $\gtrsim$ SURF' applies in terms of robustness to common image disturbances but, as we will see below, BRIEF is much faster than both. Note, however, that these results were obtained for $N = 1000$ keypoints per image. This is representative of what has to be done when matching two images against each other but not of the more difficult problem of retrieving keypoints within a very large database. We now turn to this problem.

3.5.3 Influence of the Number of Keypoints to be Matched

To demonstrate what happens when the number of keypoints to be matched increases, we use the Liberty dataset that contains over 400k patches and the corresponding ground-truth. We extracted subsets of corresponding patches of cardinalities ranging from 1 to 7000. To use the same metric as in Section 3.5.2, we report recognition rates as proportions of the descriptors for which the nearest-neighbor in descriptor space corresponds to the same real-world point.

Since the Liberty dataset consists of patches extracted around interest points, no feature detection is needed and descriptors are computed for the central point of the patch. Since both SIFT and SURF perform sampling at a certain scale, we optimized this scaling parameter for optimal performance. For a fair comparison, we did the same for BRIEF by adjusting the size of the patch that was used to create the descriptor.

Fig. 3.16 and Table 3.3 depict the resulting recognition rates. All descriptors perform less and less well with increasing dataset size and, even for small cardinalities, performances are consistently worse than those reported in Section 3.5.2 because the data set is much more challenging.

On this dataset, SIFT performs best. BRIEF-32 does better than U-SURF for cardinalities up to $N = 1000$ and worse for larger ones. To outperform U-SURF it becomes necessary to make BRIEF more discriminative by using more bits and using BRIEF-64 or BRIEF-128, which are still smaller and, even more importantly, much faster to compute as will be discussed below. Furthermore, BRIEF-128 still only requires 1024 bits, which is half of the 2048 bits required to store the 64 floats of U-SURF.

Since SIFT performs better than BRIEF, whatever its size, we also investigated what part of the loss of discriminative power simply comes from the fact that we are using a binary descriptor, as opposed to a floating point one. To this end, we used a method we recently introduced to binarize SIFT descriptors and showed to be state-of-the-art [207] and as short as quantized descriptors like DAISY [30]. It involves first computing a projection matrix that is designed to jointly minimize the in-class covariance of the descriptors and maximize the covariance across classes, which can be achieved in closed-form. This being done, optimal thresholds that turn the projections into binary vectors are computed so as to maximize recognition rates. This amounts to performing Linear Discriminant Analysis on the descriptors before binarization. Because it is coded on 16 bytes, it is denoted as LDA-16 in the graph of Fig. 3.16. It performs better than BRIEF, but at the cost of computing much more since one has to first compute the full SIFT descriptor and then binarize it. In applications where time is of the essence and memory requirements are less critical, such as establishing correspondences for augmented reality purposes, it therefore makes sense to use BRIEF-32, -64, or -128 as required by the difficulty of the scenes to be matched.

3.5.4 Comparing Computation Times

In this section we compare the timings of the different methods. Fig. 3.17 shows the total time required for detecting, describing and matching 512 feature points. BRIEF is almost two orders of magnitude faster than its fastest competitor, U-SURF. In particular, the standard version of BRIEF, BRIEF-32, appears particularly efficient as it exploits integral images for smoothing and the `POPCNT` instruction for computing the Hamming distance.

All timings are the medians over 10 instances of the task at hand. We measure the CPU wall clock time programatically, that is using the system's tick count value rather than counting cycles. When the system load is low, the two values should be close, though from a practical point of view, the wall clock time is the one that truly counts.

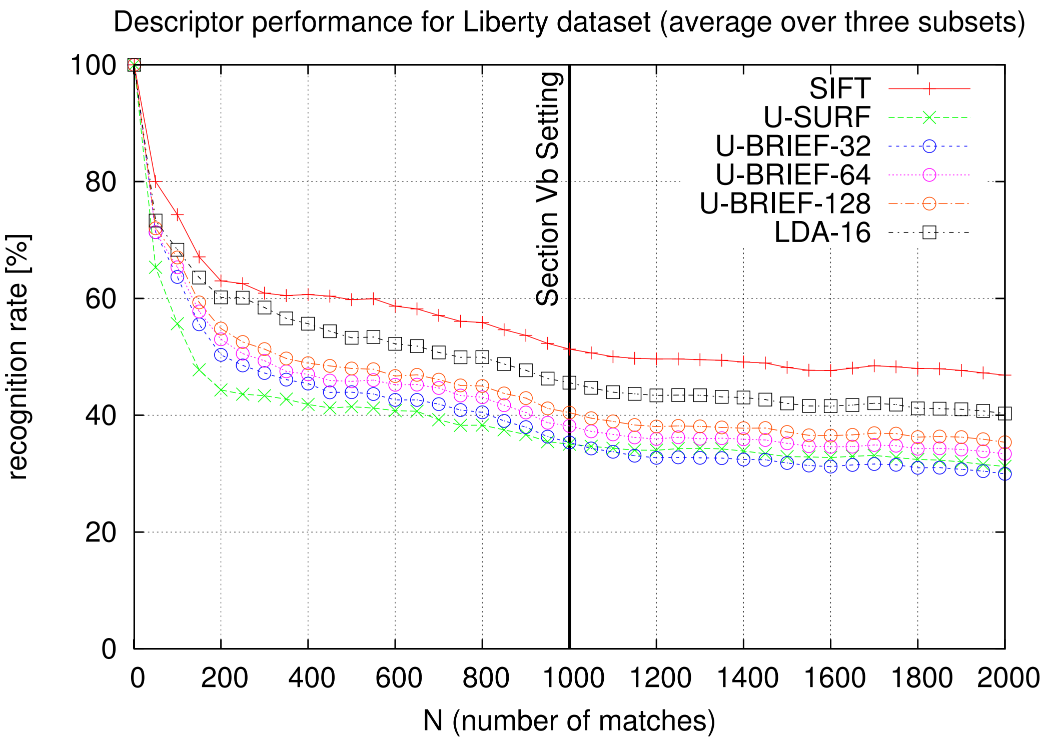


Figure 3.16: Descriptors performance as the function of the number N of corresponding pairs on the Liberty dataset.

Table 3.3: Recognition rates of descriptors for increasing numbers N of corresponding pairs from the Liberty dataset.

N	2000	3000	5000	7000
SIFT	46.87%	40.7%	40.26%	33.26%
SURF	31.23%	26.9%	25.88%	21.16%
BRIEF-32	29.97%	26.08%	23.96%	19.29%
BRIEF-64	33.37%	28.98%	27.46%	22.33%
BRIEF-128	35.37%	30.53%	29.16%	23.49%
LDA-16	40.28%	35.13%	34.56%	28.19%

3.5.5 On-the-Fly Object Detection

To demonstrate the value of D-BRIEF, we present a system designed for real-time object detection where the object to be detected can be learned *instantaneously*. In other words, unlike earlier systems, such as our own real-time mouse pad detection application presented in [169], this one does not require a training phase, which may take several minutes. Using the same implementation using SURF features, for example, is impossible unless resorting to the GPU. By contrast, the current application builds on D-BRIEF-32P and runs on a single CPU while maintaining frame-rate performance. We show results in a few frames in Fig. 3.18.

In such an application, unlike those presented earlier, full rotational invariance is clearly a desirable feature. This can be achieved thanks to BRIEF's extremely efficient processing pipeline, even without resorting to any trick for speeding things up. To this end, when the target image is acquired, we build a template database that contains 18 rotated views at 3 scales of the target, totaling up to 54 views. The additional descriptors are computed on synthetic images obtained by warping the target image. Then each incoming frame is matched against the database and the one with the highest number of matches to features-in-template score is selected. These matches are still noisy and hence fed to RANSAC, which robustly estimates a homography between the views that is used to re-project the template target's corners into the image frame.

While this basic system works at 25–30 Hz, its computation time and memory requirements increase linearly

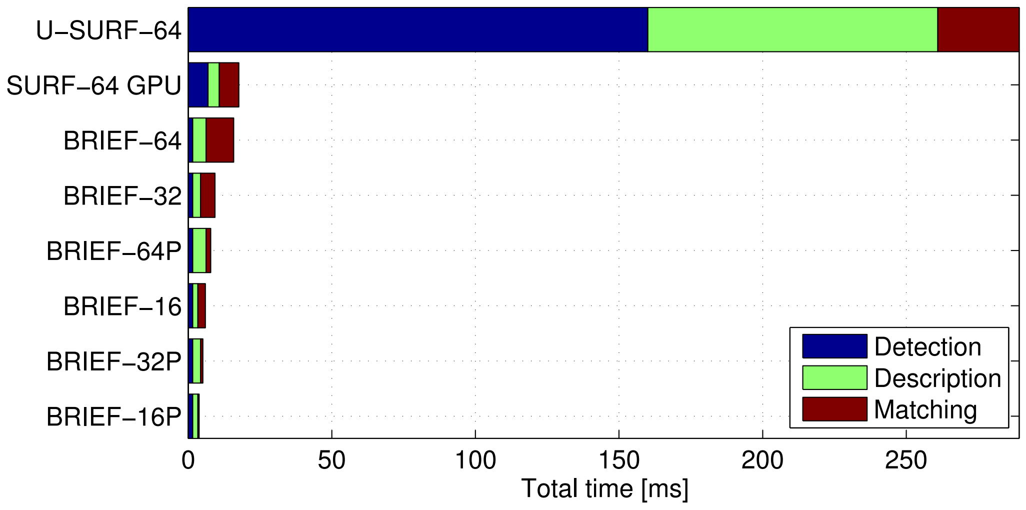


Figure 3.17: Timings for a detection–describe–match cycle. Ordered by decreasing total time. The methods shown include three versions of BRIEF and U-SURF-64 implemented on both the CPU and the GPU. The P suffix for BRIEF indicates that in the matching step, the `POPCNT` instruction has been employed.

with the number of templates. Its performance would be further enhanced by i) enabling tracking and/or ii) using a more efficient feature point search as was done in [238] to achieve real-time performance using SIFT and Ferns. Furthermore, to achieve a more efficient feature point search, a tree resembling much that of a Vocabulary Tree [158] could be used.

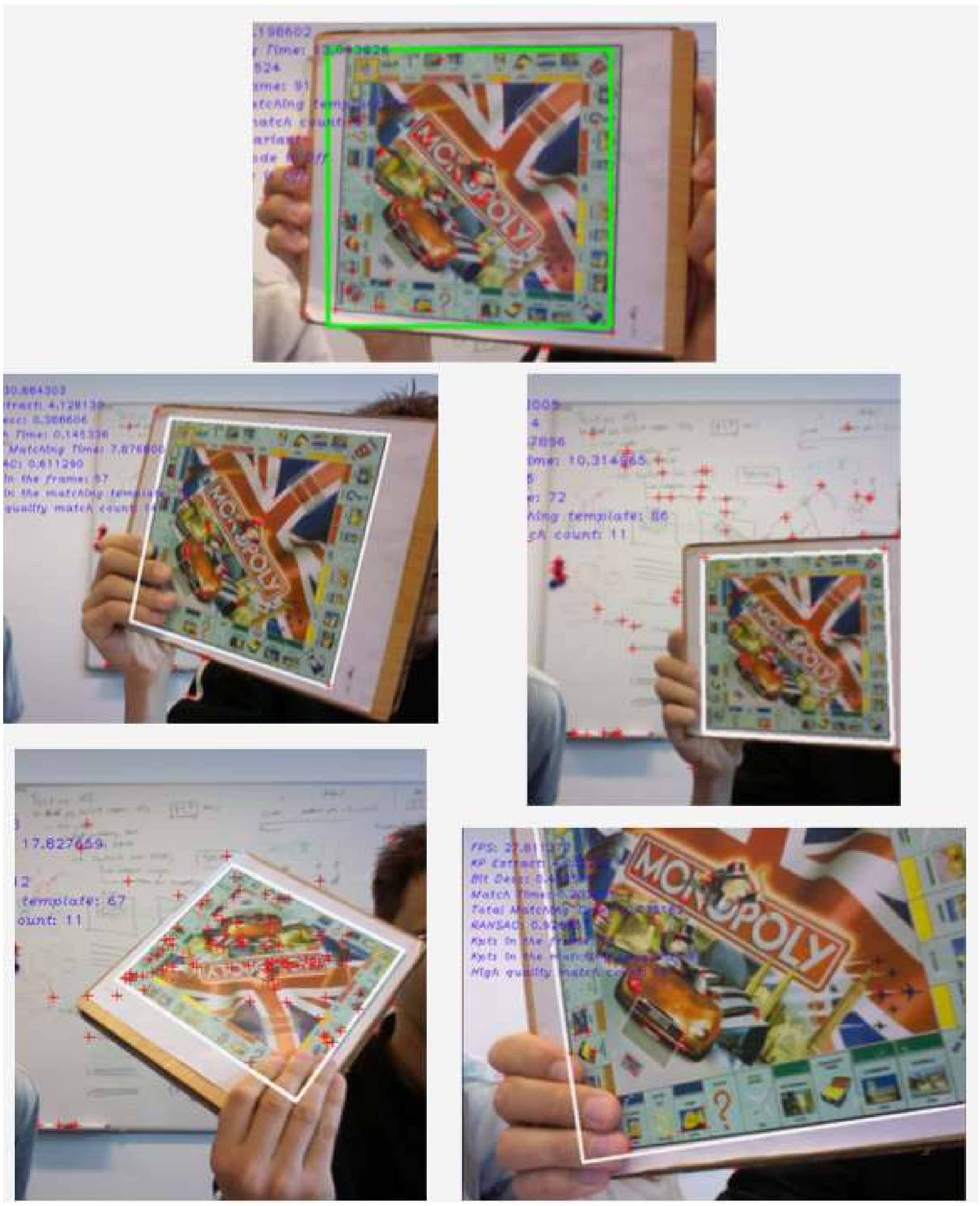


Figure 3.18: Screenshots from the object detection application. Top image: Initialization by drawing a square around the detection target. Remaining 4 images: The system at run-time. Note that it tolerates substantial scale changes and is fully invariant to in-plane rotation while running at 27 FPS on the CPU of a simple laptop.

3.6 Conclusion

In this chapter, we have introduced the BRIEF descriptor that relies on a relatively small number of intensity difference tests to represent an image patch as a binary string. Not only is construction and matching for this descriptor much faster than for other state-of-the-art ones, it also tends to yield higher recognition rates, as long as invariance to large in-plane rotations is not a requirement.

It is an important result from a practical point of view because it means that real-time matching performance can be achieved even on devices with very limited computational power. It is also important from a more theoretical viewpoint because it confirms the validity of the recent trend [194, 222] that involves moving from the Euclidean to the Hamming distance for matching purposes.

Since the publication of our first paper about BRIEF [35], [93] independently evaluated BRIEF on other matching problems, and confirmed its performance. [258] remarked that BRIEF can be interpreted as a “hashing scheme on the Kendall’s tau metric”: The Kendall’s tau metric counts the number of pairwise disagreements between two ranking lists. In the case of BRIEF, these ranking lists would be the pixel intensities of the image patches after ranking; The Hamming distance between two BRIEF descriptors can then be seen as a Monte Carlo evaluation of the Kendall’s tau distance between these two lists.

Several authors also proposed other descriptors, including ORB [182], FREAK [4], or BRISK [134]. These descriptors also use comparisons of pairs of values, but apply different schemes to sample the image locations or rely on other features such as image gradients with remarkable performance [5]. Some of these methods also come with their own detection and orientation estimation methods, which make their implementation practical. In particular ORB comes with OpenCV, and is now a popular option for matching feature points. OpenCV also provides an efficient matching method based on multi-probe LSH [141], which revealed to be a suitable method for binary descriptors.

As mentioned in the introduction, BRIEF is not based on Machine Learning. However, it was inspired by our previous work on Ferns. It also motivates us to work on binary descriptor learning, which is the topic of the next chapter.

Chapter 4

Learning Image Descriptors with Boosting

4.1 Introduction

With BRIEF, we showed that binary descriptors could be a powerful tool for Computer Vision. We also showed that randomly selected features based on intensity comparisons perform well for building a descriptor. However, this scheme is clearly non-optimal. With Tomasz Trzcinski's PhD, we wanted to learn to compute a binary descriptor, that is, to optimize the combination of features to improve the matching accuracy, without loosing the computation speed advantage of other binary descriptors.

Learning descriptors was not new [30, 197, 182]. However, the learning approach we developed with Tomasz Trzcinski and Mario Christoudias, a young postdoc, is not limited to any pre-defined sampling pattern and provides a more general framework than previous training-based methods. It also nicely scales linearly with the number of training examples, making it more amenable to large scale problems, and results in highly accurate descriptor matching.

In this chapter, we describe the method published in [224], which extends [226] and [223]. It is a supervised learning framework that finds low-dimensional but highly discriminative descriptors. With our approach, image patch appearance is modeled using local non-linear filters that are selected with boosting. We build upon [193] that also relies on boosting to compute a descriptor, and show how we can use it as a way to efficiently select features, from which we compute a compact representation. Analogous to the kernel-trick, our approach can be seen as applying a *boosting-trick* [41] to obtain a non-linear mapping of the input to a high-dimensional feature space. Unlike kernel methods, boosting allows for the definition of intuitive non-linear feature mappings that can share a close connection with existing, prevalent keypoint descriptors.

Nevertheless, as image databases grow in size, modern solutions to local feature-based image indexing and matching must not only be accurate but also highly efficient to remain viable. Binary descriptors are of particular interest as they require far less storage capacity and offer much faster matching times than conventional floating-point descriptors [77, 207, 34, 134, 182, 227], or even quantized descriptors [30]. In addition, they can be used directly in hash table techniques for efficient Nearest Neighbor search [160, 144], and their similarity can be computed very quickly on modern CPUs based on the Hamming distance.

However, as our experiments show, state-of-the-art binary descriptors often perform worse than their floating-point competitors: some are built on top of existing representations such as SIFT or GIST by relying on training data [77, 207], and are therefore limited by the performance of the intermediate representation. Others start from raw image intensity patches, but focus on computation speed and rely on fast-to-compute image features [34, 182, 134, 227], which limit their accuracy.

To address these shortcomings, we extend our learning framework to the binary case and we train a highly discriminative yet compact binary descriptor. This extension demonstrates the real advantage of our approach as it enables us to not only compress the descriptor, but also significantly decrease the processing cost and memory footprint. For each dimension of the resulting binary representation, we learn a hash function of the same form as an AdaBoost strong classifier, that is the sign of a linear combination of non-linear weak learners.

The resulting binary descriptor, which we refer to as BinBoost, significantly outperforms its binary competitors and exhibits a similar accuracy to state-of-the-art floating-point or quantized descriptors at a fraction of the storage and matching cost. Furthermore it is more complex to optimize, and we show how to efficiently optimize our hash functions using boosting.

This chapter extends our previous work [226, 223] in the following way. First, we show in Section 4.3 that our method provides a general descriptor learning framework that encompasses previously published approaches and the state-of-the-art intensity-based [34, 182, 134, 227] and gradient-based descriptors [140, 18, 207].

Our results show that the ability to effectively optimize over the descriptor filter configuration leads to a significant performance boost at no additional computational cost compared with the original hand-designed representation. We experiment with additional weak learner families from the ones previously used and we show that our learning method performs well independently of the underlying weak learner type. Finally, we provide an exhaustive experimental evaluation of our methods on several challenging datasets, including Mikolajczyk dataset [150] and UKBench [158]. We also illustrate how our method is not restricted to local feature descriptors and can be successfully extended to new application domains and other types of image data, and we demonstrate this on a face recognition problem.

The rest of this chapter is organized as follows. In Section 4.2 we discuss related work. In Section 4.3 we describe our method: we first show how to efficiently construct our set of weak learners, from which we compute a compact floating-point representation. We then explain how to extend this approach to build a binary local feature descriptor. In Section 4.4 we discuss different weak learner types and in Section 4.5 we describe the experimental setup of our method and its parameters. Section 4.6 presents the comparison of our descriptors against the state-of-the-art methods. Finally, in Section 4.7, we show how our framework can be used to learn representations of other types of data, namely face images.

4.2 Related Work

Many recent techniques form binary descriptors based on simple pixel intensity comparisons [34, 134, 182]. Huffman coding [39] and product quantization [114] have also been explored to compress histogram of oriented gradient descriptors. Similarly, [259] develops a binary edge descriptor based on a histogram of normalized gradients. Although more efficient, these hand-designed descriptors are generally not compact and not as accurate as their floating point equivalents.

For this reason, machine learning has been applied to improve both the efficiency and accuracy of image descriptor matching. Unsupervised hashing methods learn compact binary descriptors whose Hamming distance is correlated with the similarity in the original input space [77, 124, 185, 248, 247]. Semantic hashing [185] trains a multi-layer neural network to learn compact representative binary codes. Spectral hashing [248] minimizes the expected Hamming distance between similar training examples, and was recently extended to optimize over example affinities [247]. Similarly, [124, 159] find codes whose Hamming distances well approximate the original Euclidean ones. In [240, 77], iterative and sequential optimization strategies that find projections with minimal quantization error are explored. While these approaches have proven highly effective for finding compact binary codes, they rely on pre-defined distance or similarity measures and in many cases are limited to the accuracy of the original input space.

Supervised learning approaches can learn feature spaces tailored to specific tasks [111, 138, 207, 240]. They exploit labeled example pairs or triplets that encode the desired proximity relationships of the learned metric. In [111], a Mahalanobis distance metric is learned and optimized with respect to labeled distance constraints. Linear Discriminant Analysis is applied in [77, 207, 227] to learn discriminative feature embeddings. Semi-supervised sequential learning algorithms are proposed in [138, 240] for finding discriminative projections. Similar to these approaches, most methods define a linear transformation of the data in either the original or a kernelized feature space and rely on a pre-specified kernel function to capture non-linearities. While they are well-suited for image categorization and indexing tasks for which task-specific kernels have been proposed, such as in [80], they are less applicable to local descriptor matching where the appropriate choice of kernel function is less well understood.

Recent descriptor learning methods have emphasized the importance of learning not only the optimal weighting, but also the optimal *shape* or pooling configuration of the underlying representation [30, 197]. In [30], they optimize over different feature selection and pooling strategies of gradient-based features, however, the criterion considered—the area below the ROC—is not analytical making it difficult to optimize. Following [30], a convex optimization strategy was developed in [197]. To make learning tractable, however, a limited set of pooling configurations was considered and restricted to circular, symmetrically arranged pooling regions centered about the patch. As shown in our experiments, our binary descriptor achieves a similar accuracy to these methods at a fraction of the matching cost.

Jointly optimizing over descriptor weighting and shape poses a difficult problem due to the potentially large number of pooling configurations one might encounter. This is especially true for learning generic shapes where the number of pooling regions can easily be in the millions, even for small patch sizes. Fortunately, this is a problem for which AdaBoost [70] and other boosting methods [52, 236] are particularly well-suited. Although greedy, boosting is an effective method for constructing a highly accurate predictor from a large (potentially infinite) collection of constituent parts. The resulting *boosting-trick* like the kernel-trick, maps the input to a high-dimensional feature space, however, the mapping it defines is explicit, with the learned embedding assumed to be sparse [41, 179]. As a result and unlike kernel methods, boosting appears to be an efficient way to find a non-linear transformation of the input that is naturally parameterized over both the descriptor shape and weighting.

In this chapter, we introduce a family of boosted descriptors that are trained with boosting for discriminative power and compactness. Our work is inspired by Boosted Similarity Sensitive Coding (SSC) [193] which is the first application of boosting to learn an image similarity measure and was later extended in [222] to be used with a Hamming distance. Boosted SSC, however, only considers linear projections of the input and generally results in fairly high dimensional descriptions. Our methods, on the other hand, rely on more complex weak learners and produce descriptors, both binary and floating-point, of a much lower dimensionality.

We also propose a sequential learning method similar to [240, 138] except, unlike these methods, our boosting approach learns both the optimal shape and weighting of the features associated with each bit. Our descriptor can also be seen as a two layer neural network [185], since each coordinate of the descriptor is computed from a linear combination of pooled image features. As shown in our experiments, this results in highly accurate and compact descriptors whose final performance rivals that of the leading binary and floating point descriptors.

4.3 Method

In this section we describe methods for learning local feature descriptors with boosting. We first formulate our problem by defining the exponential loss objective function we use to learn a similarity embedding between image patches. We then present different similarity measures which, when plugged into our boosting framework, can be used to train floating-point and binary descriptors.

4.3.1 Problem formulation

Given an image intensity patch $\mathbf{x}$, we look for a descriptor $C(\mathbf{x}) = [C_1(\mathbf{x}), \ldots, C_D(\mathbf{x})]$ which maps the patch to a D-dimensional vector. This descriptor can be learned by minimizing the exponential loss with respect to a desired similarity function $f\left(C(\mathbf{x}), C(\mathbf{y})\right) = f_C(\mathbf{x}, \mathbf{y})$ defined over image patch pairs:

$$\mathcal{L} = \sum_{i=1}^{N} \exp(-l_i f_C(\mathbf{x}_i, \mathbf{y}_i)) \tag{4.1}$$

where $\mathbf{x}_i, \mathbf{y}_i \in \mathcal{R}^p$ are training intensity patches and $l_i \in \{-1, 1\}$ is a label indicating whether it is a similar $(+1)$ or dissimilar (-1) pair. Minimizing Eq. (4.1) finds an embedding which maximizes the similarity between pairs of similar patches, while minimizing it for pairs of different patches.

This formulation allows for numerous similarity functions f_C. We consider similarity functions of the form

$$f_C(\mathbf{x}, \mathbf{y}) = C(\mathbf{x})^T \, \mathbf{A} \, C(\mathbf{y}) \tag{4.2}$$

where $\mathbf{A} \in \mathcal{R}^{D \times D}$ is a symmetric matrix. This defines a general class of symmetric similarity measures that can be factorized to compute a feature descriptor independently over each input and used to define a wide variety of image descriptors. In what follows, we consider different choices of $\mathbf{A}$ and $C(\cdot)$ ordering them in increasing complexity.

4.3.2 Boosted Similarity Sensitive Coding (SSC)

The Boosted SSC method proposed in [193] considers a similarity function defined by a simply weighted sum of thresholded response functions $\{h_d(\cdot)\}_{d=1}^{D}$:

$$f_{SSC}(\mathbf{x}, \mathbf{y}) = \sum_{d=1}^{D} \alpha_d h_d(\mathbf{x}) h_d(\mathbf{y}) \,. \tag{4.3}$$

This function is the weighted Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ and corresponds to Eq. (4.2) where $\mathbf{A}$ is restricted to be a diagonal matrix. The importance of each dimension d given by the α_d's and the resulting D-dimensional descriptor is a floating-point vector $C(\mathbf{x}) = [\sqrt{\alpha_d} h_d(\mathbf{x})]_{d=1}^{D}$, where α is constrained to be positive.

Substituting f_{SSC} for f_C in Eq. (4.1) gives

$$\mathcal{L}_{SSC} = \sum_{i=1}^{N} \exp\left(-l_i \sum_{d=1}^{D} \alpha_d h_d(\mathbf{x}_i) h_d(\mathbf{y}_i) \right) \,. \tag{4.4}$$

In practice the space of h's is prohibitively large, possibly infinite, making the explicit optimization of $\mathcal{L}_{SSC}$ difficult, however, this constitutes a problem for which boosting is particularly well suited [70]. Although boosting is a greedy optimization scheme, it is an effective method for constructing a highly accurate predictor from a collection of weak predictors h. Due to its greedy nature, however, the weak learners found using Boosted SSC often remain highly redundant and hence inefficient. In what follows, we modify the similarity function $f_C(\mathbf{x}, \mathbf{y})$ so that it becomes better suited for learning low-dimensional, discriminative embeddings with boosting.

4.3.3 FPBoost

To mitigate the potentially redundant embeddings found by boosting we propose an alternative similarity measure f_{FP} that models the correlation between weak response functions:

$$f_{FP}(\mathbf{x}, \mathbf{y}) = \sum_{k,k'} \alpha_{k,k'} h_k(\mathbf{x}) h_{k'}(\mathbf{y}) = \mathbf{h}(\mathbf{x})^T \mathbf{A} \mathbf{h}(\mathbf{y}), \tag{4.5}$$

where $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_K(\mathbf{x})]$ and $\mathbf{A}$ is a $K \times K$ matrix of coefficients $\alpha_{k,k'}$. This similarity measure is a generalization of Eq. (4.3). In particular, f_{FP} is equivalent to the Boosted SSC similarity measure in the restricted case of a diagonal $\mathbf{A}$.

Substituting the above expression into Eq. (4.1) gives

$$\mathcal{L}_{FP} = \sum_{i=1}^{N} \exp\left(-l_i \sum_{k,k'} \alpha_{k,k'} h_k(\mathbf{x}_i) h_{k'}(\mathbf{y}_i) \right) \,. \tag{4.6}$$

We optimize $\mathcal{L}_{FP}$ using a two step learning strategy. We first apply AdaBoost to find good weak learners $\{h_k\}_{k=1}^{K}$ by minimizing Eq. (4.4) on the training samples as in [193]. Then we apply stochastic gradient descent to find an optimal weighting over the selected features that minimizes Eq. (4.6). To guarantee that the similarity function f_{FP} remains symmetric, we restrict the coefficients $\alpha_{k,k'}$ of $\mathbf{A}$ to be symmetric. This optimization strategy is sub-optimal but we found it to work well in practice.

The similarity function of Eq. (4.5) defines an implicit feature mapping over example pairs. In order to compute the feature descriptors independently over each input, we need to factorize $\mathbf{A}$. As we constrain $\mathbf{A}$ to be symmetric, we can factorize it into the following form:

$$\mathbf{A} = \mathbf{B}\mathbf{W}\mathbf{B}^T = \sum_{k=1}^{K} w_k \mathbf{b}_k \mathbf{b}_k^T \tag{4.7}$$

where $\mathbf{W} = \text{diag}([w_1, \cdots, w_K])$, $w_k \in \{-1, 1\}$, $\mathbf{B} = [\mathbf{b}_1, \cdots, \mathbf{b}_k]$, $\mathbf{b} \in \mathcal{R}^K$.

Eq. (4.5) can then be re-expressed as

$$f_{FP}(\mathbf{x}, \mathbf{y}) = \sum_{d=1}^{D} w_d \left(\sum_{k=1}^{K} b_{d,k} h_k(\mathbf{x}) \right) \left(\sum_{k=1}^{K} b_{d,k} h_k(\mathbf{y}) \right) \,. \tag{4.8}$$

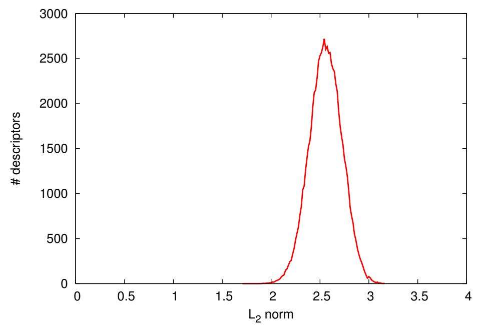


Figure 4.1: Histogram of the L_2 norms of 75k FPBoost descriptors extracted from the images of Mikolajczyk dataset [150]. The L_2 norms are upper-bounded by $\sqrt{\sum_{d=1}^{D} |\mathbf{b}_d|_1^2}$ which equals 4 in this case. A significant portion of the descriptors have a comparable magnitude and, hence, we can use Euclidean distance in place of the equivalent inner product to measure descriptor similarity.

For $D < K$ (i.e., the effective rank of $\mathbf{A}$ is $D < K$) the factorization represents a smoothed version of $\mathbf{A}$ discarding the low-energy dimensions that typically correlate with noise, and in practice leading to further performance improvements. The factorization of Eq. (4.8) defines a signed inner product between the embedded feature vectors and provides increased efficiency with respect to the original similarity measure[1]. Moreover, it is easy to show that the signed inner product is equivalent to the Euclidean distance under the assumption that the descriptors have comparable magnitudes. As shown in Fig. 4.1, this is the case in practice and, hence, we can leverage the existing methods for fast approximate nearest neighbor search which rely on Euclidean distances.

The final embedding $C(\mathbf{x}) = \mathbf{B}^T\mathbf{h}(\mathbf{x})$ results in a D-dimensional floating-point descriptor based on K weak learners that we call FPBoost$_K$-D. The projection matrix $\mathbf{B}$ defines a discriminative dimensionality reduction optimized with respect to the exponential loss objective of Eq. (4.6). As seen in our experiments, in the case of redundant weak learners this results in a considerable feature compression, and therefore offering a more compact description than the original input patch. Although compact and highly discriminant, the descriptors learned using FPboost are real-valued and, hence, can be slow to match and costly to store. Next, we consider a modification to FPboost that as we show can be used to learn a highly accurate and efficient binary descriptor.

4.3.4 BinBoost

To learn a binary descriptor we propose a modified similarity measure that extends f_{FP} to operate within a D-dimensional Hamming space:

$$\begin{aligned} f_B(\mathbf{x}, \mathbf{y}) &= \sum_{d=1}^{D} \operatorname{sgn}\left(\mathbf{b}_d^T \mathbf{h}_d(\mathbf{x})\right) \operatorname{sgn}\left(\mathbf{b}_d^T \mathbf{h}_d(\mathbf{y})\right) \\ &= \sum_{d=1}^{D} C_d(\mathbf{x}) C_d(\mathbf{y}) \end{aligned} \tag{4.9}$$

where $C_d(\mathbf{x}) = \operatorname{sgn}\left(\mathbf{b}_d^T \mathbf{h}_d(\mathbf{x})\right)$ and $\mathbf{h}_d(\mathbf{x}) = [h_{d,1}(\mathbf{x}) \dots h_{d,K}(\mathbf{x})]^T$ are K weak learners weighted by the vector $\mathbf{b}_d = [b_{d,1} \dots b_{d,K}]^T$. The resulting binary descriptor, which we call BinBoost$_K$-D, is a D-dimensional binary vector built using K weak learners by applying $C(\mathbf{x}) = \{C_d(\mathbf{x})\}_{d=1}^{D}$.

Substituting f_B for f_C in Eq. (4.1) gives

$$\mathcal{L}_B = \sum_{n=1}^{N} \exp\left(-\gamma\, l_n \sum_{d=1}^{D} C_d(\mathbf{x}) C_d(\mathbf{y})\right) . \tag{4.10}$$

This optimization problem is closely related to Eq. (4.4), but instead of weighting the dimensions with different α_d values we use a constant weighting factor γ. This enables us to compute the similarity more efficiently as it

[1]Matching two sets of descriptors each of size N is $\mathcal{O}(N^2K^2)$ under the original measure and $\mathcal{O}(NKD + N^2D)$ provided the factorization, resulting in significant savings for reasonably sized N and K, and $D \ll K$.

is now equivalent to the Hamming distance. More importantly, the $\{C_d(\cdot)\}$ functions are much more complex than the weak learners h_d as they are thresholded linear combinations of weak learner responses. The resulting optimization is discontinuous and non-convex and in practice the space of all possible weak learners h is discrete and prohibitively large. In what follows we develop a greedy optimization algorithm to solve this difficult problem and jointly optimize over the weak classifiers of each bit, $\mathbf{h}_d$ and their associated weights $\mathbf{b}_d$.

We first proceed as in regular AdaBoost. We optimize the $\{C_d(\cdot)\}$ functions iteratively, and at iteration d, the $C_d(\cdot)$ function that minimizes Eq. (4.10) is also the one that maximizes the weighted correlation of its output and the data labels [188]. Using this fact, at iteration d, the optimal $\mathbf{b}_d$ and $\mathbf{h}_d$ can be taken as

$$\underset{\mathbf{b}_d,\mathbf{h}_d}{\operatorname{argmax}} \sum_{n=1}^{N} l_n\, W_d(n) C_d(\mathbf{x}) C_d(\mathbf{y}) \,, \tag{4.11}$$

where

$$W_d(n) = \exp\left(-\gamma l_n \sum_{d'=1}^{d-1} C_{d'}(\mathbf{x}) C_{d'}(\mathbf{y})\right) \tag{4.12}$$

is a weighting that is very similar to the one used in regular Adaboost. This means that pairs that are incorrectly classified by the previous iterations are assigned a higher weight, whereas the weight of those correctly classified is decreased.

The sign function in $C_d(\cdot)$ is non-differentiable, and Eq. (4.11) is thus still hard to solve. We therefore apply the spectral relaxation trick [138, 240] and approximate the sign function using its signed magnitude, $\text{sgn}(x) \approx x$. This yields:

$$\begin{aligned}
& \underset{\mathbf{b}_d,\mathbf{h}_d}{\operatorname{argmax}} \sum_{n=1}^{N} l_n\, W_d(n) C_d(\mathbf{x}) C_d(\mathbf{y}) \\
\approx\; & \underset{\mathbf{b}_d,\mathbf{h}_d}{\operatorname{argmax}} \sum_{n=1}^{N} l_n\, W_d(n) \left(\mathbf{b}_d^T \mathbf{h}_d(\mathbf{x}_n)\right) \left(\mathbf{b}_d^T \mathbf{h}_d(\mathbf{y}_n)\right) \\
=\; & \underset{\mathbf{b}_d,\mathbf{h}_d}{\operatorname{argmax}} \sum_{n=1}^{N} l_n\, W_d(n) \mathbf{h}_d(\mathbf{x}_n)^T \mathbf{b}_d \mathbf{b}_d^T \mathbf{h}_d(\mathbf{y}_n) \\
=\; & \underset{\mathbf{b}_d,\mathbf{h}_d}{\operatorname{argmax}}\; \mathbf{b}_d^T \left(\sum_{n=1}^{N} l_n\, W_d(n) \mathbf{h}_d(\mathbf{x}_n) \mathbf{h}_d(\mathbf{y}_n)^T\right) \mathbf{b}_d \,.
\end{aligned} \tag{4.13}$$

As for Eq. (4.6), we first select a vector $\mathbf{h}_d(\mathbf{x})$ of suitable weak classifiers by minimizing Eq. (4.4) using the algorithm of [193] on the training samples, this time initially weighted by the $W_d(n)$ weights. The vector $\mathbf{b}_d$ is defined only up to a scale factor, and we then solve for it by looking for

$$\underset{\mathbf{b}_d}{\operatorname{argmax}}\, \mathbf{b}_d^T \mathbf{M} \mathbf{b}_d, \text{ s.t. } \|\mathbf{b}_d\|_2 = 1 \tag{4.14}$$

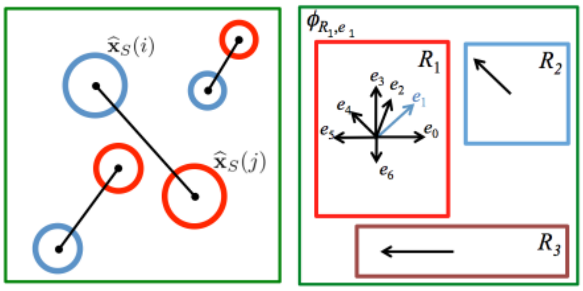
where

$$\mathbf{M} = \sum_{n=1}^{N} l_n\, W_d(n) \mathbf{h}_d(\mathbf{x}_n) \mathbf{h}_d(\mathbf{y}_n)^T \,. \tag{4.15}$$

Eq. (4.14) defines a standard eigenvalue problem and the optimal weights $\mathbf{b}_d$ can therefore be found in closed-form as the eigenvector of $\mathbf{M}$ associated with its largest eigenvalue.

Although not globally optimal, this solution returns a useful approximation to the solution to Eq. (4.11). Moreover, thanks to our boosting scheme even a sub-optimal selection of $C_d(\cdot)$ allows for an effective minimization.

We still have to explain how we choose the γ parameter. Note that its value is needed for the first time at the end of the first iteration, and we set this parameter after finding C_1 using the formula from regular Adaboost. We use the rule $\gamma = \nu \cdot \frac{1}{2} \log \frac{1+r_1}{1-r_1}$ where $r_1 = \sum_{n=1}^{N} W_1(n)\, l_n\, C_1(\mathbf{x}_n) C_1(\mathbf{y}_n)$ and ν is a shrinkage parameter used to regularize our optimization as described in [90]. In practice, we use $\nu = 0.4$.



(a) Intensity-based (b) Gradient-based

Figure 4.2: Overview of the intensity and gradient-based weak learners. To compute the responses of intensity-based weak learners, we compare the image intensity values after Gaussian smoothing at two locations i and j. Using boosting, we optimize both the locations and Gaussian kernel sizes, S. The gradient-based learners consider the orientations of gradients normalized within a given region. Boosting allows us to find the pooling configuration of the gradient regions and optimize the values of the corresponding thresholds.

4.4 Weak learners

The employed weak learner family encodes specific design choices and desired descriptor properties. In this section we present two weak learner types inspired from existing, prevalent keypoint descriptors. The simpler, yet less discriminative weak learners are based on pixel intensities. The more complex and computationally expensive weak learners rely on gradient images. Below we provide a detailed description of each along with their parameters.

4.4.1 Intensity-based learners

The intensity-based weak learners rely on BRIEF-like comparisons of pre-smoothed image intensities. More precisely, we define the output of our weak learner as:

$$h(\widehat{\mathbf{x}}_S; i, j, S) = \begin{cases} 1 & \text{if } \widehat{\mathbf{x}}_S(i) \leq \widehat{\mathbf{x}}_S(j) \\ -1 & \text{otherwise} \end{cases} \tag{4.16}$$

where $\widehat{\mathbf{x}}_S(i)$ is the pixel intensity of $\mathbf{x}$ pre-smoothed with a Gaussian kernel of size $S \in \{3, 5, 7, \ldots, 15\}$ at position i.

The above formulation allows us to optimize the selection of the sampling points as it was done, *e.g.* in [182], except we minimize a loss function with boosting rather than the responses' corellation with a stochastic algorithm.

Inspired by the sampling scheme of BRISK [134] and FREAK [3], we also optimize the value of the Gaussian kernel size S which defines the amount of smoothing applied to the image before comparing the intensity values, in addition to the positions i and j. This adds an additional degree of freedom to our optimization framework and, therefore, encompasses the formulation of many recently proposed binary feature descriptors, such as BRISK, FREAK and ORB.

4.4.2 Gradient-based learners

The gradient-based weak learners consider the orientations of intensity gradients over image regions [6]. They are parameterized by a rectangular region R over the image patch $\mathbf{x}$, an orientation e, and a threshold T, and are defined as

$$h(\mathbf{x}; R, e, T) = \begin{cases} 1 & \text{if } \phi_{R,e}(\mathbf{x}) \leq T \\ -1 & \text{otherwise} \end{cases}, \tag{4.17}$$

with

$$\phi_{R,e}(\mathbf{x}) = \sum_{m \in R} \xi_e(\mathbf{x}, m) \Big/ \sum_{e' \in \Phi, m \in R} \xi_{e'}(\mathbf{x}, m), \tag{4.18}$$

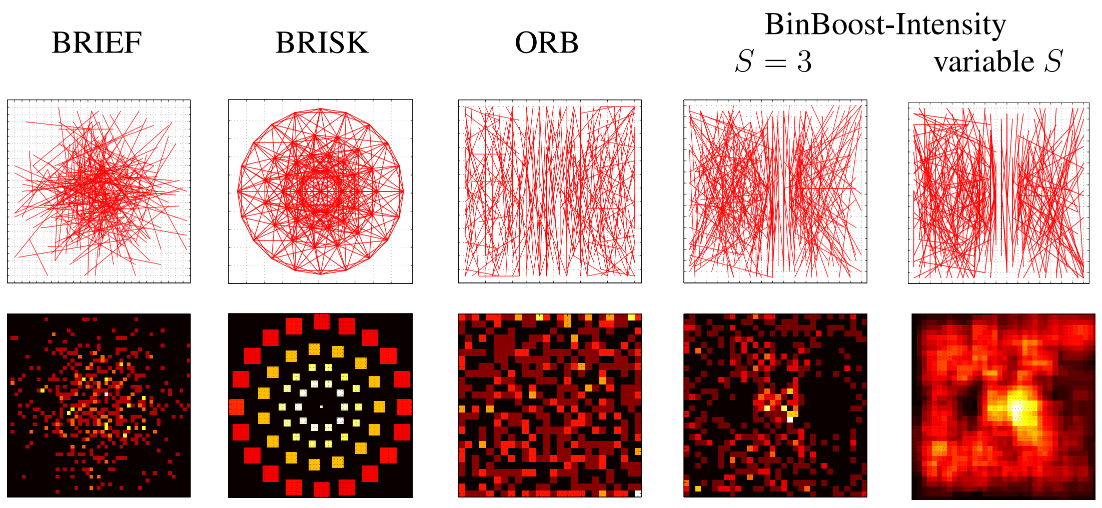


Figure 4.3: Visualization of the intensity tests (first row) and spatial weight heat maps (second row) employed by BRIEF, ORB, BRISK and our BinBoost$_1$-256 descriptor trained with intensity-based weak learners on rectified patches from the Liberty dataset. BRIEF picks its intensity tests from an isotropic Gaussian distribution around the center of the patch, while the sampling pattern of BRISK is deterministic. The intensity tests of ORB are selected to increase the variance of the responses, while reducing their correlation. This results in a pronounced vertical trend which can also be seen in the case of BinBoost. Nevertheless, the heat maps show that the tests for BinBoost-Intensity are — similarly to BRIEF — more dense around the center of the patch while the ones used in ORB present a more uniform distribution.

and

$$\xi_e(\mathbf{x}, m) = \max(0, \cos(e - o(\mathbf{x}, m)) \, , \tag{4.19}$$

where $o(\mathbf{x}, m)$ is the orientation of the image gradient in $\mathbf{x}$ at location m. The orientation e is quantized to take values in $\Phi = \{0, \frac{2\pi}{q}, \frac{4\pi}{q}, \cdots, (q-1)\frac{2\pi}{q}\}$ with q is the number of quantization bins. As noted in [6] this representation can be computed efficiently using integral images.

4.5 Experimental setup

In this section, we first describe our evaluation framework. We then present a set of initial experiments which validate our approach and allow us to select the correct parameters for our descriptors. Our approach improves over the state-of-the-art mostly with the binary version of our boosted descriptors, and we focus here on this version. Nevertheless the optimized parameters remain valid also for the floating-point descriptor.

4.5.1 Evaluation framework

We evaluate the performance of our methods using three publicly available datasets: Liberty, Notre Dame, and Yosemite [30]. Each of them contains over 400k scale- and rotation-normalized 64×64 patches. These patches are sampled around interest points detected using Difference of Gaussians and the correspondences between patches are found using a multi-view stereo algorithm. The resulting datasets exhibit substantial perspective distortion and changing lighting conditions. The ground truth available for each of these datasets describes 100k, 200k and 500k pairs of patches, where 50% correspond to match pairs, and 50% to non-match pairs. In our experiments, we use sub-sampled patches of size 32×32 and the descriptors are trained on each of the 200k datasets and we use the held-out 100k dataset for testing. We report the results of the evaluation in terms of ROC curves and 95% error rate which is the percent of incorrect matches obtained when 95% of the true matches are found, as in [30].

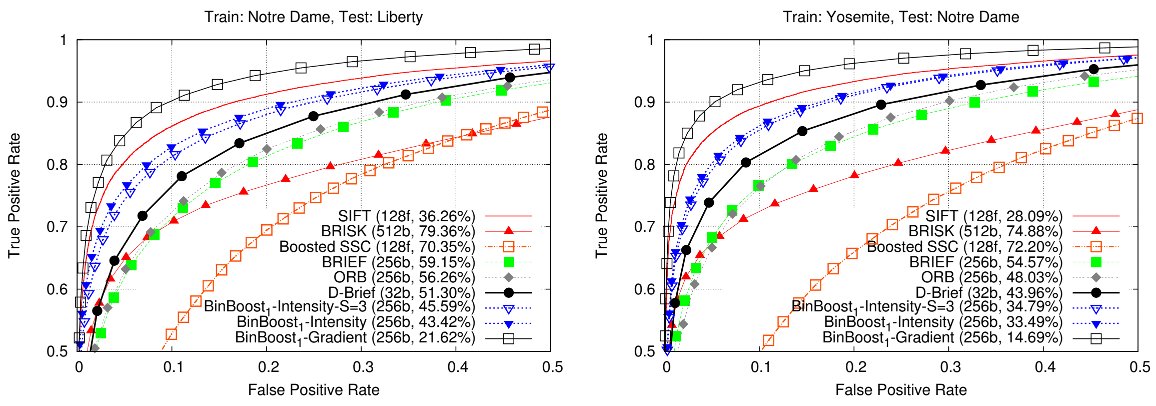


Figure 4.4: Performance of BinBoost$_1$-256 with different weak learner types compared with the state-of-the-art binary descriptors and SIFT as a baseline. Out of all descriptors based on intensity tests, BinBoost-Intensity performs the best. This shows that our framework allows to effectively optimize over the other state-of-the-art binary descriptors and boost their performances at no additional computational cost. Nevertheless, the performance of BinBoost-Intensity cannot match that of floating-point SIFT which is outperformed when using the more discriminative gradient-based weak learners (BinBoost-Gradient).

4.5.2 Weak learner types

To analyze the impact of the weak learner type on descriptor performances, we train a BinBoost$_1$-256 descriptor where each bit corresponds to one weak learner. For our gradient-based descriptor we use $q = 8$ orientation bins, as this is equal to the number of bins proposed for SIFT.

First, we compared the sampling patterns employed in the state-of-the-art binary intensity-based descriptors, such as BRIEF, BRISK and ORB, with the pooling learned with our framework when using intensity-based weak learners. Fig. 4.3 shows the visualization of intensity tests and heat maps of the spatial weighting employed by each descriptor. For BRIEF, intensity tests are from an isotropic Gaussian distribution with the origin of the coordinate system located at the patch center [34]. By contrast, the sampling pattern of BRISK is deterministic. The intensity tests of ORB are selected to increase the variance of the responses, while reducing their correlation. This results in a pronounced vertical trend which can also be seen in the case of BinBoost. Nevertheless, the heat maps show that the tests for BinBoost-Intensity are more dense around the center of the patch, similar to BRIEF, while the ones used in ORB present a more uniform distribution.

To evaluate the influence of the weak learner type on performance, in Fig. 4.4 we compared the results obtained with BinBoost-Intensity and BinBoost-Gradient with those of Boosted SSC, BRIEF, ORB, BRISK, D-Brief [227] and SIFT. The performance of Boosted SSC remains inferior to the other descriptors as the weak learners proposed in [193] rely on thresholding single pixel intensity values and do not provide enough discriminative power. Our experiments show that even though BinBoost-Intensity with variable Gaussian kernel size performs the best out of all the intensity-based descriptors, it is only slightly better than BinBoost-Intensity with filter size equal to 3. As shown in Fig. 4.5, our learning framework does not find a clear correlation between the size of the smoothing kernel and the distance to the patch center, contrary to the sampling pattern of BRISK. Interestingly, even though the optimized sampling scheme of ORB resembles this of BinBoost-Intensity, our framework improves the results over BRIEF much more than ORB. This may be explained when looking at the spatial weighting employed by BinBoost and ORB, where we can see that certain parts of the patch are much more densely sampled in the case of BinBoost, whereas the sampling scheme of ORB is rather uniform.

Nevertheless, BinBoost-Intensity cannot match the performance of SIFT as the discriminative power of the underlying weak learners is not sufficient. When using gradient-based weak learners, we are able to outperform 128-dimensional floating-point SIFT with only 256 bits. Since the performance of gradient-based weak learners remains superior to the intensity-based learners, we use only the former to compute our BinBoost descriptor.

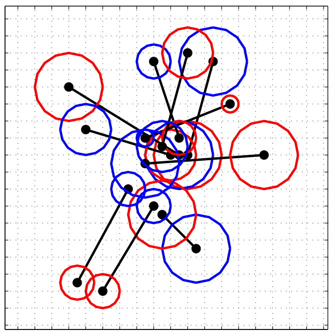


Figure 4.5: Visualization of the first ten intensity-based weak learners with variable kernel size S trained on Liberty dataset. When optimizing on both the pixel positions and the sizes of the Gaussian kernels, our boosting framework does not yield a clear pattern, in particular there is no clear correlation between the size of the smoothing kernel and the distance to the patch center, contrary to the sampling pattern proposed for BRISK. It nevertheless outperforms BRISK in our experiments.

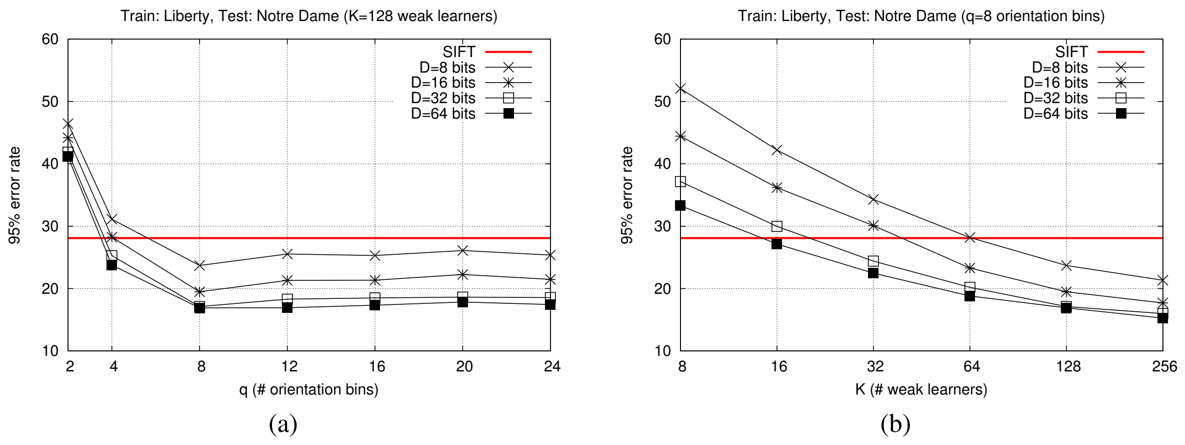


Figure 4.6: Influence of **(a)** the number of orientation bins q and **(b)** the number of weak learners K on the descriptor performance for dimensionalities $D = 8, 16, 32, 64$ bits. The performances are optimal with $q = 8$ orientation bins, which is also the number used in SIFT. Increasing the number of weak learners K from $K = 128$ to $K = 256$ provides only a minor improvement—at greatly increased computational cost—and, hence, we choose for our final descriptor $K = 128$.

4.5.3 Numerical parameters

Our boosting framework defines a generic optimization strategy that unlike many previous approaches, such as [30], does not require fine tuning of multiple parameters. BinBoost has only three main parameters that provide a clear trade-off between the performance and complexity of the final descriptor: the number of orientation bins used by the weak learners, the number of weak learners, and the final dimensionality of the descriptor. We study below the influence of each of them on the performance of our descriptor.

Number of orientation bins q defines the granularity of the gradient-based weak learners. Fig. 4.6(a) shows the results obtained for different values of q and D. For most values of D, the performance is optimal for $q = 8$ as finer orientation quantization does not improve the performance and we keep $q = 8$ in the remaining experiments. Interestingly, this is also the number of orientation bins used in SIFT.

Number of weak learners K determines how many gradient-based features are evaluated per dimension and in Fig. 4.6(b) we show the 95% error rates for different values of K. Increasing the value of K results in increased computational cost and since performance seems to saturate after $K = 128$, we keep this value for our final descriptor.

Dimensionality D is the number of bits of our final descriptor. Fig. 4.7 shows that with $D = 64$ bits, our descriptor

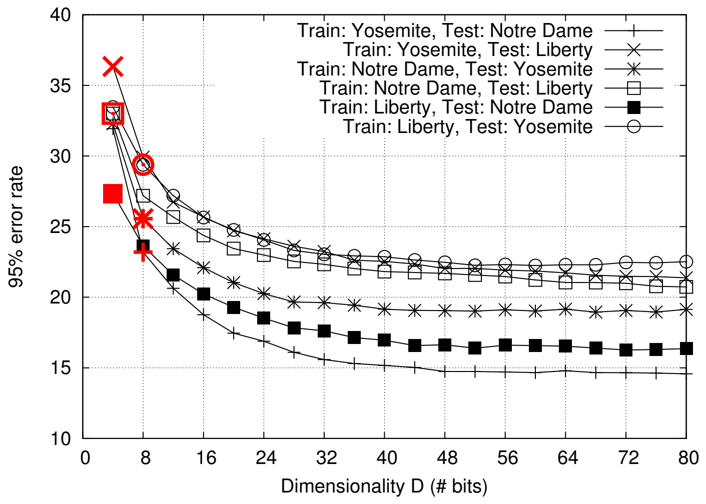


Figure 4.7: Performance for different dimensionalities D. With $D = 64$ bits, BinBoost reaches its optimal performance as increasing the dimensionality further does not seem to improve the results. In bold red we mark the dimensionality for which BinBoost starts outperforming SIFT.

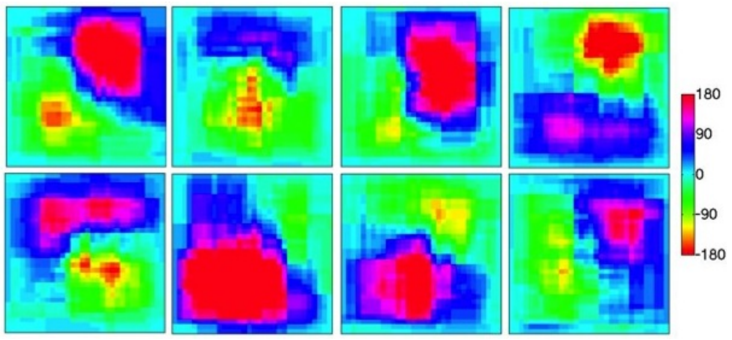


Figure 4.8: Visualization of the selected weak learners for the first 8 bits learned on 200k pairs of 32×32 patches from the Notre Dame dataset (best viewed on screen). For each pixel of the figure we show the average orientation weighted by the weights of the weak learners $\mathbf{b}_d$. For different bits, the weak learners cluster about different regions and orientations illustrating their complementary nature.

reaches its optimal performance as increasing the dimensionality further does not seem to improve the results.

Using these parameters we trained our compact BinBoost descriptor on the Notre Dame dataset. A visualization of the learned weighting and pooling configuration is shown in Fig. 4.8 for the first 8 bits of the descriptor. The weak learners of similar orientations tend to cluster about different regions for each bit thus illustrating the complementary nature of the learned hash functions.

4.6 Results

In this section we provide an extensive comparison of our method against the state-of-the-art descriptors on the Brown [30] and Mikolajczyk [150] datasets. We also show the performance our descriptor for performing visual search on the UKBench dataset [158].

We compare our approach against SIFT [140], SURF [18], the binary LDAHash descriptor [207], Boosted SSC [193], the binary ITQ descriptor applied to SIFT [77], and the fast binary BRIEF [34], ORB [182] and BRISK [134] descriptors.

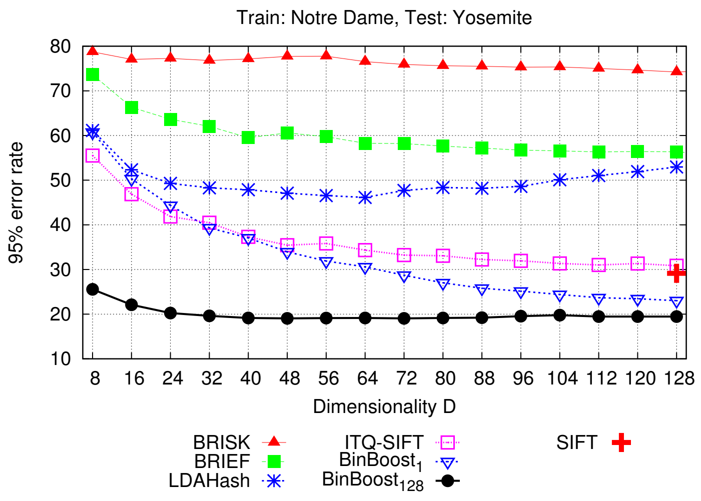


Figure 4.9: 95% error rates for binary descriptors of different dimensionality. For reference, we plot the results obtained with SIFT. BinBoost outperforms the state-of-the-art binary descriptors and the improvement is especially visible for lower dimensionality.

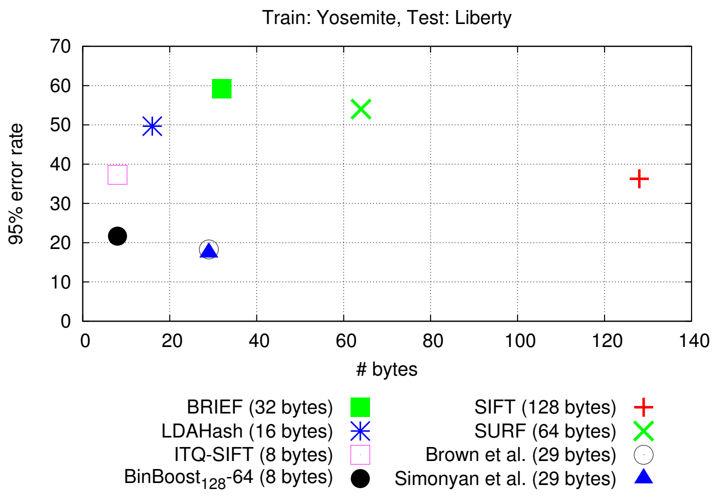


Figure 4.10: Descriptor performances as a function of their memory footprint. For floating-point descriptors we assume 1 byte per dimension as this quantization was reported as sufficient for SIFT [230]. Our BinBoost descriptor offers a significantly lower memory footprint over its floating-point descriptors while providing competitive performances.

4.6.1 Implementation

For SIFT, we use the publicly available implementation of A. Vedaldi [230]. For SURF, LDAHash, BRIEF, BRISK, ORB and ITQ we use the implementation available from their authors. For the other methods, we use our own implementation or we report the results from the literature. For Boosted SSC, we use 128 dimensions as this obtained the best performance. When matching the descriptors we use a fast `POPCOUNT`-based implementation for computing Hamming distances between binary descriptors and matched floating-point descriptors using their Euclidean distance.

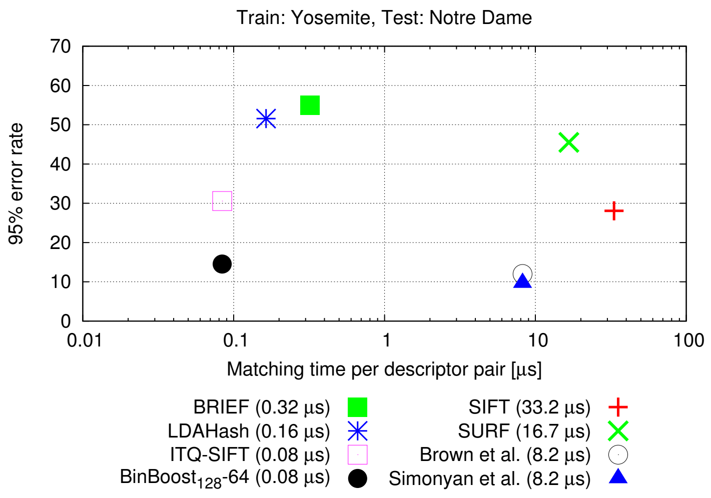


Figure 4.11: Descriptor performances as a function of their matching times. The reported times were computed from 100k test pairs (*i.e.* 100k distance computations were performed) on a Macbook Pro with an Intel i7 2.66 GHz CPU using the POPCOUNT instruction and averaged over 100 runs. To make the comparison fair, we optimized the matching strategy for floating-point descriptors by representing them with unsigned characters. The advantage of binary descriptors, out of which BinBoost performs the best in terms of 95% error rate, is clear.

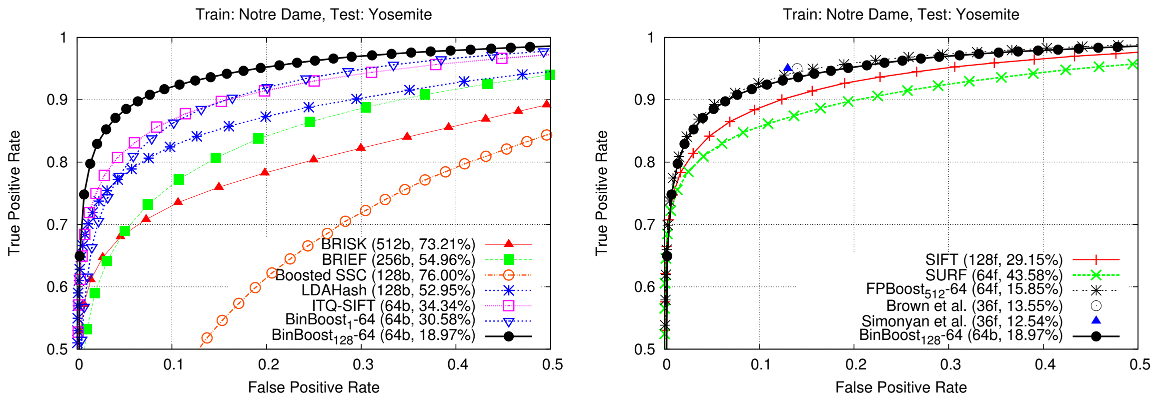


Figure 4.12: Comparison of our BinBoost descriptor to the state-of-the-art binary (**left**) and floating-point (**right**) descriptors. In parentheses: the number of floating-point (f) or binary (b) dimensions and the 95% error rate. Our BinBoost descriptor significantly outperforms its binary competitors across all false positive rates. It also outperforms SIFT and provides similar performances to the recent floating-point descriptors, even though it is much faster to match and has a lower memory footprint.

4.6.2 Brown datasets

We first compare our method using the Liberty, Notre Dame and Yosemite datasets [30] according to the evaluation protocol described in Sec. 4.5.1. Fig. 4.12 shows the ROC curves for BinBoost and the state-of-the-art methods. Table 4.1 summarizes the 95% error rates. Both show that BinBoost significantly outperforms the baselines. It performs almost twice as well as SIFT in terms of 95% error rate, while requiring only 64 bits (8 bytes) instead of 128 bytes for SIFT. Moreover, since BinBoost can be efficiently implemented using integral images, the computation time of our descriptor is comparable with that of SIFT using Vedaldi's implementation—approximately 1ms per descriptor on a Macbook Pro with an Intel i7 2.66 GHz CPU. The performance improvement of BinBoost with respect to the recent binary descriptors, such as LDAHash or BRIEF, is even greater, BinBoost achieving a 95% error rate that is almost a factor of 3 lower than that obtained with these methods.

Since the dimensionality of the other binary descriptors can be varied depending on the required performance quality, Fig. 4.9 compares the 95% error rates of these descriptors for different numbers of bits used. BinBoost

		Binary						Floating-point				
Train	**Test**	BinBoost$_{128}$-64 *8 bytes*	BinBoost$_1$-64 *8 bytes*	ITQ-SIFT [77] *8 bytes*	LDAHash [207] *16 bytes*	BRIEF *32 bytes*	BRISK *64 bytes*	SURF *64 bytes*	SIFT *128 bytes*	FPBoost$_{512}$-64 *64 bytes*	Brown [30] *29 bytes*	Simonyan [197] *29 bytes*
Yosemite	Notre Dame	**14.54**	26.80	30.56	51.58	54.57	74.88	45.51	28.09	14.80	11.98	9.67
Liberty		**16.90**	29.60	31.07						14.68	-	-
Yosemite	Liberty	**21.67**	33.54	37.31	49.66	59.15	79.36	54.01	36.27	22.39	18.27	17.44
Notre Dame		**20.49**	31.90	36.95						17.90	16.85	14.51
Notre Dame	Yosemite	**18.97**	30.58	34.34	52.95	54.96	73.21	43.58	29.15	15.85	13.55	12.54
Liberty		**22.88**	38.13	34.43						20.85	-	-

Table 4.1: 95% error rates for different training and testing configurations and the corresponding results for BinBoost with 64 and 8 bits and its competitors. For the descriptors that do not depend on the training data, we write one result per testing dataset, for others we give the results for two different training datasets. Below the descriptor names we write the number of bytes used to encode them. For the floating point descriptors (SIFT, SURF, FPBoost, Brown *et al.* [30], Simonyan *et al.* [197]) we assume 1 byte per dimension, as this quantization was reported as sufficient for SIFT [230]. BinBoost significantly outperforms its binary competitors, while requiring less memory. For reference, we also give the results of the floating-point descriptors: BinBoost performs similarly to the best floating-point descriptors even though it is shorter and binary which enables a significant speedup in matching time as shown in Fig. 4.11.

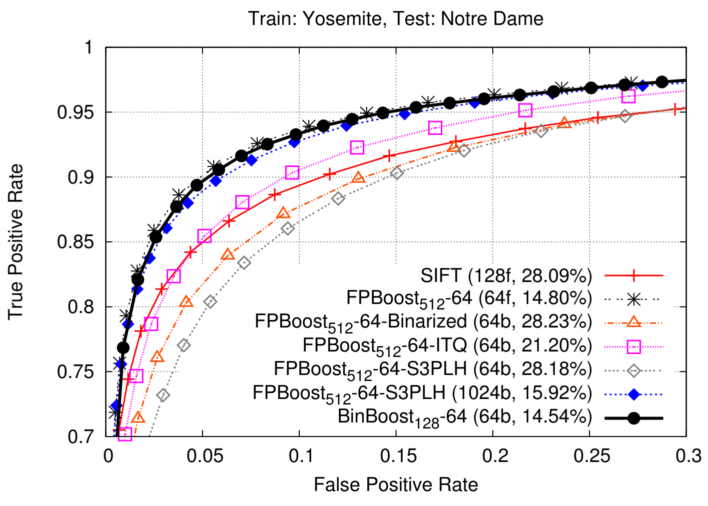


Figure 4.13: Performance of our BinBoost descriptor compared with different binarization methods applied on FPBoost. Binarizing the discriminative projections found with FPBoost either by simple thresholding or with Iterative Quantization (ITQ) results in large binarization errors significantly reducing its accuracy. On the other hand, the sequential projection learning of S3PLH requires a fairly large number of bits to recover the original performance of FPBoost. In contrast, by jointly optimizing over the feature weighting and pooling strategy of each bit, our BinBoost approach results in a highly compact and accurate binary descriptor whose performance is similar with FPBoost but at a fraction of the storage cost.

clearly outperforms them across all dimensions at the lower end of the spectrum. However, the biggest improvement can be seen for lower dimensionality.

Moreover, our BinBoost descriptor remains competitive to the best descriptors of [30] and [197], even though the memory footprint of their descriptors is almost 4 times greater as shown in Fig. 4.10. The real advantage of BinBoost, however, is that it allows for extremely fast similarity computation using the Hamming distance[2], whereas the descriptors of [30] and [197] are floating-point and cannot benefit from the same optimization, even when quantized very coarsely. As presented in Fig. 4.11, this results in a speedup of over 2 orders of magnitude in terms of similarity search.

[2]On modern CPUs this can be implemented as a bitwise XOR operation on the descriptors followed by a `POPCOUNT` instruction which counts the number of bits set to 1.

To verify the performance of our descriptor, we also compare it to several binarization techniques applied to FPBoost. Results are displayed in Fig. 4.13. Binarizing the FPBoost coordinates by thresholding them at an optimal threshold found as in [207] results in large binarization errors significantly decreasing the accuracy of the resulting binary representation. This error can be reduced using Iterative Quantization [77], however, the orthogonality constraints used in this approach largely limit the extent to which it can be minimized. In contrast, sequential projection learning (S3PLH) [240] can find non-orthogonal projections that more faithfully mitigate binarization error, however, it requires a fairly large number of bits to recover FPBoost's original performance. Unlike these methods, by effectively combining multiple weak learners within each hash function, our algorithm results in a more accurate predictor with far fewer bits.

4.6.3 Mikolajczyk dataset

We tested the generalization performance of our descriptor when trained on the Brown datasets and evaluated on the significantly different Mikolajczyk dataset [150]. We report results using the Notre Dame dataset, however, similar results were found for all the Brown datasets. We followed the evaluation protocol of [150] that compares descriptors using a single keypoint detector, and used the OpenCV SURF Hessian-based detector. For each image pair we detect 1000 keypoints per image and match them using exhaustive search. We then filter outliers using a distance ratio threshold of 0.8 as in [140]. We evaluate each descriptor in terms of the recognition rate which is the number of correctly matched keypoints.

Fig. 4.14 shows the results obtained for the `bark, boat, graf, trees, ubc` and `wall` sequences. In all the sequences BinBoost$_1$-256 and FPBoost$_{512}$-64 outperform the other descriptors. The performance of BinBoost$_{128}$-64, however, does not perform as well as when evaluated on the Brown datasets, which indicates that there is an inherent efficiency tradeoff when training on a different condition. Nonetheless, the extended BinBoost$_{128}$-128 and BinBoost$_{128}$-256 descriptors outperform the other methods while being shorter or of the same length.

4.6.4 Visual Search on UKBench

We further evaluate our approach for performing visual search using the University of Kentucky Benchmark (UKBench) dataset [158] that contains over 10k images of 2600 objects, each object being depicted in 4 images taken from different viewpoints. As in other approaches, we first build a database of the almost one million descriptors extracted from all the dataset images. For each query image, we then search for the nearest neighbors in the database using their associated keypoint descriptors to vote for the most similar images in the database. Finally, we sort the database images according to the number of votes they receive and retrieve those associated with the highest number of votes. As with our previous experiments, we consider descriptors trained using the Notre Dame dataset with similar results seen for the other Brown datasets. In our evaluation we randomly selected 500 query images from the dataset and use the remaining 10k images to create a database. We ran the experiment three times and report the average results along with the standard deviation values.

Table 4.2 summarizes the results we obtained for different descriptors. To evaluate the performance we report mean average precision (mAP) and percentage of correct number of images retrieved at the top of the list (Correct@1). Out of all the evaluated descriptors BinBoost$_{128}$-256 performs the best followed by BinBoost$_1$-256. FPBoost performs slightly worse than BinBoost, while still outperforming SIFT and other intensity-based binary descriptors. Overall, the boosted keypoints descriptors provide the best performance of all the tested descriptors, even though they were trained on a significantly different dataset.

4.7 Face Descriptors

In this section we evaluate our method for matching face images. While this constitues a rather different problem than modeling the appearance of local keypoints, as we show our method is generic and can be easily adapted to new application domains. For our evaluation we used a dataset of face images [255] that consists of faces imaged under different viewpoints. From this dataset we created two sets of 100k and 200k pairs of images. Similarly to Liberty, Notre Dame and Yosemite datasets, each set is balanced and contains an equal number of image pairs belonging to the same person as those of different people. We used the 200k dataset to train our descriptors and the 100k set to test them.

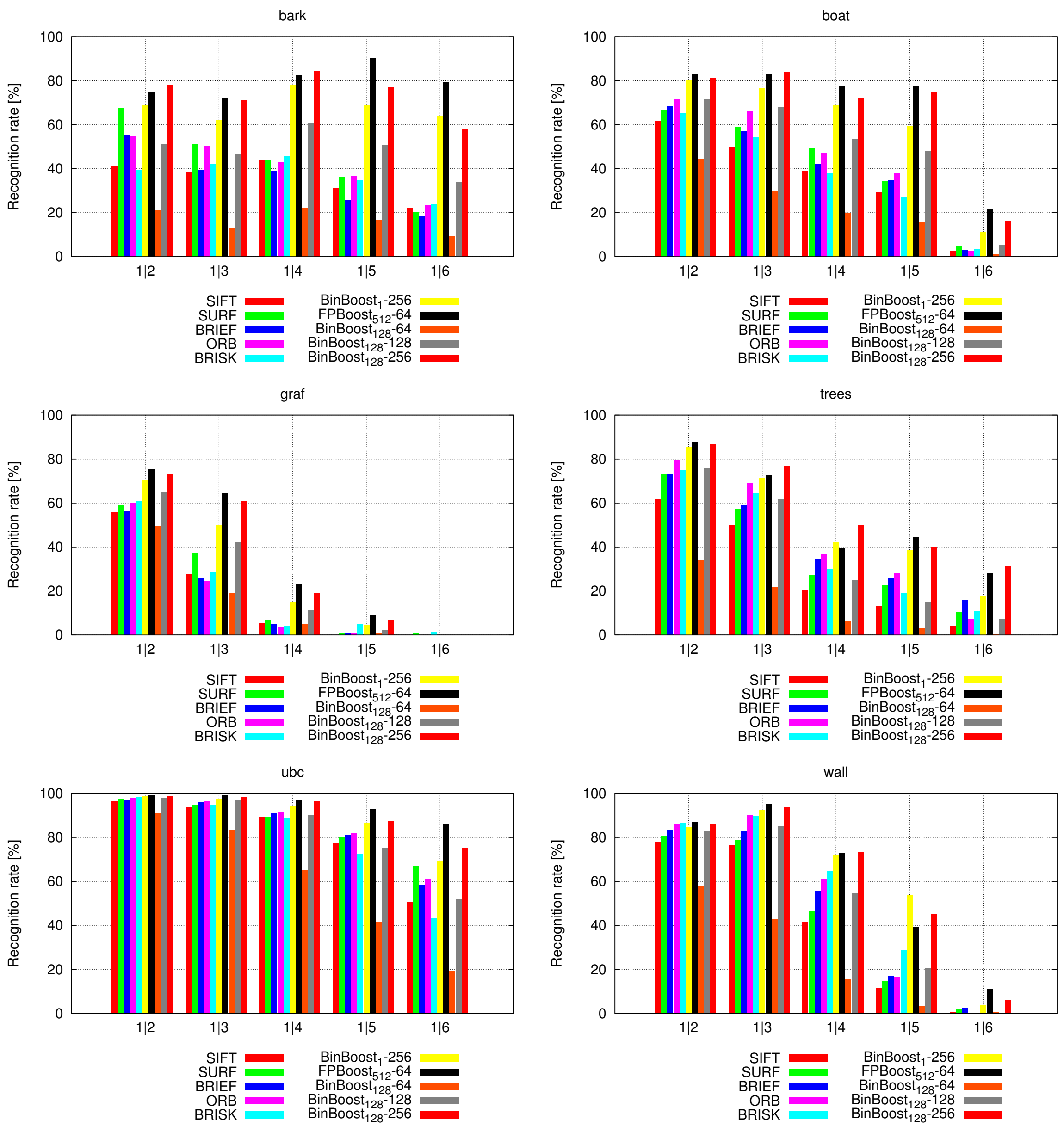


Figure 4.14: Recognition rates for the Mikolajczyk dataset. The proposed BinBoost and FPBoost descriptors significantly outperform the competitors, both binary and floating-point, while being shorter or of the same length. This shows that, although generalization of the learned descriptors remains a challenging task, they can still perform well under conditions that differ greatly from the training conditions.

Fig. 4.15 compares the learned spatial weightings obtained with the Brown and Faces datasets. When we train our BinBoost descriptor on the images extracted around interest points, the weak learners clearly concentrate around the center of the patch. In fact, the obtained weighting closely resembles the Gaussian weighting employed by SIFT. In contrast, for face images the weak learners concentrate about the lower and upper image regions that correspond to the location of the eyes and mouth, and as also observed in [2] constitute discriminative facial features. This further demonstrates the flexibility of our approach and its ability to adapt to new types of image data.

Fig. 4.16 shows the qualitative results obtained using BinBoost$_1$-256. BinBoost remains largely invariant to the significant viewpoint and intensity changes present in this dataset, while still being able to discriminate between different people. Most of the mis-classifications are due to occlusions and extreme viewpoint variation such as side views.

Descriptor	**mAP** $\pm\sigma$	**Correct@1** $\pm\sigma$
BRISK	0.402 ± 0.006	61.830 ± 0.884
ORB	0.418 ± 0.005	64.902 ± 1.931
SIFT	0.455 ± 0.008	68.235 ± 2.183
BRIEF	0.457 ± 0.014	68.562 ± 0.493
FPBoost$_{512}$-64	0.476 ± 0.006	70.000 ± 1.709
BinBoost$_{128}$-128	0.493 ± 0.017	72.222 ± 2.747
BinBoost$_{1}$-256	0.533 ± 0.010	76.144 ± 2.467
BinBoost$_{128}$-256	$\mathbf{0.556} \pm \mathbf{0.008}$	$\mathbf{79.216} \pm \mathbf{1.870}$

Table 4.2: Results of visual search on the UKBench dataset [158]: mean average precision (mAP) and percentage of correctly retrieved images at the first position (Correct@1) are reported. Average results are shown across three random train and test splits along with the standard deviation. BinBoost$_{128}$-256 outperforms the other descriptors, even though it is trained on the Notre Dame dataset. The other learned descriptors, namely BinBoost$_{1}$-256 and FPBoost$_{512}$-64, achieve worse results, though their performance is still better than SIFT and the other intensity-based descriptors.

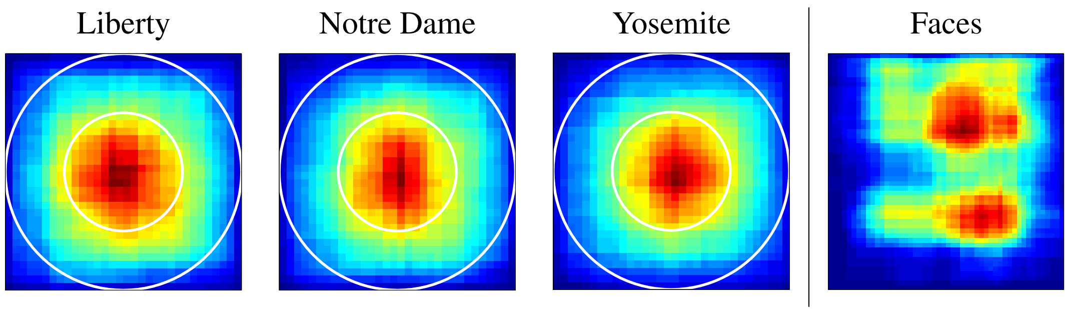


Figure 4.15: Learned spatial weighting obtained with BinBoost$_{1}$-256 trained on the Liberty, Notre Dame, Yosemite and Faces datasets. For the first three datasets, the learned weighting closely resembles the Gaussian weighting employed by SIFT (white circles indicate $\sigma/2$ and σ used by SIFT). However, the learned spatial weighting on the Faces dataset is focused about the eyes and mouth that constitute discriminative facial features.

In Fig. 4.17 we plot the quantitative results of BinBoost$_{1}$-256, FPBoost$_{256}$-64 and BinBoost$_{128}$-64 descriptors compared with LBP, a widely used face descriptor [2]. Our boosted descriptors result in a significant improvement over the baseline. Furthermore, compared with LBP our FPBoost descriptor achieves a reduction in 95% error rate by more than a factor of 2. Similar to [37, 244] this demonstrates the potential advantages of exploiting image data to learn a face descriptor. More importantly, it illustrates the flexibility of our approach beyond local keypoint descriptors.

4.8 Conclusion

In this chapter we presented an efficient framework to train highly discriminative and compact local feature descriptors that leverages the boosting-trick to simultaneously optimize both the weighting and sampling strategy of a set of non-linear feature responses. We first showed how boosting can be used to result in an accurate yet compact floating-point descriptor. We then considered a binary extension of our approach that shares a similar accuracy but operates at a fraction of the matching and storage cost. We explored the use of both intensity- and gradient-based features within our learning framework and performed an evaluation across a variety of descriptor matching tasks. In each task, our approach achieved a signficant improvement over the state-of-the-art descriptors, such as BRIEF and BRISK, in both accuracy and efficiency by optimizing their sampling patterns. Finally, we showed that our method can be easily generalized to new applicaiton domains, such as faces.

During his PhD, Tomasz Trzcinki did an internship at Google Zurich, in the Computer Vision team headed by Henrik Stewenius, which works on image retrieval. During this internship, Tomasz implemented BinBoost with Google's software framework.

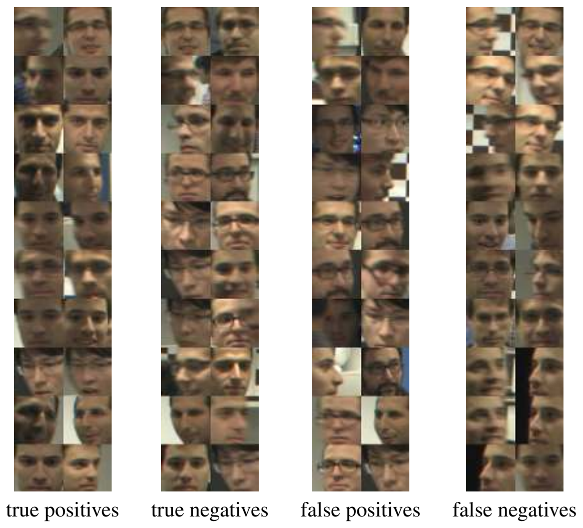


Figure 4.16: Matching results on the Faces dataset using our 256-bit BinBoost$_1$-256 at the 95% error rate, *i.e.* when 95% of the positive image pairs are correctly classified. BinBoost remains robust to significant viewpoint changes and motion blur. The mis-classified examples are mostly due to occlusion and extreme variations in viewpoint such as side views.

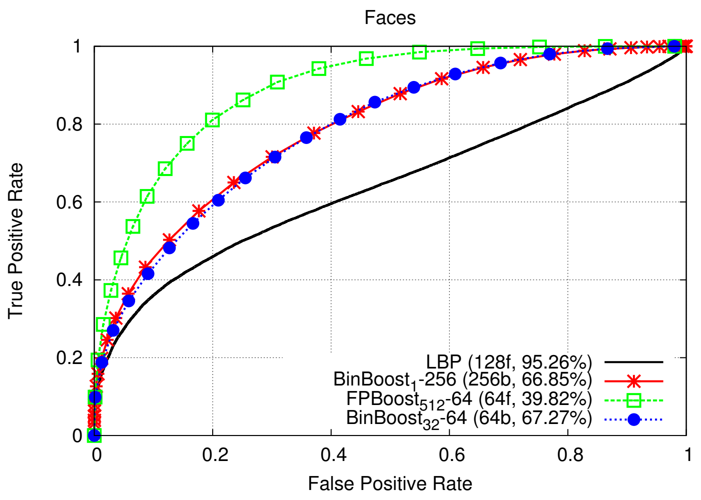


Figure 4.17: The performance of our boosted descriptors on the Faces dataset compared with the commonly used LBP face descriptor [2]. BinBoost$_1$-256 significantly outperforms LBP. Similarly to the results obtained for local feature descriptors, we can see that BinBoost$_{128}$-64 performs equally to BinBoost$_1$-256, but with only 64 bits per descriptor. FPBoost performs even better with the 95% error rate reduced by more than twice compared with the LBP baseline.

Chapter 5

TILDE: A Temporally Invariant Learned DEtector

5.1 Introduction

While the previous chapters focus on local descriptors of feature points, this chapter considers their detections. Existing hand-crafted keypoint detectors exhibit excellent repeatability when the scale and viewpoint change or the images are blurred [150]. However, their reliability degrades significantly when the images are acquired outdoors at different times of day and in different weathers or seasons, as shown in Fig. 5.1. This is a severe handicap when attempting to match images taken in fair and foul weather, in the morning and evening, in winter and summer, even with illumination invariant descriptors [83, 217, 84, 245].

In this chapter, we present the approach we initially published in [232] and developed in collaboration with Yannick Verdie and Kwang Moo Yi, two young postdocs. In this approach, we learn a keypoint detector that extracts keypoints which are stable under such challenging conditions and allow matching in situations as difficult as the one depicted by Fig. 5.1. To this end, we first introduce a simple but effective method to identify potentially stable points in training images. We then use them to train a regressor that produces a score map whose values are local maxima at these locations. By running it on new images, we can extract keypoints with simple non-maximum suppression.

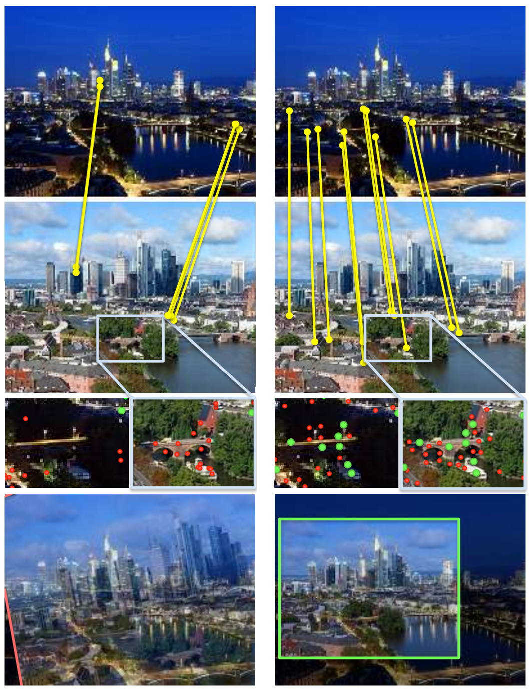
Learning methods have been used before in the context of keypoint detection [180, 203] to reduce the number of operations required when finding the *same* keypoints as handcrafted methods. However, in spite of an extensive literature search, we have only found one method [210] that attempts to improve the repeatability of keypoints by learning. This method focuses on learning a classifier to filter out initially detected keypoints but achieved limited improvement. This may be because their method was based on pure classification and also because it is non-trivial to find good keypoints to be learned by a classifier in the first place.

Our approach is inspired by an algorithm we proposed in [199] that relies on regression to extract centerlines from images of linear structures. Using this idea for our purposes has required us to develop a new kind of regressor that is robust to complex appearance variation so that it can efficiently and reliably process the input images.

Before we published [232], there was no standard benchmark dataset designed to test the robustness of keypoint detectors to these kinds of temporal changes. We therefore created our own from images from the Archive of Many Outdoor Scenes (*AMOS*) [110] and our own panoramic images to validate our approach. We will use our dataset in addition to the standard *Oxford* [150] and *EF* [260] datasets to demonstrate that our approach significantly outperforms state-of-the-art methods in terms of repeatability.

In summary, our contribution in [232] is threefold:

- We introduce a "Temporally Invariant Learned DEtector" (TILDE), a new regression-based approach to extracting feature points that are repeatable under drastic illumination changes causes by changes in weather, season, and time of day.
- We propose an effective method to generate the required training set of "good keypoints to learn."



(a) With SURF [17] keypoints (b) With our keypoints

Figure 5.1: Image matching example using Speeded-Up Robust Features (SURF) [17] and our method. *Same number of keypoints* and descriptor [140] was used for both keypoint detectors. Detected keypoints are shown in the third row, with the repeated ones in green. For SURF, only one keypoint detected in the daytime image was detected in the nighttime image. Our method on the other hand returns many common keypoints regardless of the drastic lighting change.

- We created a new benchmark dataset for evaluation of feature point detectors on outdoor images captured at different times ands seasons.

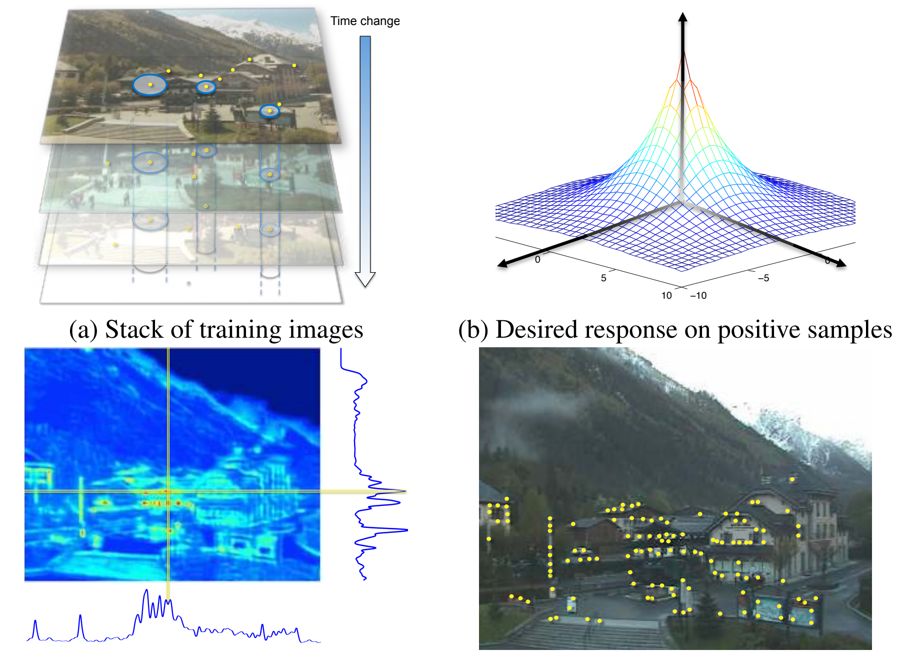
In the remainder of this chapter, we first discuss related work, give an overview of our approach, and then detail our regression-based approach. We finally present the comparison of our approach to state-of-the-art keypoint detectors.

5.2 Related Work

Handcrafted Keypoint Detectors An extraordinary large amount of work has been dedicated to developing ever more effective feature point detectors. Even though the methods that appeared in the 1980s [153, 69, 87] are still in wide use, many new ones have been developed since. [68] proposed the SFOP detector to use junctions as well as blobs, based on a general spiral model. [91] and the WADE detector of [186] use symmetries to obtain reliable keypoints. With SIFER and D-SIFER, [143, 142] used Cosine Modulated Gaussian filters and 10th order Gaussian derivative filters for more robust detection of keypoints. Edge Foci [260] and [82] use edge information for robustness against illumination changes. Overall, these methods have consistently improved the performance of keypoint detectors on the standard dataset [150], but still suffer severe performance drop when applied to outdoor scenes with temporal differences.

One of the major drawbacks of handcrafted methods are that they cannot be easily adapted to the context, and consequently lack flexibility. For instance, SFOP [68] works well when calibrating cameras and WADE [186] shows good results when applied to objects with symmetries. However, their advantages are not easily carried on to the problem we tackle here, such as finding similar outdoors scenes [126].

Learned Keypoint Detectors Although work on keypoint detectors were mainly focused on handcrafted methods, some learning based methods have already been proposed [180, 210, 89, 175]. With FAST, [180] introduced



(a) Stack of training images (b) Desired response on positive samples

(c) Regressor response for a new image (d) Keypoints detected in the new image

Figure 5.2: Overview of our approach. We rely on a stack of training images, captured from the same viewpoint but under different illuminations (a), and a simple method to select good keypoints to learn. We train a regressor on image patches to return peaked values like in (b) at the keypoint locations, and small values far from these locations. Applying this regressor to each patch of a new image gives us a score map such as the one in (c), from which we can extract keypoints as in (d) by looking for local maxima with large values.

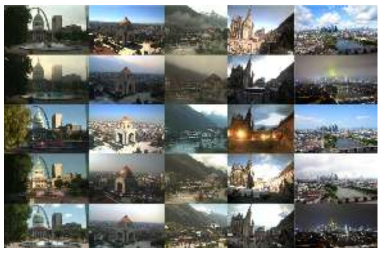
Machine Learning techniques to learn a fast corner detector. However, learning in their case was only aimed toward the speed up of the keypoint extraction process. Repeatability is also considered in the extended version FAST-ER [181], but it did not play a significant role. [210] trained the WaldBoost classifier [202] to learn keypoints with high repeatability on a pre-aligned training set, and then filter out an initial set of keypoints according to the score of the classifier. Their method, called TaSK, is probably the most related to our method in the sense that they use pre-aligned images to build the training set. However, the performance of their method is limited by the initial keypoint detector used.

Recently, [89] proposed to learn a classifier which detects *matchable* keypoints for Structure-from-Motion (SfM) applications. They collect *matchable* keypoints by observing which keypoints are retained throughout the SfM pipeline and learn these keypoints. Although their method shows significant speed-up, they remain limited by the quality of the initial keypoint detector. [175] learns convolutional filters through random sampling and looking for the filter that gives the smallest pose estimation error when applied to stereo visual odometry. Unfortunately, their method is restricted to linear filters, which are limited in terms of flexibility, and it is not clear how their method can be applied to other tasks than stereo visual odometry.

We propose a generic scheme for learning keypoint detectors, and a novel efficient regressor specified for this task. We will compare it to state-of-the-art handcrafted methods as well as TaSK, as it is the closest method from the literature, on several datasets.

5.3 Learning a Robust Keypoint Detector

In this section, we first outline our regression-based approach briefly and then explain how we build the required training set. We will formalize our algorithm and describe the regressor in more details in the following section.


(a) Sample images of the selected scenes from *AMOS*

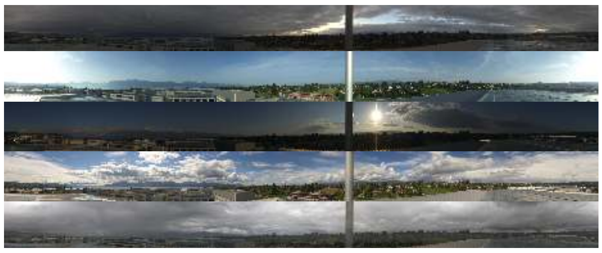

(b) Sample images of the *Panorama* sequence

Figure 5.3: Example figures from the *Webcam* dataset. The *Webcam* dataset is composed of six scenes from various locations: (a) five scenes taken from the Archive of Many Outdoor Scenes (*AMOS*) dataset [110], namely *StLouis*, *Mexico*, *Chamonix*, *Courbevoie*, and *Frankfurt*. (b) *Panorama* scenes from the roof of a building which shows a 360 degrees view.

5.3.1 Overview of our Approach

Let us first assume that we have a set of training images of the same scene captured from the same point of view but at different seasons and different times of the day, such as the set of Fig. 5.3(a). Let us further assume that we have identified in these images a set of locations that we think can be found consistently over the different imaging conditions. We propose a practical way of doing this in Section 5.3.2 below.

Let us call *positive samples* the image patches centered at these locations in each training image. The patches far away from these locations are *negative samples*.

To learn to find these locations in a new input image, we propose to train a regressor to return a value for each patch of a given size of the input image. These values should have a peaked shape similar to the one shown in Fig. 5.2(b) on the positive samples, and we also encourage the regressor to produce a score that is as small as possible for the negative samples. As shown in Fig. 5.2(c), we can then extract keypoints by looking for local maxima of the values returned by the regressor, and discard the image locations with low values by simple thresholding. Moreover, our regressor is also trained to return similar values for the same locations over the stack of images. This way, the regressor returns consistent values even when the illumination conditions vary.

5.3.2 Creating the Training Set

As shown in Fig. 5.3, to create our dataset of positive and negative samples, we first collected series of images from outdoor webcams captured at different times of day and different seasons. We identified several suitable webcams from the *AMOS* dataset [110]—webcams that remained fixed over long periods of time, protected from the rain, etc. We also used panoramic images captured by a camera located on the top of a building.

To collect a training set of positive samples, we first detect keypoints independently in each image of this dataset. We use SIFT [140], but other detectors could be considered as well. We then iterate over the detected keypoints, starting with the keypoints with the smallest scale. If a keypoint is detected at about the same location in most of the images from the same webcam, its location is likely to be a good candidate to learn.

In practice we consider that two keypoints are at about the same location if their distance is smaller than the scale estimated by SIFT and we keep the best 100 repeated locations. The set of positive samples is then made of the patches from *all* the images, including the ones where the keypoint was not detected, and centered on the average location of the detections.

This simple strategy offers several advantages: we keep only the most repeatable keypoints for training, discarding the ones that were detected only infrequently. We also introduce as positive samples the patches where a highly repeatable keypoint was missed. This way, we can focus on the keypoints that can be detected reliably under different conditions, and correct the mistakes of the original detector.

To create the set of negative samples, we simply extract patches at locations that are far away from the keypoints used to create the set of positive samples.

5.4 An Efficient Piece-wise Linear Regressor

In this section, we first introduce the form of our regressor, which is made to be applied to every patch from an image efficiently, then we describe the different terms of the proposed objective function to train for detecting keypoints reliably, and finally we explain how we optimize the parameters of our regressor to minimize this objective function.

5.4.1 A Piece-wise Linear Regressor

Our regressor is a piece-wise linear function expressed using Generalized Hinging Hyperplanes (GHH) [29, 243]:

$$\mathbf{F}(\vec{\mathbf{x}};\vec{\omega}) = \sum_{n=1}^{N} \delta_n \max_{m=1}^{M} \vec{\mathbf{w}}_{nm}^{\top}\vec{\mathbf{x}} \quad , \tag{5.1}$$

where $\vec{\mathbf{x}}$ is a vector made of image features extracted from an image patch, $\vec{\omega}$ is the vector of parameters of the regressor and can be decomposed into $\vec{\omega} = \left[\vec{\mathbf{w}}_{11}^{\top}, \ldots, \vec{\mathbf{w}}_{MN}^{\top}, \delta_1, \ldots, \delta_N\right]^{\top}$. The $\vec{\mathbf{w}}_{nm}$ vectors can be seen as linear filters. The parameters δ_n are constrained to be either -1 or +1. N and M are meta-parameters which control the complexity of the GHH. As image features we use the three components of the LUV color space and the image gradients—horizontal and vertical gradients and the gradient magnitude—computed at each pixel of the $\vec{\mathbf{x}}$ patches.

[243] showed that any continuous piecewise-linear function can be expressed in the form of Eq. (5.1). It is well suited to our keypoint detector learning problem, since applying the regressor to each location of the image involves only simple image convolutions and pixel-wise maximum operators, while regression trees require random access to the image and the nodes, and CNNs involve higher-order convolutions for most of the layers. Moreover, we will show that this formulation also facilitates the integration of different constraints, including constraints between the responses for neighbor locations, which are useful to improve the performance of the keypoint extraction.

Instead of simply aiming to predict the score computed from the distance to the closest keypoint in a way similar to what was done in [199], we argue that it is also important to distinguish the image locations that are close to keypoints from those that are far away. The values returned by the regressor for image locations close to keypoints should have a local maximum at the keypoint locations, while the actual values for the locations far from the keypoints are irrelevant as long as they are small enough to discard them by simple thresholding. We therefore first introduce a classification-like term that enforces the separation between these two different types of image locations. We also rely on a term that enforces the response to have a local maximum at the keypoint locations, and a term that regularizes the responses of the regressor over time. To summarize, the objective function $\mathcal{L}$ we minimize over the parameters ω of our regressor can be written as the sum of three terms:

$$\underset{\omega}{\text{minimize}} \quad \mathcal{L}_c(\omega) + \mathcal{L}_s(\omega) + \mathcal{L}_t(\omega) \ . \tag{5.2}$$

5.4.2 Objective Function

In this subsection we describe in detail the three terms of the objective function introduced in Eq. (5.2). The individual influences of each term are evaluated empirically and discussed in Section 5.5.4.

Classification-Like Loss $\mathcal{L}_c$ As explained above, this term is useful to separate well the image locations that are close to keypoints from the ones that are far away. It relies on a max-margin loss, as in traditional SVM [45]. In particular, we define it as:

$$\mathcal{L}_c(\omega) = \gamma_c \|\vec{\omega}\|_2^2 + \frac{1}{K}\sum_{i=1}^{K} \max\left(0, 1 - y_i \mathbf{F}\left(\vec{\mathbf{x}}_i;\vec{\omega}\right)\right)^2 \quad , \tag{5.3}$$

where γ_c is a meta-parameter, $y_i \in \{-1,+1\}$ is the label for the sample $\mathbf{x}_i$, and K is the number of training data.

Shape Regularizer Loss $\mathcal{L}_s$ To have local maxima at the keypoint locations, we enforce the response of the regressor to have a specific shape at these locations. For each positive sample i, we force the response shape by defining a loss term related to the desired response shape $\mathbf{h}$, similar to the one used in [199] and shown in Fig. 5.2(b):

$$\mathbf{h}(x,y) = e^{\alpha(1-\frac{\sqrt{x^2+y^2}}{\beta})} - 1 \ , \tag{5.4}$$

where x, y are pixel coordinates with respect to the center of the patch, and α, β meta-parameters influencing the sharpness of the shape.

However, we want to enforce only the general shape and not the scale of the responses to not interfere with the classification-like term $\mathcal{L}_c$. We therefore introduce an additional term defined as:

$$\mathcal{L}_s(\omega) = \frac{\gamma_s}{K_p} \sum_{i|y_i=+1} \sum_n \left\| \mathbf{w}_{n\eta_i(n)} * \mathbf{x}_i - (\overrightarrow{\mathbf{w}}_{n\eta_i(n)}^\top \overrightarrow{\mathbf{x}_i})\mathbf{h} \right\|_2^2 \ , \tag{5.5}$$

where $*$ denotes the convolution product, K_p is the number of positive samples; γ_s is a meta-parameter for weighting the term that will be estimated by cross-validation. $\eta_i(n) = \text{argmax}_m \ \overrightarrow{\mathbf{w}}_{nm}^\top \overrightarrow{\mathbf{x}}_i$ is used to enforce the shape constraints only on the filters that contribute to the regressor response of the $\max$ operator.

It turns out that it is more convenient to perform the optimization of this term in the Fourier domain. If we denote the 2D Fourier transform of $\mathbf{w}_{nm}$, $\mathbf{x}_i$, and $\mathbf{h}$ as $\mathbf{W}_{nm}$, $\mathbf{X}_i$, and $\mathbf{H}$, respectively, then by applying Parseval's theorem and the Convolution theorem, Eq. (5.5) becomes

$$\mathcal{L}_s(\omega) = \frac{\gamma_s}{K_p} \sum_{i|y_i=+1} \sum_n \overrightarrow{\mathbf{W}}_{n\eta_i(n)}^\top \mathbf{S}_i^\top \mathbf{S}_i \overrightarrow{\mathbf{W}_{n\eta_i(n)}} \ , \tag{5.6}$$

where

$$\mathbf{S}_i = \left(\text{diag}\left(\overrightarrow{\mathbf{X}_i}\right) - \overrightarrow{\mathbf{X}}_i \overrightarrow{\mathbf{H}}\right)^\top \ . \tag{5.7}$$

This way of enforcing the shape of the responses is a generalization of the approach of [178] to any type of shape. In practice, we approximate $\mathbf{S}_i$ with the mean over all positive training samples for efficient learning. We also use Parseval's theorem and the feature mapping proposed in Ashraf *et al.*'s work [14] for easy calculation [1].

Temporal Regularizer Loss $\mathcal{L}_t$ To enforce the repeatability of the regressor over time, we force the regressor to have similar responses at the same locations over the stack of training images. This is simply done by adding a term $\mathcal{L}_t$ defined as:

$$\mathcal{L}_t(\omega) = \frac{\gamma_t}{K} \sum_{i=1}^{K} \sum_{j \in \mathcal{N}_i} \left(\mathbf{F}(\overrightarrow{\mathbf{x}}_i; \overrightarrow{\omega}) - \mathbf{F}(\overrightarrow{\mathbf{x}}_j; \overrightarrow{\omega})\right)^2 \ , \tag{5.8}$$

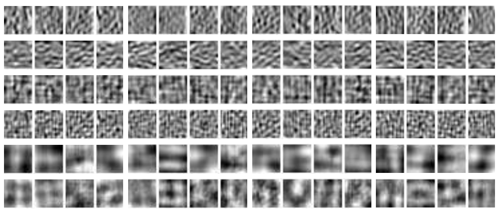
where $\mathcal{N}_i$ is the set of samples at the same image locations as $\mathbf{x}_i$ but from the other training images of the stack. γ_t is again a meta-parameter to weight this term.

5.4.3 Learning the Piece-wise Linear Regressor

Optimization After dimension reduction using Principal Component Analysis (PCA) applied to the training samples to decrease the number of parameters to optimize, we solve Eq. (5.2) through a greedy procedure similar to gradient boosting. We start with an empty set of hyperplanes $\mathbf{w}_{n,m}$ and we iteratively add new hyperplanes that minimize the objective function until we reach the desired number (we use $N = 4$ and $M = 4$ in our experiments). To estimate the hyperplane to add, we apply a trust region Newton method [136], as in the widely-used LibLinear library [61].

After initialization, we randomly go through the hyperplanes one by one and update them with the same Newton optimization method. Fig. 5.4(a) shows the filters learned by our method on the *StLouis* sequence. We perform a simple cross-validation using grid search in log-scale to estimate the meta-parameters γ_c, γ_s, and γ_t on a validation set.

Approximation To further speed up our regressor, we approximate the learned linear filters with linear combinations of separable filters using the method proposed in [200]. Convolutions with separable filters are significantly faster than convolutions with non-separable ones, and the approximation is typically very good. Fig. 5.4(b) shows an example of such approximated filters.


(a) Original filters

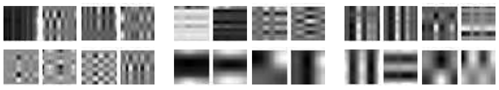

(b) Separable filters used for approximation

Figure 5.4: (a) The original 96 linear filters $\mathbf{w}_{nm}$ learned by our method on the *StLouis* sequence. Each row corresponds to a different image feature, respectively the horizontal image gradient, the vertical image gradient, the magnitude of the gradient, and the three color components in the LUV color space. (b) The 24 separable filters learned for each dimension independently using the method of [200]. Each original filter can be approximated as a linear combination of the separable filters, which can be convolved with the input images very efficiently.

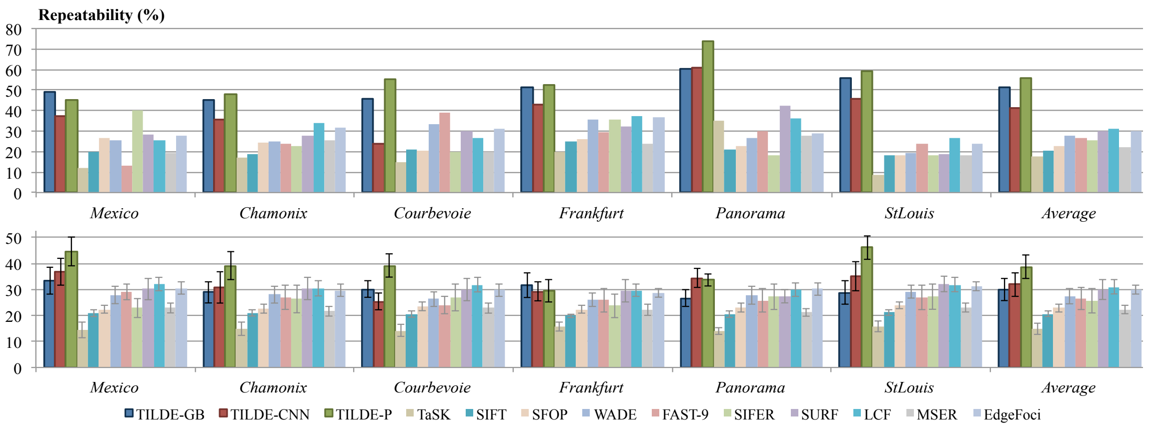


Figure 5.5: Repeatability (2%) score on the *Webcam* dataset. **Top:** average repeatability scores for each sequence trained on the respective sequences. **Bottom:** average repeatability score when trained on one sequence (the name of the training sequence is given below each graph) and tested on all other sequences. Although the gap reduces on the bottom graph, our method significantly outperforms the state-of-the-art in both cases, which shows that our method can generalise to unseen scenes.

5.5 Results

In this section we first describe our experimental setup and present both quantitative and qualitative results on our *Webcam* dataset and the standard *Oxford* and *EF* dataset.

5.5.1 Experimental Setup

We compare our approach to TaSK, SIFT, SFOP, WADE, FAST-9, SIFER, SURF, LCF, MSER [148], and Edge-Foci. In the following, our full method will be denoted TILDE-P. TILDE-P24 denotes the same method, after approximation of the piece-wise linear regressor using 24 separable filters.

To evaluate our regressor itself, we also compared it against two other regressors. The first regressor, denoted TILDE-GB, is based on boosted regression trees and is an adaptation of the one used in [199] for centerline detection to keypoint detection, with the same parameters used for implementation as in the original work. The second regressor we tried, denoted TILDE-CNN, is a Convolutional Neural Network, with an architecture similar to the LeNet-5 network [128] but with an additional convolution layer and a max-pooling layer. The first, third, and fifth layers are convolutional layers; the first layer has a resolution of 28×28 and filters of size 5×5, the third

Table 5.1: Repeatability performance of our best regressors. The best results are in bold. Our approach provides the highest repeatability, when using our piece-wise linear regressor. Note that on *Oxford* and *EF* datasets the performance are slightly better when using smaller number of separable filters to approximate the original ones, probably because the approximated filters tend to be smoother.

	Webcam	*Oxford*		*EF*	
#keypoints	*(2%)*	*Stand.*	*(2%)*	*Stand.*	*(2%)*
TILDE-GB	33.3	54.5	32.8	43.1	16.2
TILDE-CNN	36.8	51.8	49.3	43.2	27.6
TILDE-P24	40.7	**58.7**	**59.1**	**46.3**	**33.0**
TILDE-P	**48.3**	58.1	55.9	45.1	31.6
FAST-9	26.4	53.8	47.9	39.0	28.0
SFOP	22.9	51.3	39.3	42.2	21.2
SIFER	25.7	45.1	40.1	27.4	17.6
SIFT	20.7	46.5	43.6	32.2	23.0
SURF	29.9	56.9	57.6	43.6	28.7
TaSK	14.5	25.7	15.7	22.8	10.0
WADE	27.5	44.3	51.0	25.6	28.6
MSER	22.3	51.5	35.9	38.9	23.9
LCF	30.9	55.0	40.1	41.6	23.1
EdgeFoci	30.0	54.9	47.5	46.2	31.0

layer has 10 features maps of size 12×12 and filters of size 5×5, and the fifth layer 50 feature maps of size 4×4, and filters of size 3×3. The second, fourth, and sixth layers are max-pooling layers of size 2×2. The seventh layer is a layer of 500 neurons fully connected to the previous layer, which is followed by the eighth layer which is a fully-connected layer with a sigmoid activation function, followed by the final output layer. For the output layer we use the l_2 regression cost function.

5.5.2 Quantitative Results

We thoroughly evaluated the performance of our approach using the same repeatability measure as [181], on our *Webcam* dataset, and the *Oxford* and *EF* datasets. The repeatability is defined as the number of keypoints consistently detected across two aligned images. As in [181] we consider keypoints that are less than 5 pixels apart when projected to the same image as repeated. However, the repeatability measure has two caveats: First, a keypoint close to several projections can be counted several times. Moreover, with a large enough number of keypoints, even simple random sampling can achieve high repeatability as the density of the keypoints becomes high.

We therefore make this measure more representative of the performance with two modifications: First, we allow a keypoint to be associated only with its nearest neighbor, in other words, a keypoint cannot be used more than once when evaluating repeatability. Second, we restrict the number of keypoints to a small given number, so that picking the keypoints at random locations would results with a repeatability score of only 2%, reported as *Repeatability (2%)* in the experiments.

We also include results using the standard repeatability score, 1000 keypoints per image, and a fixed scale of 10 for our methods, which we refer to as *Oxford Stand.* and *EF Stand.*, for comparison with previous papers, such as [150, 260]. Table 5.1 shows a summary of the quantitative results.

Repeatability on our Webcam Dataset

Fig. 5.5 gives the repeatability scores for our *Webcam* dataset. Fig. 5.5-top shows the results of our method when trained on each sequence and tested on the same sequence, with the set of images divided into disjoint train, validation, and test sets. Fig. 5.5-bottom shows the results when we apply our detector trained on one sequence to all other unseen sequences from the *Webcam* dataset. We significantly outperform state-of-the-art methods when using a detector trained specifically to each sequence. Moreover, while the gap is reduced when we test on un-seen sequences, we still outperform all compared methods by a significant margin, showing the generalization capability of our method.

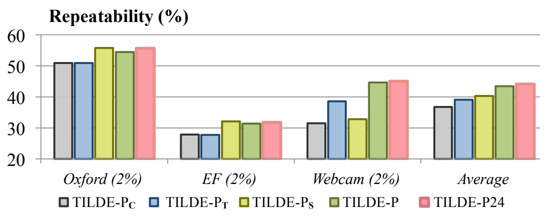


Figure 5.6: Effects of the three terms of the objective function, and of the approximation using separable filters.

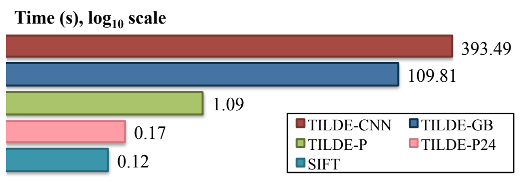


Figure 5.7: Time comparison for the full pipeline of our various regressors compared with the SIFT detector. Evaluations were run on the same machine on an 640×418 image.

Repeatability on Oxford and EF Datasets

In Fig. 5.8 we also evaluate our method on *Oxford* and *EF* datasets. *Oxford* dataset is simpler in the sense that it does not exhibit the drastic changes of the *Webcam* dataset but it is a reference for the evaluation of keypoint detectors. *EF* dataset on the other hand exhibits drastic illumination changes and is very challenging. It is therefore interesting to evaluate our approach on these datasets.

Instead of learning a new keypoint detector on this dataset, we apply the detector learned using the *Chamonix* sequence from the *Webcam* dataset. Our method still achieves state-of-the-art performance. We even significantly outperform state-of-the-art methods in the case of the *Bikes*, *Trees*, *Leuven* and *Rushmore* images, which are outdoor scenes. Note that we also obtain good results for *Boat* which has large scale changes, although we currently do not consider scale in learning and detecting. Repeatability score shown here is lower than what was reported in previous works [150, 181] as we consider a smaller number of keypoints. As mentioned before, considering a large number of keypoints artificially improves the repeatability score.

5.5.3 Qualitative Results

We also give in Fig. 5.9 some qualitative results on the task of matching challenging pairs of images captured at different days under different weather conditions. Our matching pipeline is as follow: we first extract keypoints in both images using the different methods we want to compare, compute the keypoints descriptors, and compute the homography between the two images using RANSAC. Since the goal of this comparison is to evaluate keypoints

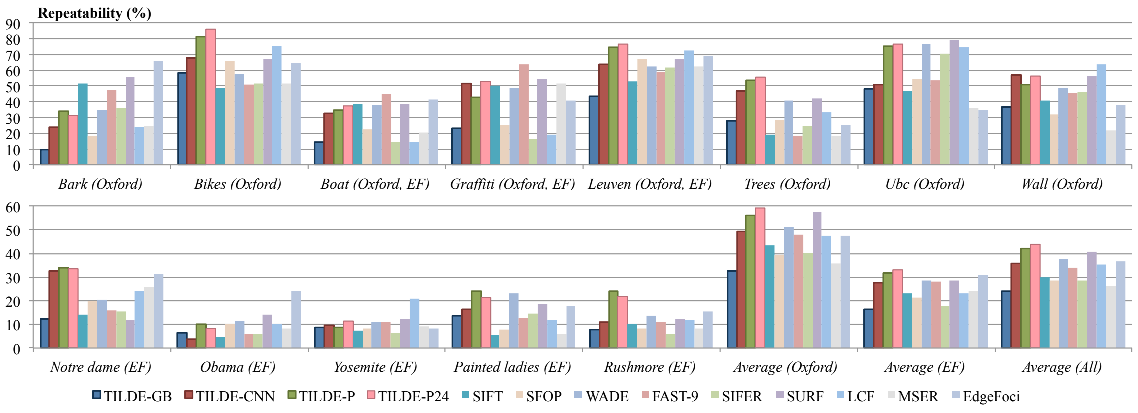


Figure 5.8: Repeatability (2%) score on the *Oxford* and *EF* datasets. Our methods are trained on the *Chamonix* sequence from the *Webcam* dataset and tested on *Oxford* and *EF* datasets.

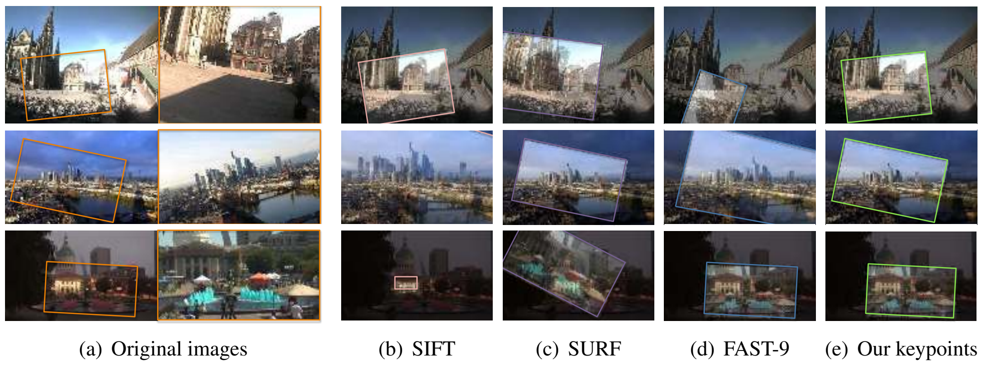


Figure 5.9: Qualitative results on several images from different sequences. From top to bottom: *Courbevoie*, *Frankfurt*, and *StLouis*. (a) Pairs of images to be matched, with ground truth transformation, transformations obtained with (b) the SIFT detector, (c) the SURF detector, (d) the FAST-9 detector, and (e) our TILDE detector.

not descriptors, we use the SIFT descriptor for all methods. Note that we also tried using other descriptors [17, 182, 35, 4, 134] but due to the drastic difference between the matched images, only SIFT descriptors with ground truth orientation and scale worked. We compare our method with the SIFT [140], SURF [17], and FAST-9 [181] detectors, using the same number of keypoints (300) for all methods. Our method allows to retrieve the correct transformations between the images even under such drastic changes of the scene appearance.

5.5.4 Effects of the Three Loss Terms

Fig. 5.6 gives the results of the evaluation of the influence of each loss term of Eq. (5.2) by evaluating the performance of our detector without each term. We will refer to our method when using only the classification loss as TILDE-P_C, when using both classification loss and the temporal regularization as TILDE-P_T, and when using the classification loss and the shape regularization as TILDE-P_S. We achieve the best performance when all three terms are used together. Note that the shape regularization enhances the repeatability on *Oxford* and *EF*, two completely unseen datasets, whereas the temporal regularization helps when we test on images which are similar to the training set.

5.5.5 Computation Times

Fig. 5.7 gives the computation time of SIFT and each variant of our method. TILDE-P24 is not very far from SIFT. Note that our method is highly parallelizable, while our current implementation does not benefit from any parallelization. We therefore believe that our method can be significantly sped up with a better implementation.

5.6 Conclusion

In this chapter, we have introduced a learning scheme to detect keypoints reliably under drastic changes of weather and lighting conditions. We proposed an effective method for generating the training set to learn regressors. We learned three regressors, which among them, the piece-wise linear regressor showed best result. We evaluated our regressors on our new outdoor keypoint benchmark dataset. Our regressors significant outperforms the current state-of-the-art on our new benchmark dataset and also achieve state-of-the-art performances on *Oxford* and *EF* datasets, demonstrating their generalisation capability.

Chapter 6

Gradient Response Maps for Real-Time Detection of Texture-Less Objects

As we discussed in the previous chapters, feature points are very powerful for some problems, such as matching of outdoor images, or the recognition of well-textured objects. However, this kind of approach will fail on texture-less objects such as those of Fig. 6.1, whose appearance is often dominated by their projected contours.

To overcome this problem, with Stefan Hinterstoisser, a PhD candidate at TU Munich, and Peter Sturm, who was doing a sabbatical there, and Cedric Cagniard, another PhD candidate at TU Munich, we propose several methods based on real-time multi-modality template recognition for rigid 3D object instances [96, 95, 98].

I started collaborating with Stefan after his visit at the CVLab in 2008. The motivation for this work came from my own visit at Willow Garage, a Robotics company that was also developing OpenCV, as they needed to detect textureless objects for many different problems. It was also the first time we worked with the Kinect, the RGB-D camera coming with the Microsoft X-Box. After he defended his PhD, Stefan went to work for Willow Garage, which was then bought by Google. Currently, Stefan still works at Google on 3D object detection.

Our approach here has its roots in much earlier work on template matching [24, 107, 74, 166]. Our templates can both be built and matched very quickly. We showed that this makes it very easy and virtually instantaneous to learn new object instances by simply adding new templates to the database while maintaining reliable real-time recognition. One of the contributions is to show how to combine color and depth information into what we called "multi-modal templates," or LineMOD, and that these two cues are complementary. In this work, our templates are still hand-crafted. However, its success motivated the development of a learning-based method to compute the templates, which will be described in the next chapter.

Instead of making the templates invariant to small deformations and translations as with DOT [96, 171], our first attempt of computing object descriptors, we build here a representation of the input images which has similar invariance properties. This allows us to consider all gradient orientations in local image neighborhoods instead of the dominant ones only. To further increase the robustness, our method simultaneously leverages the information of multiple acquisition modalities to define a template. In this work, we focus on the combination of 2D gradients and 3D normals generated from a color image and a dense depth map. However, our approach is very generic and could easily integrate other modalities as long as they provide measurements aligned with the image that can be quantized. By introducing a novel similarity measure taking them into account, we can now avoid the problems due to too strong gradients in the background as illustrated by Fig. 6.1. For visual image integration, we show how to extract gradients from the color images which are more robust to the background than gradients computed on gray value images. For depth integration, we propose a method that robustly computes 3D surface normals from dense depth maps in real-time, making sure to preserve depth discontinuities on occluding contours and to smooth out discretization noise of the sensor.

To avoid slowing down detection when using this finer method, we have to make careful considerations about how modern CPUs work. A naive implementation would result in many "memory cache misses", which slow down the computations, and we thus show how to structure our image representation in memory to prevent them and to additionally exploit heavy SSE parallelization. We consider this as an important contribution: Because of the nature of the hardware improvements, it is not guaranteed anymore that legacy code will run faster on the new versions of CPUs [22]. This is particularly true for Computer Vision, which algorithms are often computationally expensive. It is now required to take the CPU architecture into account, which is not an easy task.

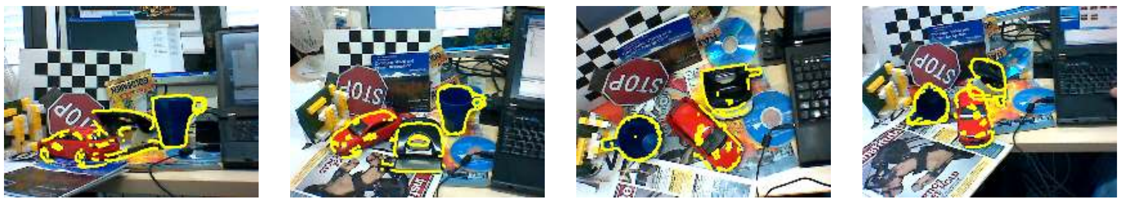


Figure 6.1: Our method can detect texture-less 3D objects in real-time under different poses over heavily cluttered background using gradient orientation.

In the remainder of this chapter, we first discuss related work before we explain our approach. We then discuss the theoretical complexity of our approach. We finally present experiments and quantitative evaluations for challenging scenes.

6.1 Related Work

Template Matching has played an important role in tracking-by-detection applications for many years. This is due to its simplicity and its capability to handle different types of objects. It neither needs a large training set nor a time-consuming training stage, and can handle low-textured or texture-less objects, which are, for example, difficult to detect with feature points-based methods [140, 97]. Unfortunately, this increased robustness often comes at the cost of an increased computational load that makes naïve template matching inappropriate for real-time applications. So far, several works have attempted to reduce this complexity.

An early approach to Template Matching [166] and its extension [74] include the use of the Chamfer distance between the template and the input image contours as a dissimilarity measure. For instance, Gavrila and Philomin [74] introduced a coarse-to-fine approach in shape and parameter space using Chamfer Matching [24] on the Distance Transform of a binary edge image. The Chamfer Matching minimizes a generalized distance between two sets of edge points. Although being fast when using the Distance Transform (DT), the disadvantage of the Chamfer Transform is its sensitivity to outliers which often result from occlusions.

Another common measure on binary edge images is the Hausdorff distance [183]. It measures the maximum of all distances from each edge point in the image to its nearest neighbor in the template. However, it is sensitive to occlusions and clutter. Huttenlocher *et al.* [107] tried to avoid that shortcoming by introducing a generalized Hausdorff distance which only computes the maximum of the k-th largest distances between the image and the model edges and the l-th largest distances between the model and the image edges. This makes the method robust against a certain percentage of occlusions and clutter. Unfortunately, a prior estimate of the background clutter in the image is required but not always available. Additionally, computing the Hausdorff distance is computationally expensive and prevents its real-time application when many templates are used.

Both Chamfer Matching and the Hausdorff distance can easily be modified to take the orientation of edge points into account. This drastically reduces the number of false positives as shown in [166], but unfortunately also increases the computational load.

All these methods use binary edge images obtained with a contour extraction algorithm, using the Canny method [36] for example, and they are very sensitive to illumination changes, noise and blur. For instance, if the image contrast is lowered, the number of extracted edge pixels progressively decreases which has the same effect as increasing the amount of occlusion.

The method proposed in [205] tries to overcome these limitations by considering the image gradients in contrast to the image contours. It relies on the dot product as a similarity measure between the template gradients and those in the image. Unfortunately, this measure rapidly declines with the distance to the object location, or when the object appearance is even slightly distorted. As a result, the similarity measure must be evaluated densely, and with many templates to handle appearance variations, making the method computationally costly. Using image pyramids provides some speed improvements, however, fine but important structures tend to be lost if one does not carefully sample the scale space.

Contrary to above mentioned methods, there are also approaches addressing the general visual recognition problem: they are based on learning and aim at detecting object categories rather than *a priori* known object instances as seen in challenges such as PASCAL [60]. While they are more powerful in terms of class generalization, they are usually much slower during learning and runtime which makes them unsuitable for online applications.

Amit [9], for instance, proposed a coarse to fine approach, however, in contrast to [205], by making use of spreading gradient orientations in local neighborhoods. The amount of spreading is learned for each object part in an initial stage. While this approach — used for license plate reading — achieves high recognition rates, it is not real-time capable.

Histogram of Gradients (HoG) [46] is another related and very popular method. It statistically describes the distribution of intensity gradients in localized portions of the image. The approach is computed on a dense grid with uniform intervals and uses overlapping local histogram normalization for better performance. It has proven to give reliable results but tends to be slow due to the computational complexity.

Ferrari *et al.* [65] provided a learning based method that recognizes objects via a Hough-style voting scheme with a non-rigid shape matcher on object boundaries of a binary edge image. The approach applies statistical methods to learn the model from few images that are only constrained within a bounding box around the object. While giving very good classification results, the approach is neither appropriate for object tracking in real-time due to its expensive computation nor is it precise enough to return the accurate pose of the object. Additionally, it is sensitive to the results of the binary edge detector, an issue that we discussed before.

Kalal *et al.* [117] developed the most recent learning based approach that put more focus on online learning. They showed how a classifier can be trained online in real-time, with a training set generated automatically. However, as we will see in the experiments, this approach is only suitable for smooth background transitions and not appropriate to detect known objects over unknown backgrounds.

Opposite to the above mentioned learning based methods, there are also approaches that are specifically trained on different viewpoints. As with our template-based approach, they can detect objects under different poses but typically require a large amount of training data and a long offline training phase. For example, in [235, 105, 170], one or several classifiers are trained to detect faces or cars under various views.

More recent approaches for 3D object detection are related to object class recognition. Stark *et al.* [204] rely on 3D CAD models and generate a training set by rendering them from different viewpoints. Liebelt and Schmid [135] combine a geometric shape and pose prior with natural images. Su *et al.* [212] use a dense, multivew representation of the viewing sphere combined with a part-based probabilistic representation. While these approaches are able to generalize to the object class they are not real-time capable and require expensive training.

From the related works which take into account multi-modal data information there are mainly approaches related to pedestrian detection [58, 59, 75, 253]. They use three kinds of clues: image intensity, depth and motion (optical flow). The most recent approach of Enzweiler *et. al* [58] builds part based models of pedestrians in order to handle occlusions caused by other objects and not specifically self occlusions which are modeled in other approaches [59, 253]. Besides pedestrian detection, there has been an approach to object classification, pose estimation and reconstruction introduced by [213]. Similar to us, the training data set is composed of depth and image intensities while the object classes are detected using the modified Hough transform. While being quite effective in real applications these approaches still require exhaustive training using large training data sets. This is usually prohibited in robotic applications where the robot has to explore an unknown environment and learn new objects online.

As mentioned in the introduction, we recently proposed a method to detect texture-less 3D object instances from different viewpoints based on templates [96]. Each object is represented as a set of templates, relying on local dominant gradient orientations to build a representation of the input images and the templates. Extracting the dominant orientations is useful to tolerate small translations and deformations. It is fast to perform and most of the time discriminant enough to avoid generating too many false positive detections.

However, we noticed that this approach degrades significantly when the gradient orientations are disturbed by stronger gradients of different orientations coming from background clutter in the input images. In practice, this often happens in the neighborhood of the silhouette of an object, which is unfortunate as the silhouette is a very important cue especially for texture-less objects. The method we propose in this chapter does not suffer from this problem while running at the same speed. Additionally, we show how to extend our approach to handle multiple modalities at the same time. As we will see this increases the robustness significantly.

6.2 Proposed Approach

In this section, we describe our template representation and show how a new representation of the input image can be built and used to parse the image to quickly find objects. We will start by deriving our similarity measure, emphasizing the contribution of each aspect of it. We also show how we implement our approach to efficiently use modern processor architectures. Additionally, we demonstrate how to integrate different modalities to increase robustness.

6.2.1 Similarity Measure

Our unoptimized similarity measure can be seen as the measure defined by Steger in [205] modified to be robust to small translations and deformations. Steger suggests to use:

$$\mathcal{E}_{\text{Steger}}(\mathcal{I}, \mathcal{T}, c) = \sum_{r \in \mathcal{P}} |\cos(\text{ori}(\mathcal{O}, r) - \text{ori}(\mathcal{I}, c + r))| \ , \tag{6.1}$$

where $\text{ori}(\mathcal{O}, r)$ is the gradient orientation in radians at location r in a reference image $\mathcal{O}$ of an object to detect. Similarly, $\text{ori}(\mathcal{I}, c + r)$ is the gradient orientation at c shifted by r in the input image $\mathcal{I}$. We use a list, denoted by $\mathcal{P}$, to define the locations r to be considered in $\mathcal{O}$. This way we can deal with arbitrarily shaped objects efficiently. A template $\mathcal{T}$ is therefore defined as a pair $\mathcal{T} = (\mathcal{O}, \mathcal{P})$.

Considering only the gradient orientations and not their norms makes the measure robust to contrast changes, and taking the absolute value of the cosine allows it to correctly handle object occluding boundaries: It will not be affected if the object is over a dark background, or a bright background.

The similarity measure of Eq. (6.1) is very robust to background clutter, but not to small shifts and deformations. A common solution is to first quantize the orientations and to use local histograms like in SIFT [140] or HoG [46]. However this can be unstable when strong gradients appear in the background. In DOT [96], we kept the dominant orientations of a region. This was faster than building histograms but suffers from the same instability. Another option is to apply Gaussian convolution to the orientations like in DAISY [221], but this would be too slow for our purpose.

We therefore propose a more efficient solution that easily incorporates many different modalities. Given a set of aligned reference images $\{\mathcal{O}_m\}_{m \in \mathcal{M}}$ of the object from a set $\mathcal{M}$ of modalities we redefine a template as $\mathcal{T} = (\{\mathcal{O}_m\}_{m \in \mathcal{M}}, \mathcal{P})$. $\mathcal{P}$ is a list of pairs (r, m) made of the locations r of a discriminant feature in modality m. Each template is created by extracting for each modality a small set of its most discriminant features from the corresponding reference image and by storing their locations. In the feature selection process, we take the location of the features into account to avoid an accumulation of features in one local area of the object while the rest of the object is not sufficiently described. As shown in Fig. 6.2, the modalities we use in our experiments come from a standard camera and a depth sensor aligned with the camera.

Our similarity measure is robust to small translations and deformations and incorporates different modalities. It can be formalized as:

$$\mathcal{E}(\{\mathcal{I}_m\}_{m \in \mathcal{M}}, \mathcal{T}, c) = \sum_{(r,m) \in \mathcal{P}} \left(\max_{t \in \mathcal{R}(c+r)} f_m(\mathcal{O}_m(r), \mathcal{I}_m(t)) \right) \ , \tag{6.2}$$

where $\mathcal{R}(c+r) = \left[c + r - \frac{T}{2}, c + r + \frac{T}{2}\right] \times \left[c + r - \frac{T}{2}, c + r + \frac{T}{2}\right]$ defines the neighborhood of size T centered on location $c + r$ in the input image $\mathcal{I}_m$ and the function $f_m(\mathcal{O}_m(r), \mathcal{I}_m(t))$ computes the similarity score for modality m between the reference image at location r and the input image at location t. Thus, for each feature we align the local neighborhood exactly to the associated location whereas in DOT [96], HoG [46] or SIFT [140], the features are adjusted only to some regular grid. As a result, we tremendously gain robustness when using the silhouette of the object. We show below how to compute this measure efficiently.

6.2.2 Spreading the Features

In order to avoid evaluating the $\max$ operator in Eq. (6.2) every time a new template must be evaluated against an image location, we first introduce a new binary representation — denoted by $\mathcal{J}_m$ — of the features of modality

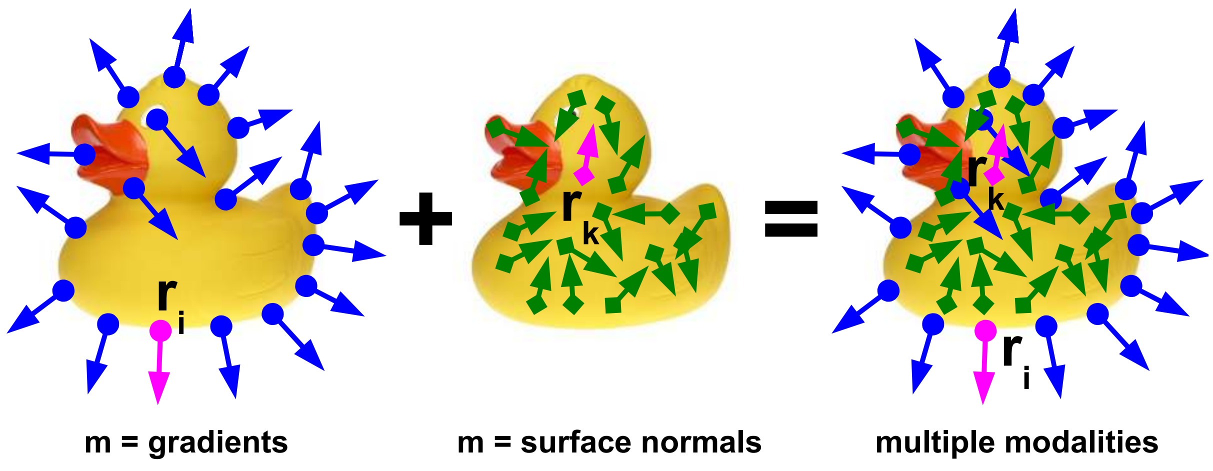


Figure 6.2: A toy duck with different modalities. **Left:** Strong and discriminative image gradients are mainly found on the contour. The gradient location r_i is displayed in pink. **Middle:** Reliable surface normals are mainly found on the body of the duck. The normal location r_k is displayed in pink. **Right:** LINE can combine multiple cues which are complementary: gradients are usually found on the object contour while surface normals are found on the object interior

m around each image location. We will then use this representation together with lookup tables to efficiently precompute these maximal values.

The computation of $\mathcal{J}_m$ is depicted in Fig. 6.3. We first quantize the input data for each modality into a small number of n_o values as done in previous approaches [140, 46, 96]. This allows us to "spread" the data of the input image $\mathcal{I}_m$ around their locations to obtain a new representation of the original image.

For efficiency, we encode the possible combinations of quantized input data spread to a given image location l using a binary string: Each individual bit of this string corresponds to one quantized value, and is set to 1 if this value is present in the neighborhood of l. The strings for all the image locations form the image $\mathcal{J}_m$ on the right part of Fig. 6.3. These strings will be used as indices of lookup tables for fast precomputation of the similarity measure, as it is described in the next subsection.

$\mathcal{J}_m$ can be computed very efficiently: We first compute a map for each quantized feature value, whose values are set to 1 if the corresponding pixel location in the input image has this feature value, and 0 if it does not. $\mathcal{J}_m$ is then obtained by shifting these maps over the range of $\left[-\frac{T}{2}, +\frac{T}{2}\right] \times \left[-\frac{T}{2}, +\frac{T}{2}\right]$ and merging all shifted versions with an OR operation.

6.2.3 Precomputing Response Maps

As shown in Fig. 6.4, $\mathcal{J}_m$ is used together with modality dependent lookup tables to precompute the value of the $\max$ operation in Eq. (6.2) for each location and each possible quantized value i in the template. We store the results into 2D maps $\mathcal{S}_{i,m}$ where m is the specific modality. Then, to evaluate the similarity function, we will just have to sum values read from these $\mathcal{S}_{i,m}$s.

We use a lookup table $\tau_{i,m}$ for each modality and for each of the n_o quantized orientations, computed offline as:

$$\tau_{i,m}[\mathcal{L}_m] = \max_{l \in \mathcal{L}_m} f_m(i, l) \, , \tag{6.3}$$

where

- i is the index of the quantized value of modality m. To keep the notations simple, we also use i to represent the corresponding value;
- $\mathcal{L}_m$ is a list of values of a special modality m appearing in a local neighborhood of a value i as described in Section 6.2.2. In practice, we use the integer value corresponding to the binary representation of $\mathcal{L}_m$ as an index to the element in the lookup table.

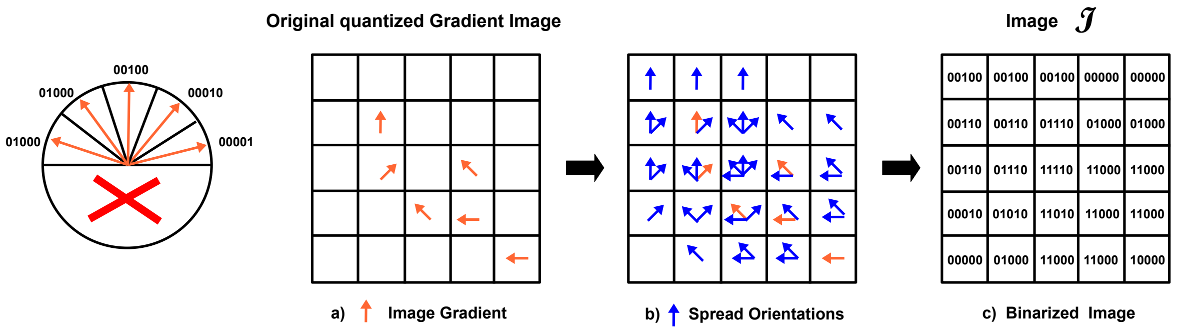


Figure 6.3: Spreading demonstrated on the example of gradient orientations. For simplicity we omit modality m. **Left:** The gradient orientations and their binary code. We do not consider the direction of the gradients. **(a)** The gradient orientations in the input image, shown in orange, are first extracted and quantized. **(b)** Then, the locations around each orientation are also labeled with this orientation, as shown by the blue arrows. This allows our similarity measure to be robust to small translations and deformations. **(c)** $\mathcal{J}$ is an efficient representation of the orientations after this operation, and can be computed very quickly. For this figure, $T = 3$ and $n_o = 5$. In practice, we use $T = 8$ and $n_o = 8$.

For quantized value i we can now compute the value at each location c of the response map $\mathcal{S}_{i,m}$ as:

$$\mathcal{S}_{i,m}(c) = \tau_{i,m}[\mathcal{J}_m(c)] \; . \tag{6.4}$$

Finally, the similarity measure of Eq. (6.2) can be evaluated as:

$$\mathcal{E}(\{\mathcal{I}_m\}_{m\in\mathcal{M}}, \mathcal{T}, c) = \sum_{(r,m)\in\mathcal{P}} \mathcal{S}_{\mathcal{O}_m(r),m}(c+r) \; . \tag{6.5}$$

Since the maps $\mathcal{S}_{i,m}$ are shared between the templates, matching several templates against the input image can be done very fast once they are computed.

6.2.4 Linearizing the Memory for Parallelization

Thanks to Eq. (6.5), we can match a template against the whole input image by only adding the values in the response maps $\mathcal{S}_{i,m}$. However, one of the advantages of spreading the quantized values as was done in Section 6.2.2 is that it is sufficient to do the evaluation only every T^{th} pixel without reducing the recognition performance. If we want to exploit this property efficiently, we have to take into account the architecture of modern computers.

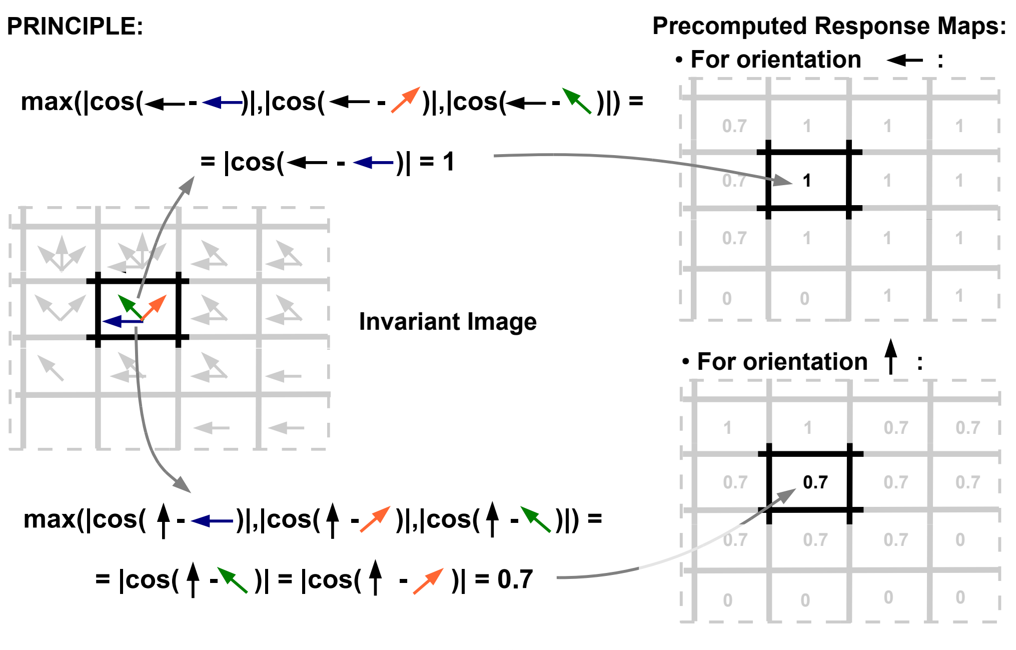
Modern processors do not only read one data value at a time from the main memory but several ones simultaneously, called a *cache line*. Accessing the memory at random places results in a *cache miss* and slows down the computations. On the other hand, accessing several values from the same cache line is very cheap. As a consequence, storing data in the same order as they are read speeds up the computations significantly. In addition, this allows parallelization: For instance, if 8-bit values are used as it is the case for our $\mathcal{S}_{i,m}$ maps, SSE instructions can perform operations on 16 values in parallel. On the GPU, even more operations can be performed simultaneously (e.g. 1024 operations on the NVIDIA Quadro GTX 590).

Therefore, as shown in Fig. 6.5, we store the precomputed response maps $\mathcal{S}_{i,m}$ into memory in a cache-friendly way: We restructure each response map so that the values of one row that are T pixels apart on the x axis are now stored next to each other in memory. We continue with the row which is T pixels apart on the y axis once we finished with the current one.

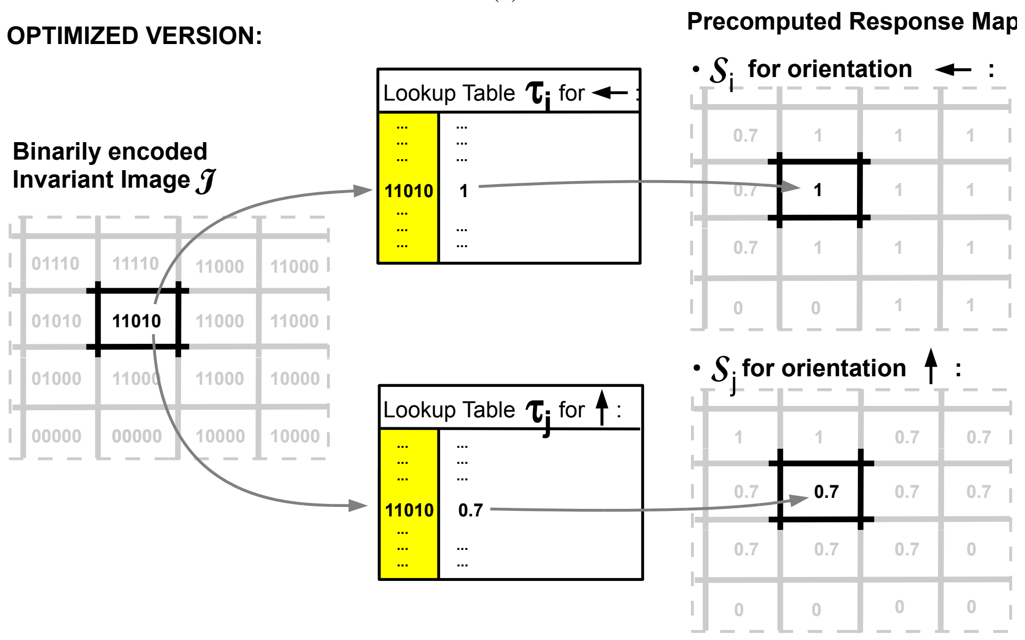
Finally, as described in Fig. 6.6, computing the similarity measure for a given template at each sampled image location can be done by adding the linearized memories with an appropriate offset computed from the locations r in the templates.

6.2.5 Modality Extraction

We now turn to how we handle the different modalities and demonstrate this on image and depth data.


PRINCIPLE:
Precomputed Response Maps:
• For orientation :
max(|cos(-)|,|cos(-)|,|cos(-)|) =
= |cos(-)| = 1
Invariant Image
• For orientation :
max(|cos(-)|,|cos(-)|,|cos(-)|) =
= |cos(-)| = |cos(-)| = 0.7

(a)


OPTIMIZED VERSION:
Precomputed Response Map
• $\mathcal{S}_i$ for orientation :
Lookup Table τ_i for :
Binarily encoded Invariant Image $\mathcal{J}$
11010
• $\mathcal{S}_j$ for orientation :
Lookup Table τ_j for :

(b)

Figure 6.4: Precomputing the Response Maps $\mathcal{S}_i$ on the example of gradient orientation. For simplicity we omit modality m. **(a):** There is one response map for each quantized orientation. They store the maximal similarity between their corresponding orientation and the orientations ori_j already stored in the "Invariant Image". **(b):** This can be done very efficiently by using the binary representation of the list of orientations in $\mathcal{J}$ as an index to lookup tables of the maximal similarities.

Image Cue

We chose to consider image gradients because they proved to be more discriminant than other forms of representations [140, 205] and are robust to illumination change and noise. Additionally, image gradients are often the only reliable image cue when it comes to texture-less objects. Considering only the orientation of the gradients and not their norms makes the measure robust to contrast changes, and taking the absolute value of cosine between them allows it to correctly handle object occluding boundaries: It will not be affected if the object is over a dark

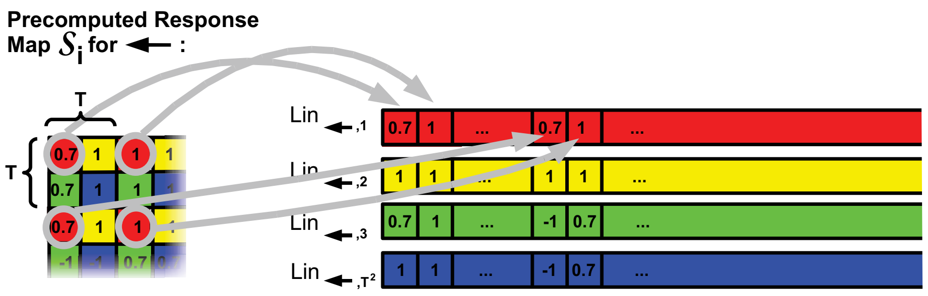


Figure 6.5: Restructuring the way the response images $\mathcal{S}_{i,m}$ are stored in memory. For simplicity we omit modality m within the figure. The values of one image row that are T pixels apart on the x axis are stored next to each other in memory. Since we have T^2 such linear memories per response map, and n_o quantized orientations, we end up with $T^2 \cdot n_o$ different linear memories.

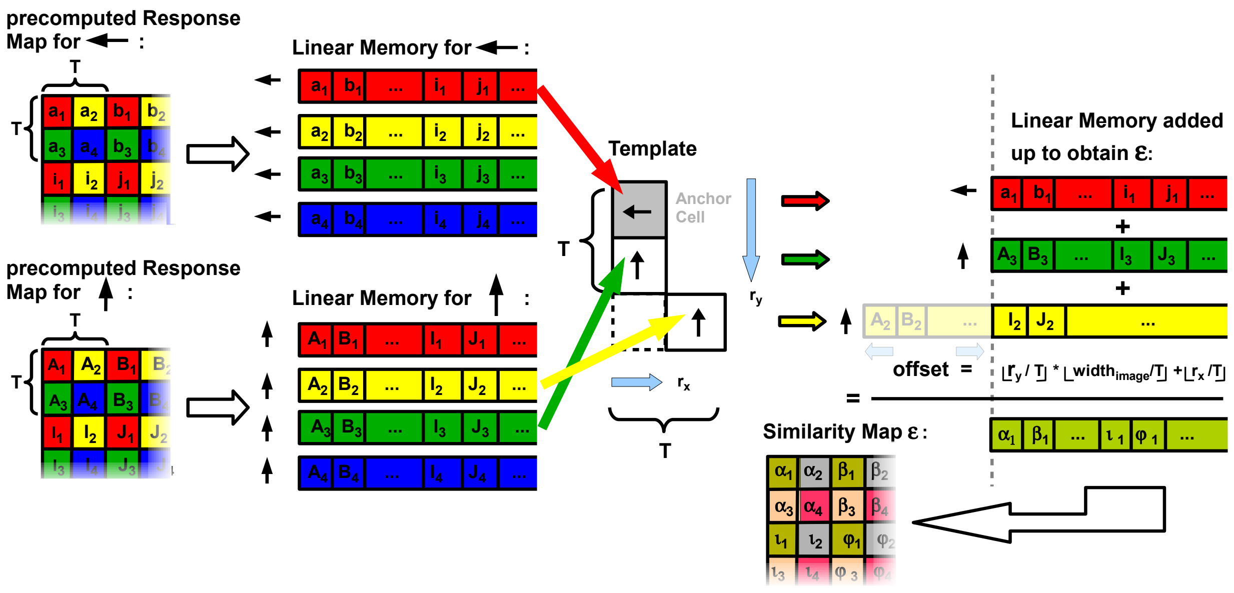

precomputed Response Map for ← :
Linear Memory for ← :
precomputed Response Map for ↑ :
Linear Memory for ↑ :
Template
Anchor Cell
T
r_y
r_x
Linear Memory added up to obtain ε:
offset = ⌊r_y / T⌋ * ⌊width_image/T⌋ + ⌊r_x /T⌋
Similarity Map ε :

Figure 6.6: Using the linear memories on the example of gradient orientation. We can compute the similarity measure over the input image for a given template by adding up the linear memories for the different orientations of the template (the orientations are visualized by black arrows pointing in different directions), shifted by an offset depending on the locations $(r_x, r_y)^\top$ in the template. Performing these additions with parallel SSE instructions further speeds up the computation. In the final similarity map $\mathcal{E}$ only each Tth pixel has to be parsed to find the object.

background, or a bright background.

To increase robustness, we compute the orientation of the gradients on each color channel of our input image separately and for each image location use the gradient orientation of the channel whose magnitude is largest. Given an RGB color image $\mathcal{I}$, we compute the gradient orientation map $\mathcal{I}_\mathcal{G}(x)$ at location x with

$$\mathcal{I}_\mathcal{G}(x) = \text{ori}(\hat{\mathcal{C}}(x)) \tag{6.6}$$

where

$$\hat{\mathcal{C}}(x) = \underset{\mathcal{C}\in\{R,G,B\}}{\arg\max} \left\| \frac{\partial \mathcal{C}}{\partial x} \right\| \tag{6.7}$$

and R, G, B are the RGB channels of the corresponding color image, as it was done in [64]. Our similarity measure is then:

$$f_\mathcal{G}(\mathcal{O}_\mathcal{G}(r), \mathcal{I}_\mathcal{G}(t)) = |cos(\mathcal{O}_\mathcal{G}(r) - \mathcal{I}_\mathcal{G}(t))| \tag{6.8}$$

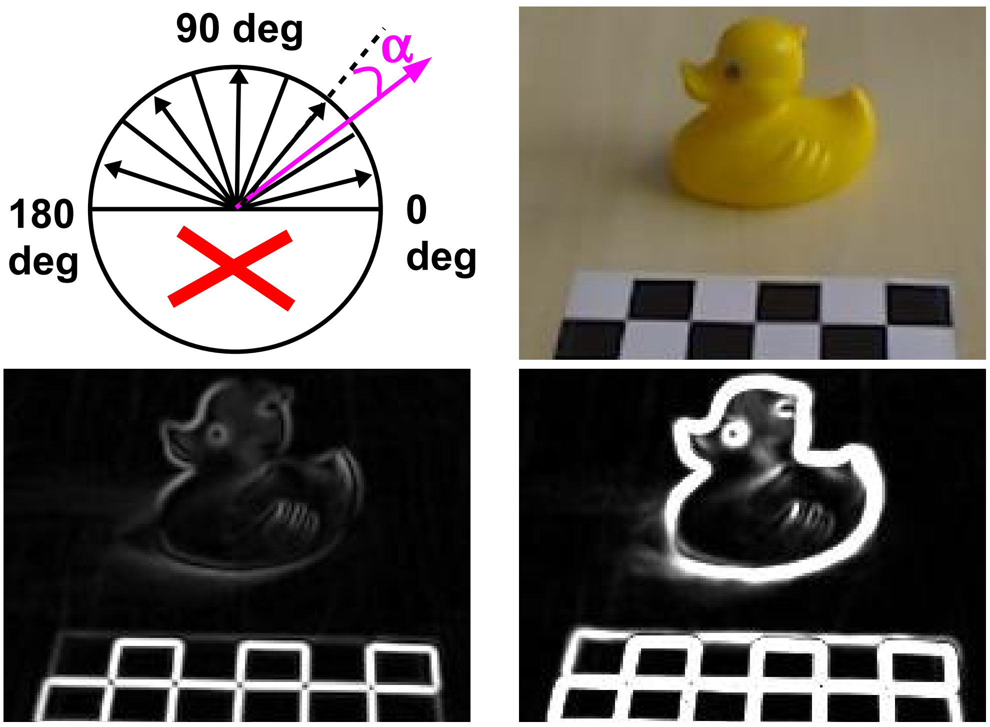


Figure 6.7: **Upper Left:** Quantizing the gradient orientations: the pink orientation is closest to the second bin. **Upper right:** A toy duck with a calibration pattern **Lower Left:** The gradient image computed on a gray value image. The object contour is hardly visible. **Lower right:** Gradients computed with our method. Details of the object contours are clearly visible.

where $\mathcal{O}_\mathcal{G}(r)$ is the gradient orientation map of the reference image at location r and $\mathcal{I}_\mathcal{G}(t)$ the gradient orientation map of the current image at location t respectively.

In order to quantize the gradient orientation map we omit the gradient direction, consider only the gradient orientation and divide the orientation space into n_0 equal spacings as shown in Fig. 6.7. To make the quantization robust to noise, we assign to each location the gradient whose quantized orientation occurs most often in a 3×3 neighborhood. We also keep only the gradients whose norms are larger than a small threshold. The whole unoptimized process takes about 31ms on the CPU for a VGA image.

Depth Cue

Similar to the image cue, we decided to use quantized surface normals computed on a dense depth field for our template representation as shown in Fig. 6.8. They allow us to represent both close and far objects while fine structures are preserved.

In the following, we propose a method for the fast and robust estimation of surface normals in a dense range image. Around each pixel location x, we consider the first order Taylor expansion of the depth function $\mathcal{D}(x)$:

$$\mathcal{D}(x + dx) - \mathcal{D}(x) = dx^\top \nabla\mathcal{D} + h.o.t. \tag{6.9}$$

Within a patch defined around x, each pixel offset dx yields an equation that constrains the value of $\nabla\mathcal{D}$, allowing to estimate an optimal gradient $\hat{\nabla\mathcal{D}}$ in a least-square sense. This depth gradient corresponds to a 3D plane going through three points X, X_1 and X_2:

$$X = \vec{v}(x)\mathcal{D}(x), \tag{6.10}$$

$$X_1 = \vec{v}(x + [1, 0]^\top)(\mathcal{D}(x) + [1, 0]\hat{\nabla\mathcal{D}}), \tag{6.11}$$

$$X_2 = \vec{v}(x + [0, 1]^\top)(\mathcal{D}(x) + [0, 1]\hat{\nabla\mathcal{D}}). \tag{6.12}$$

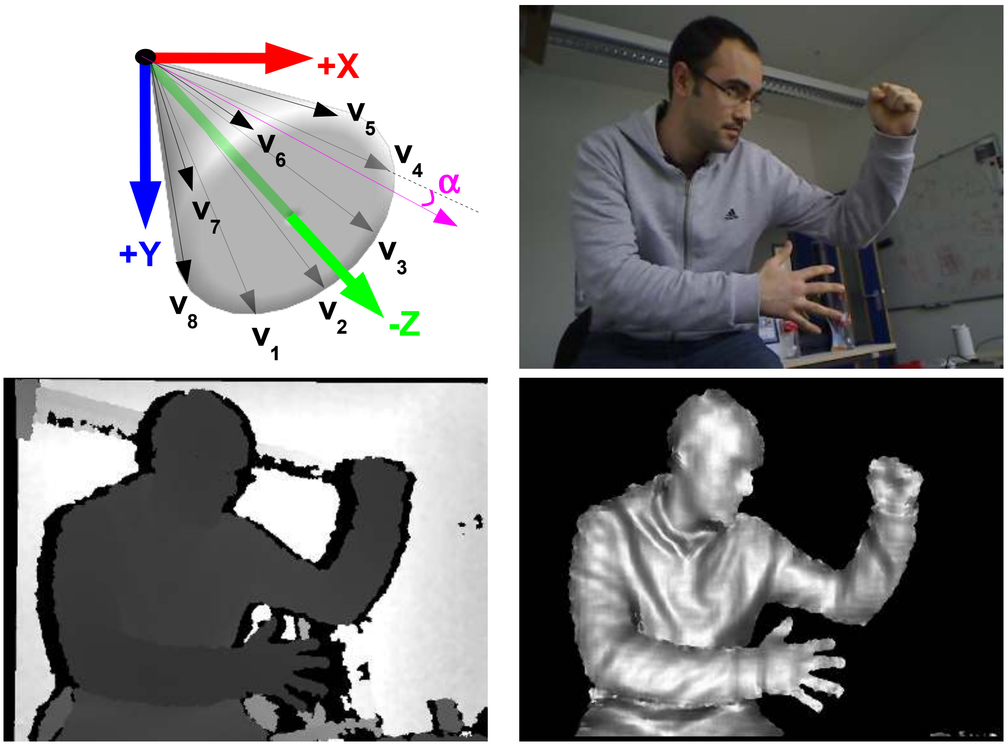


Figure 6.8: **Upper Left:** Quantizing the surface normals: the pink surface normal is closest to the precomputed surface normal v_4. It is therefore put into the same bin as v_4. **Upper right:** A person standing in an office room. **Lower Left:** The corresponding depth image. **Lower right:** Surface normals computed with our approach. Details are clearly visible and depth discontinuities are well handled. We removed the background for visibility reasons.

where $\vec{v}(x)$ is the vector along the line of sight that goes through pixel x and is computed from the internal parameters of the depth sensor. The normal to the surface at the 3D point that projects on x can be estimated as the normalized cross-product of $X_1 - X$ and $X_2 - X$.

However this would not be robust around occluding contours, where the first order approximation of Eq. (6.9) no longer holds. Inspired by bilateral filtering, we ignore the contributions of pixels whose depth difference with the central pixel is above a threshold. In practice, this approach effectively smooths out quantization noise on the surface, while still providing meaningful surface normal estimates around strong depth discontinuities. Our similarity measure is then defined as the dot product of the normalized surface normals:

$$f_{\mathcal{D}}(\mathcal{O}_{\mathcal{D}}(r), \mathcal{I}_{\mathcal{D}}(t)) = \mathcal{O}_{\mathcal{D}}(r)^{\top} \mathcal{I}_{\mathcal{D}}(t) \tag{6.13}$$

where $\mathcal{O}_{\mathcal{D}}(r)$ is the normalized surface normal map of the reference image at location r and $\mathcal{I}_{\mathcal{D}}(t)$ the normalized surface normal map of the current image at location t.

Finally, as shown in Fig. 6.8, we measure the angles between the computed normal and a set of precomputed vectors to quantize the normal directions into n_0 bins. These vectors are arranged in a right circular cone shape originating from the peak of the cone pointing towards the camera. To make the quantization robust to noise, we assign to each location the quantized value that occurs most often in a 5×5 neighborhood. The whole process is very efficient and needs only 14ms on the CPU and less than 1ms on the GPU.

6.2.6 Computation Time Study

In this section we compare the numbers of operations required by the original method from [205] and the LINE method we propose. In order to do a fair comparison, we only assume one modality used in LINE.

The time required by $\mathcal{E}_{\text{Steger}}$ from [205] to evaluate R templates over an $M \times N$ image is $M \cdot N \cdot R \cdot G \cdot (S + A)$, where G the average number of gradients in a template, S the time to evaluate the similarity function between two gradient orientations and, A the time to add two values.

Changing $\mathcal{E}_{\text{Steger}}$ to Eq. (6.2) and making use of $\mathcal{J}_{\mathcal{G}}$ leads to a computation time of $M \cdot N \cdot T^2 \cdot O + \frac{M \cdot N}{T^2} \cdot R \cdot G \cdot (L + A)$, where L is the time needed for accessing once the lookup tables τ_i and O is the time to OR two values together. The first term corresponds to the time needed to compute $\mathcal{J}_{\mathcal{G}}$, the second one to the time needed to actually compute Eq. (6.2).

Precomputing the response maps $\mathcal{S}_{i,\mathcal{G}}$ further changes the complexity of LINE to $M \cdot N \cdot (T^2 \cdot O + n_o \cdot L) + \frac{M \cdot N}{T^2} \cdot R \cdot G \cdot A$.

Linearizing our memory allows the additional use of parallel SSE instructions. In order to run 16 operations in parallel, we approximate the response values in the lookup tables using bytes. The final complexity of our algorithm is then $M \cdot N \cdot (T^2 \cdot O + (n_o + 1) \cdot L) + \frac{M \cdot N}{16T^2} \cdot R \cdot G \cdot A$.

In practice we use $T = 8$, $M = 480$, $N = 640$, $R > 1000$, $G \approx 100$ and $n_o = 8$. If we assume for simplicity that $L \approx A \approx O \approx 1$ time unit, this leads to a speed improvement compared to the original energy formulation $\mathcal{E}_{\text{Steger}}$ of a factor $T^2 \cdot 16(1 + S)$ if we assume that the number of templates R is large. Note that we did not incorporate the cache friendliness of LINE since it is very hard to model. Still, since [205] evaluates the similarity of two orientations with the normalized dot product of the two corresponding gradients, S can be set to 3 and we obtain a theoretical gain in speed of at least a factor of 4096.

6.2.7 Experimental Validation

We compared our approaches, which we call LINE-MOD (for "multimodal-LINE"), LINE-2D, which uses only the image intensities and a variant that we call LINE-3D, that uses only the depth map, to DOT [96], HOG [46], TLD [117] and Steger [205]. DOT is a representative method for fast template matching while HOG and Steger stand for slow but very robust template matching. In contrast to them, TLD represents the newest insights in online learning.

Instead of gray value gradients, we used the color gradient method of Sec. 6.2.5 for DOT, HOG and Steger, which resulted in an enhancment of recognition. Moreover, we used the author's implementations for DOT and TLD. For the approach of Steger we used our own implementation with 4 pyramid levels. For HOG, we also used our own optimized implementation and replaced the Support Vector Machine mentioned in the original work of HOG by a nearest neighbor search. In this way, we can use it as a robust representation and quickly learn new templates as with the other methods.

The experiments were performed on one processor of a standard notebook with an Intel Centrino Processor Core2Duo with 2.4 GHz and 3 GB of RAM. For obtaining the image and the depth data we used the Primesense$^{(tm)}$ PSDK 5.0 device.

6.2.8 Robustness

We used six sequences made of over 2000 real images each. Each sequence presents illumination and large viewpoint changes over heavy cluttered background. Ground truth is obtained with a calibration pattern attached to each scene that enables us to know the actual location of the object. The templates were learned over homogeneous background.

We consider the object to be correctly detected if the location given back is within a fixed radius of the ground truth position. As depicted in the left columns of Fig. 6.11 and Fig. 6.12, LINE-MOD always outperforms all the other approaches and shows only few false positives. We believe that this is due to the complementarity of the object features that compensate for the weaknesses of each other. This is especially interesting, as the depth cue alone often performs not very well. Apart from LINE-MOD, one can see that LINE-2D outperforms all other methods most of the time. The reason for the weak detection results of TLD is that this method works well unders smooth background transition. However, as we see in our experiments, it is not suitable to detect known objects over unknown backgrounds.

The superiority of LINE-MOD becomes more obvious in Table 6.1: If we set the threshold for each approach to allow for 97% true positive rate and only evaluate the hypothesis with the largest response, we obtain for LINE-MOD a high detection rate with a very small false positive rate. This is in contrast to LINE-2D, where the true positive rate is often over 90%, but the false positive rate is not negligible, which makes expensive post-processing necessary. In LINE-MOD, using only the response with the largest value might be sufficient in most cases.

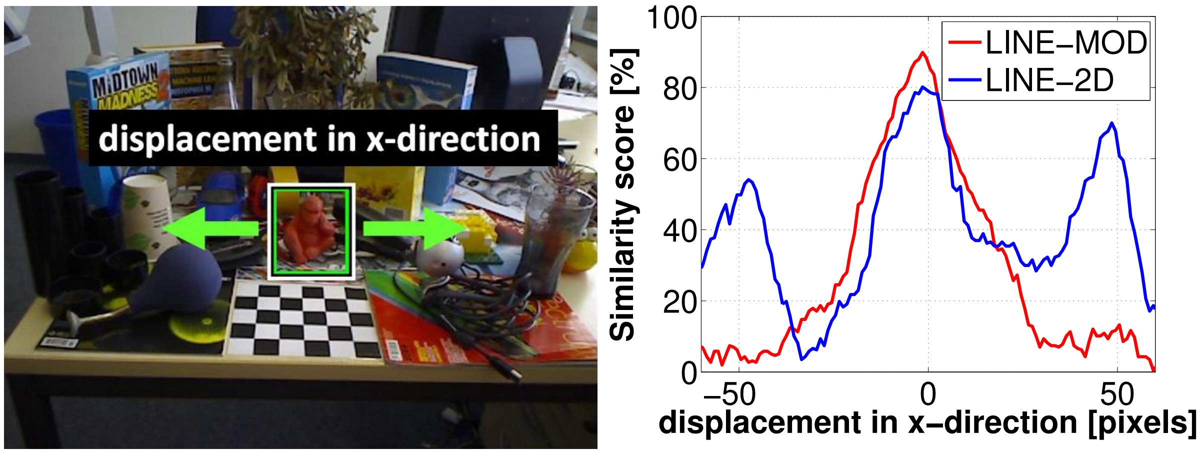


Figure 6.9: Combining many modalities results in a more discriminative response function. Here we compare LINE-MOD against LINE-2D on the shown image. We plot the response function of both methods with respect to the true location of the monkey. One can see that the response of LINE-MOD exhibits a single and discriminative peak whereas LINE-2D has several peaks which are of comparable height. This is one explanation why LINE-MOD works better and produces fewer false positives.

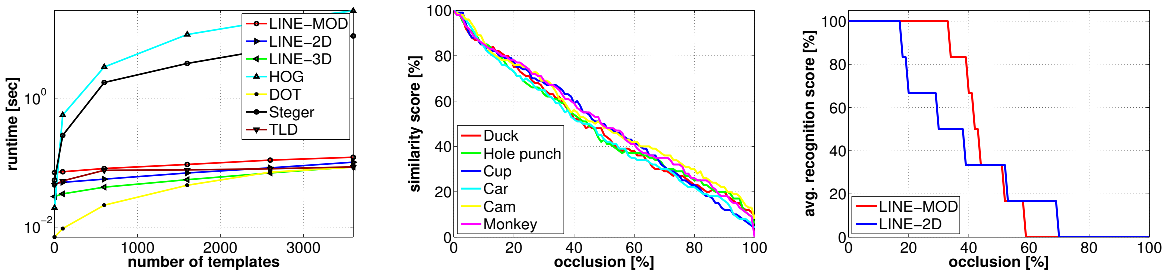


Figure 6.10: **Left:** Our new approach runs in real-time and can parse a 640×480 image with over 3000 templates at about 10 fps. **Middle:** Our new approach is linear with respect to occlusion. **Right:** Average recognition score for the six objects of Sec.6.2.8 with respect to occlusion.

One reason for this high robustness is the good separability of the multimodal approach as shown in the middle of Fig. 6.11 and Fig. 6.12: one can see that a specific threshold—about 80 in our implementation—separates almost all true positives well from almost all false positives. This has several advantages. First, we will detect almost all instances of the object by setting the threshold to this specific value. Second, we also know that almost every returned template with a similarity score above this specific value is a true positive. And third, the threshold is always around the same value which supports the conclusion that it might also work well for other objects. One hint why the novel multimodal approach has such a good separability property is given in Fig. 6.9. One can see that the response function has only one clear peak around the true location of the object while LINE-2D shows other peaks with almost the same height.

6.2.9 Speed

Learning new templates only requires extracting and storing multimodal features, which is almost instantaneous. Therefore, we concentrate on runtime performance.

The runtimes given in Fig. 6.10 show that the general LINE approach is real-time and can parse a VGA image with over 3000 templates with about 10 fps on the CPU. The small difference of computation times between LINE-MOD and LINE-2D and LINE-3D comes from the slightly slower preprocessing step of LINE-MOD that includes the two preprocessing steps of LINE-2D and LINE-3D.

DOT is initially faster than LINE but becomes slower as the number of templates increases. This is because

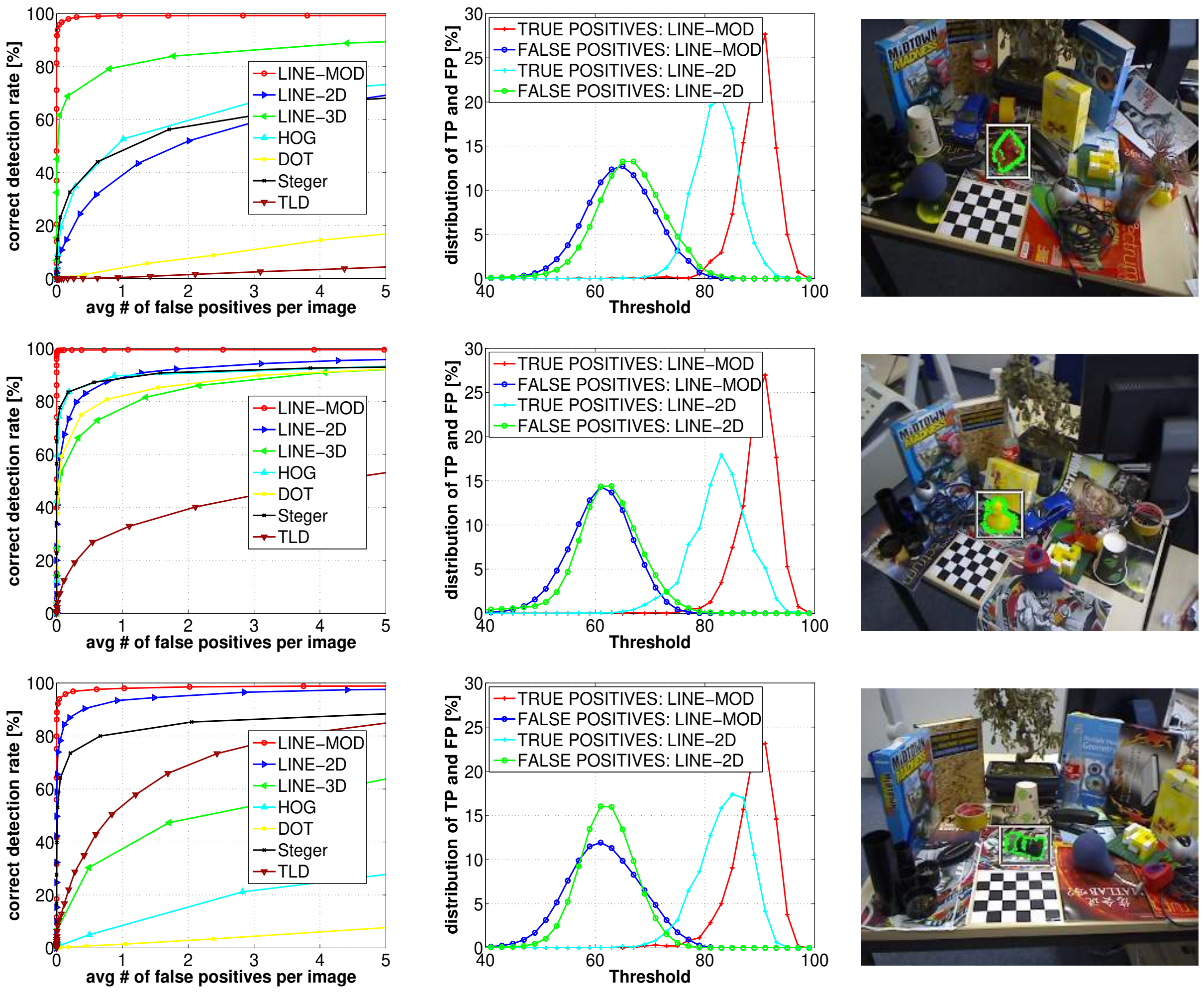


Figure 6.11: Comparison of LINE-MOD with LINE-2D, which is based on gradients, LINE-3D, which is based on normals, DOT [96], HOG [46], Steger [205] and TLD [117] on real 3D objects. Each row corresponds to a different sequence (made of over 2000 images each) on heavy cluttered background: A monkey, a duck and a camera. The approaches were learned on a homogeneous background. **Left:** Percentage of true positives plotted against the average percentage of false positives. The multimodal templates provide about the same recognition rates for all objects while the other approaches have a much larger variance depending on the object type. LINE-MOD outperforms the other approaches in most cases. **Middle:** The distribution of true and false positives plotted against the threshold. They are well separable from each other. **Right:** One sample image of the corresponding sequence shown with the object detected by LINE-MOD.

the runtime of LINE is independent of the template size whereas the runtime of DOT is not. Therefore, to handle larger objects DOT has to use larger templates which makes the approach slower once the number of templates increases.

Our implementation of Steger *et al.* is approximately 100 times slower than our LINE-MOD method. Note that we use 4 pyramid levels for more efficiency which is one of the reasons for the different speed improvement given in Sec. 6.2.6, where we assumed no image pyramid.

TLD uses a tree classifier similar to [170] which is the reason why the timings stay relatively equal with respect to the number of templates. Since this chapter is concerned with detection, we only evaluate the TLD detector for this experiment and not the tracking method.

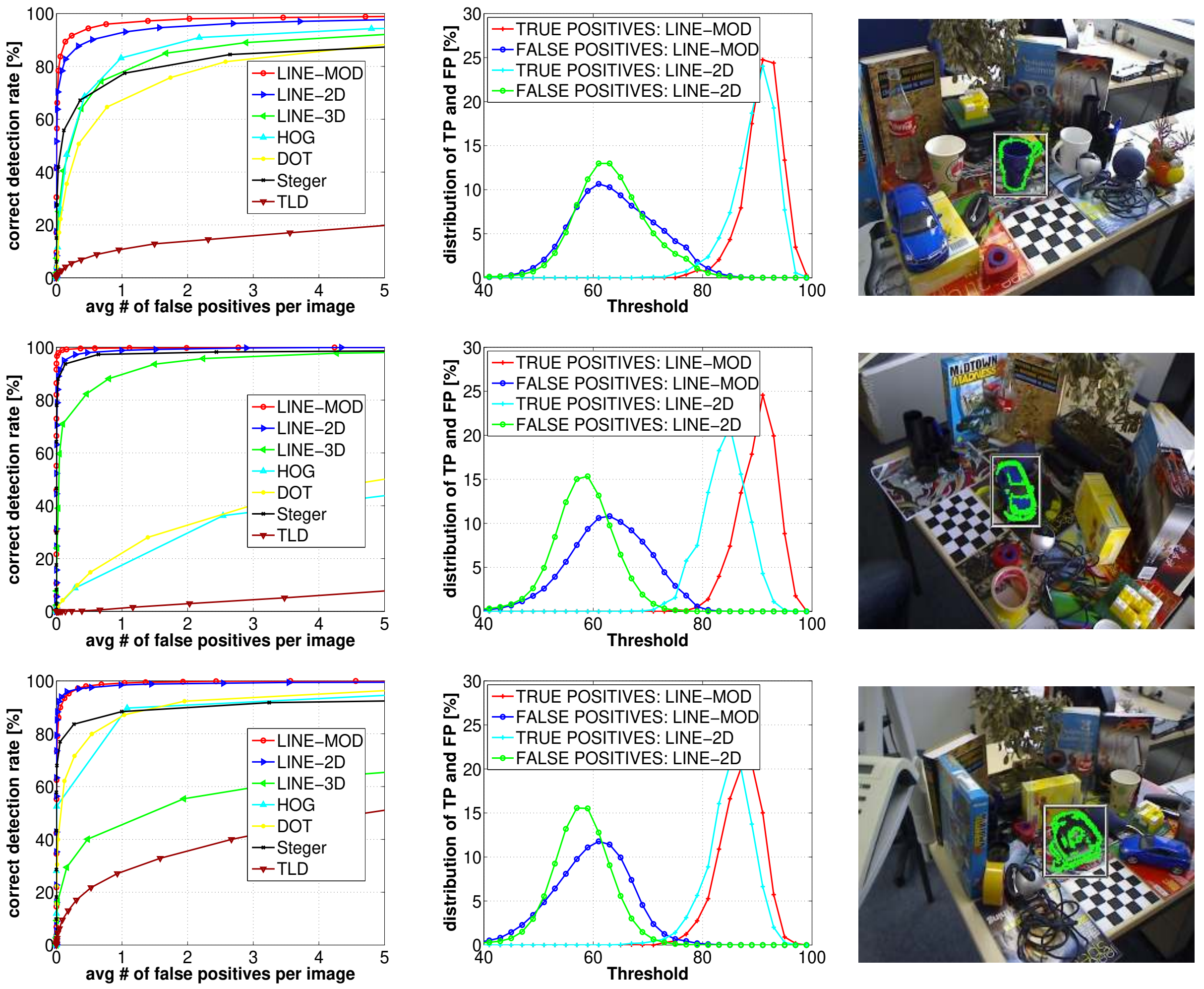


Figure 6.12: Same experiments as shown in Fig. 6.11, however for different objects: a cup, a car and a hole punch. For each object we tested the approaches on over 2000 images each.

Sequence	LINE-MOD	LINE-2D	LINE-3D	HOG	DOT	Steger	TLD
Toy-Monkey (2164 pics)	**97.9%–0.3%**	50.8%–49.1%	86.1%–13.8%	51.8%–48.2%	8.6%–91.4%	69.6%–30.3%	0.8%–99.1%
Camera (2173 pics)	**97.5%–0.3%**	92.8%–6.7%	61.9%–38.1%	18.2%–81.8%	1.9%–98.0%	96.9%–0.4%	53.3%–46.6%
Toy-Car (2162 pics)	**97.7%–0.0%**	96.9%–0.4%	95.6%–2.5%	44.1%–55.9%	34.0%–66.0%	83.6%–16.3%	0.1%–98.9%
Cup (2193 pics)	**96.8%–0.5%**	92.8%–6.0%	88.3%–10.6%	81.1%–18.8%	64.1%–35.8%	90.2%–8.9%	10.4%–89.6%
Toy-Duck (2223 pics)	**97.9%–0.0%**	91.7%–8.0%	89.0%–10.0%	87.6%–12.4%	78.2%–21.8%	92.2%–7.6%	28.0%–71.9%
Hole punch (2184 pics)	**97.0%–0.2%**	96.4%–0.9%	70.0%–30.0%	92.6%–7.4%	87.7%–12.0%	90.3%–9.7%	26.5%–73.4%

Table 6.1: True and false positive rates for different thresholds on the similarity measure of different methods. In some cases no hypotheses were given back so the sum of true and false positives can be lower than 100%. Our LINE-MOD approach obtains very high recognition rates at the cost of almost no false positives, and outperforms all the other approaches. The corresponding best values are shown in bold print.

6.2.10 Occlusions

We also tested the robustness of LINE-MOD with respect to occlusion. We added synthetic noise and illumination changes to the images, incrementally occluded the six different objects of Section 6.2.8 and measured the corresponding response values. As expected, the similarity measure used by LINE-MOD behaves linearly in the percentage of occlusion as reported in Fig. 6.10. This is a desirable property since it allows detection of partly occluded templates by setting the detection threshold with respect to the tolerated percentage of occlusion. We also experimented with real scenes where we first learned our six objects in front of a homogeneous background

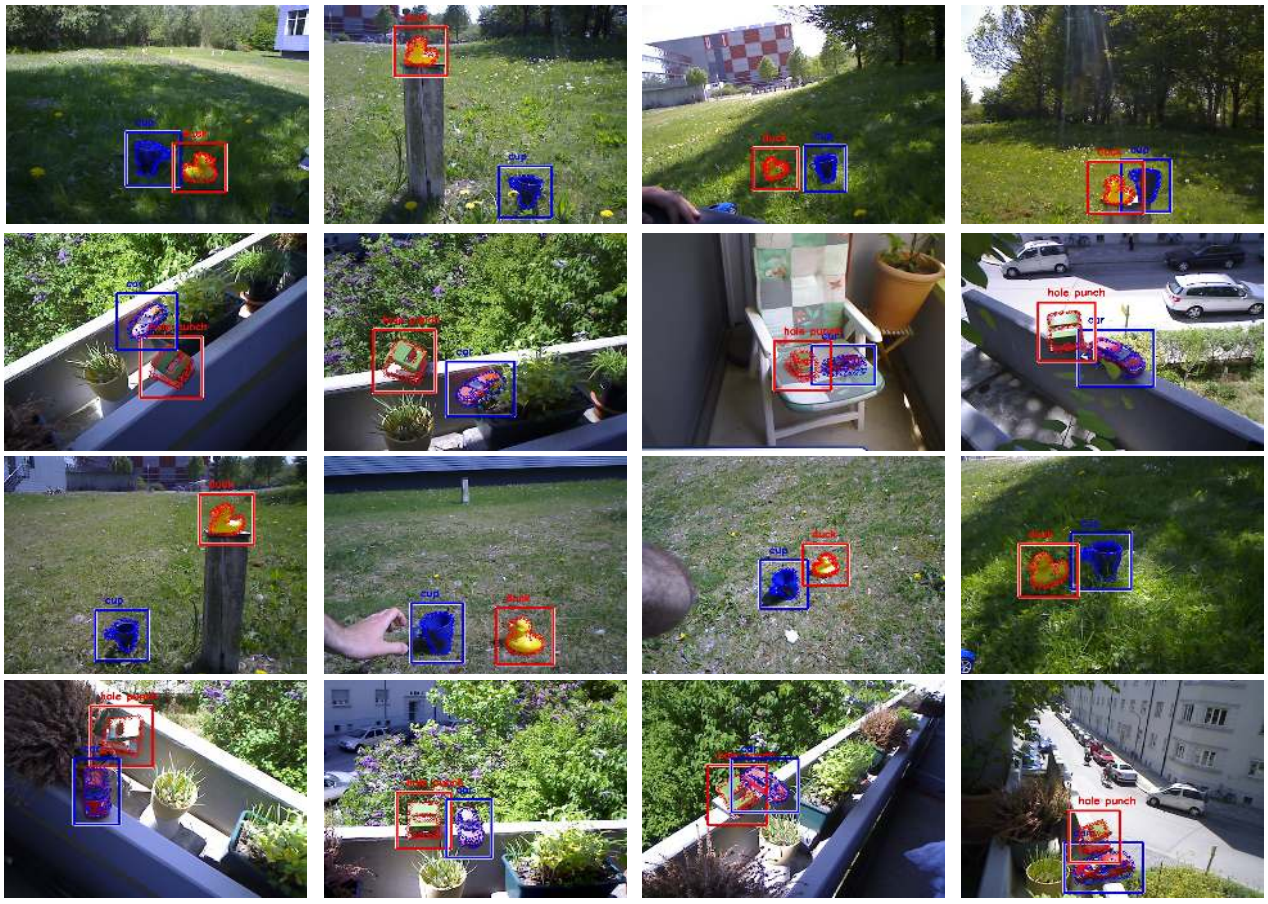


Figure 6.13: Different texture-less 3D objects are detected with LINE-2D in real-time under different poses in difficult outside scenes with partial occlusion, illumination changes and strong background clutter.

and then added heavy 2D and 3D background clutter. For recognition we incrementally occluded the objects. We define our object as correctly recognized if the template with the highest response is found within a fixed radius of the ground truth object location. The average recognition result is displayed in Fig. 6.10: Even with over 30% occlusion LINE-MOD is still able to recognize objects.

6.2.11 Number of Templates

Finally, we have to give the average number of templates needed to detect an arbitrary object from a large number of viewpoings. In our application, approximately 2000 templates are needed to detect an object with 360 degree tilt rotation, 90 degree inclination rotation and in-plane rotations of $\pm$ 80 degree. The tilt and inclination rotations are visualized as the half-sphere in Fig. 6.16. The detection works for scale changes in the range of $[1.0; 2.0]$.

6.2.12 Examples

In Figs. 6.13, 6.14 and 6.15 we detect texture-less objects in different heavy cluttered inside and outside scenes. The objects are detected under partial occlusion, drastic pose and illumination changes. In Fig. 6.13 and 6.14, we only use gradient features, whereas in Fig. 6.15, we also use 3D normal features. Note, that we could not apply LINE-MOD outside since the Primesense device was not able to produce a depth map under strong sunlight.

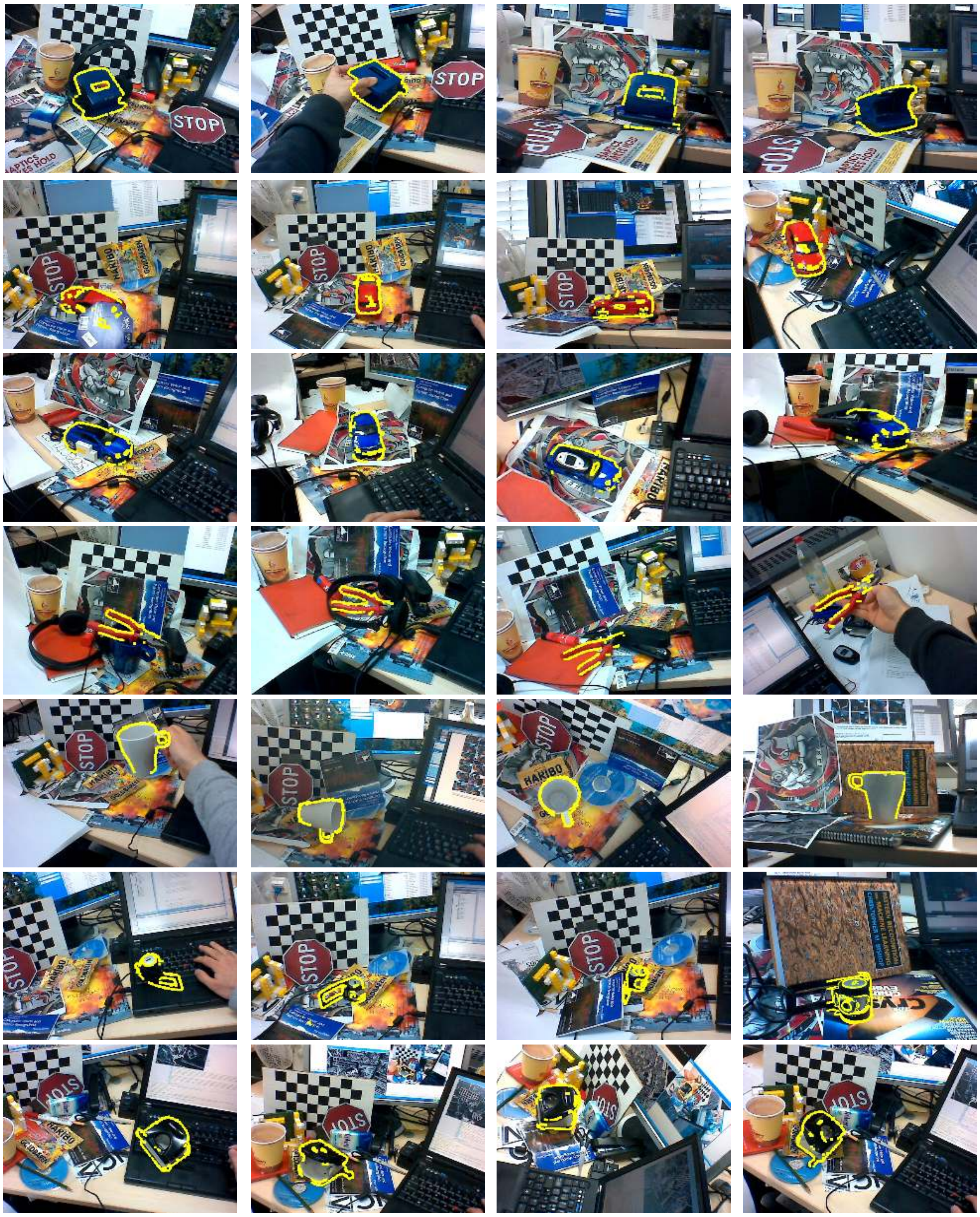


Figure 6.14: Different texture-less 3D objects are detected with LINE-2D in real-time under different poses on heavily cluttered background with partial occlusion.

6.3 Conclusion

We have presented a method to exploit different modalities for real-time object detection. Our novel approach is able to correctly detect 3D texture-less objects in real-time under heavy background clutter, illumination changes

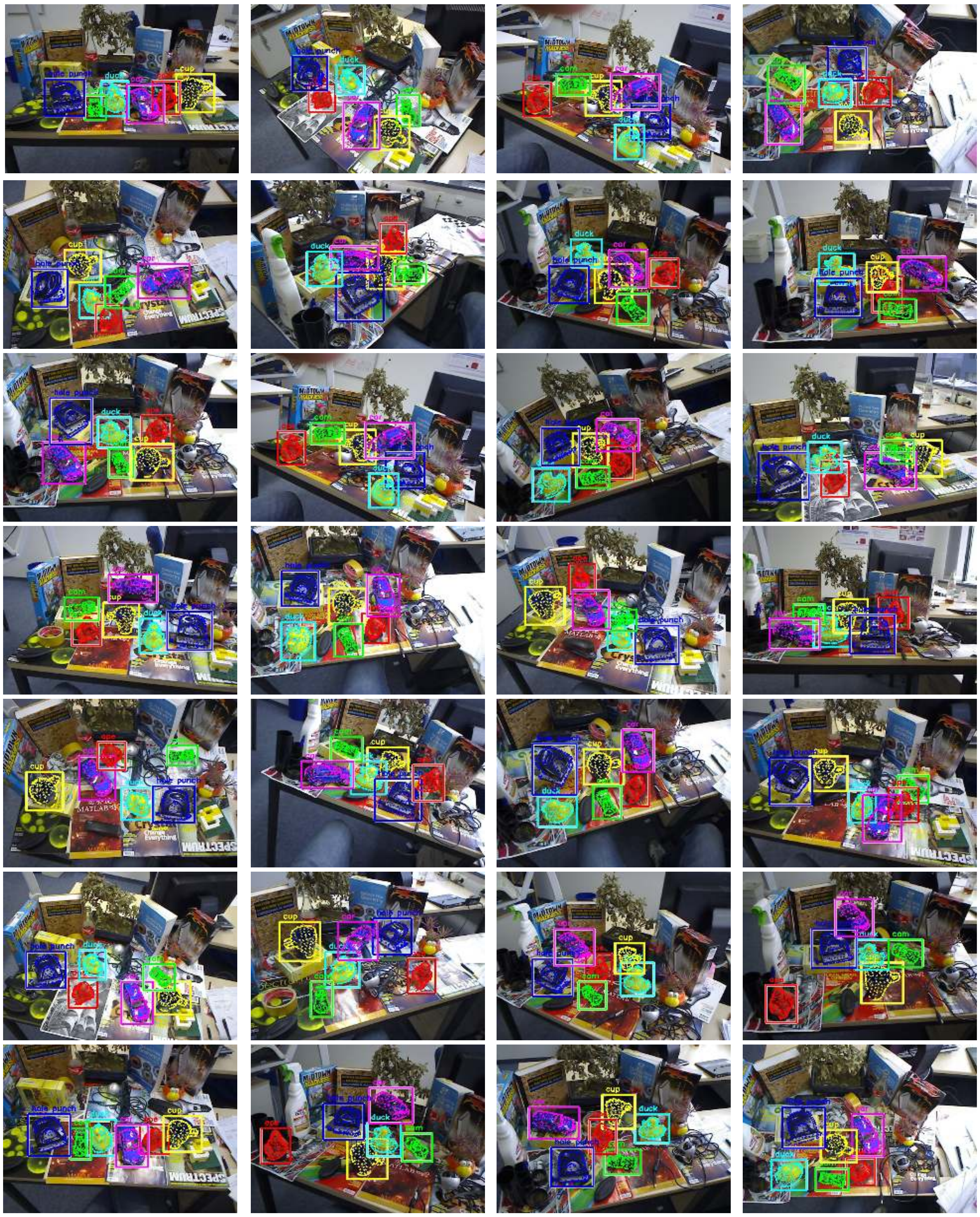


Figure 6.15: Different texture-less 3D objects detected simultaneously in real-time by our LINE-MOD method under different poses on heavily cluttered background with partial occlusion.

and noise with almost no false positives. We showed how to efficiently preprocess image and depth data to robustly integrate both cues into our approach. Furthermore, we demonstrated how to take advantage of the architecture of modern computers to build a fast but very discriminant representation of the input images that can be used to

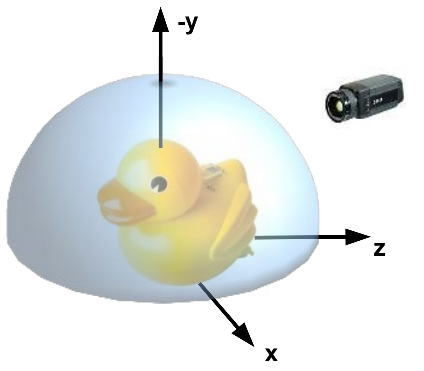


Figure 6.16: An object can be detected using approximately 2000 templates. The half-sphere visualizes the detection range. Additionally, in-plane rotations of $\pm$ 80 degree and scale changes in the range from $[1.0, 2.0]$ can be handled.

consider few thousands of arbitrarily sized and arbitrarily shaped templates in real-time. Additionally, we have shown that our approach outperforms state-of-the-art methods with respect to the combination of recognition rate and speed, especially in heavily cluttered environments.

The dataset introduced with the publication corresponding to this chapter, while not perfect, attracted the attention of some researchers to the problem of 3D detection of poorly textured objects [47, 31, 26, 220, 215]. The LINE methods and their predecessor DOT [96] were also used in other object detection works [216, 103, 176].

Chapter 7

Learning Descriptors for Object Recognition and 3D Pose Estimation

7.1 Introduction

The previous chapter introduced LineMOD, a fast descriptor of object templates, designed for 3D object detection. In this chapter, we described an approach, originally publised in [252], to learning to compute a suitable descriptor. We will show that this new descriptor outperforms LineMOD and HOG [46], another view descriptor. This work was developed together with Paul Wohlhart, a young postdoc in my group at TU Graz. This work is also one of our first use of Deep Learning.

Impressive results have been achieved in 3D pose estimation of objects from images during the last decade. However, current approaches cannot scale to large-scale problems because they rely on one classifier per object, or multi-class classifiers such as Random Forests, whose complexity grows with the number of objects. So far the only recognition approaches that have been demonstrated to work on large scale problems are based on Nearest Neighbor (NN) classification [158, 113, 50], because extremely efficient methods for NN search exist with an average complexity of $O(1)$ [161, 155].

Moreover, as argued in the previous chapter, Nearest Neighbor (NN) classification also offers the possibility to trivially add new objects, or remove old ones, which is not directly possible with neural networks, for example. However, to the best of our knowledge, such an approach has not been applied to the 3D pose estimation problem, while it can potentially scale to many objects seen under large ranges of poses. For example, [50] only focuses on object recognition without considering the 3D pose estimation problem.

For NN approaches to perform well, a compact and discriminative description vector is required. Such representations that can capture the appearance of an object under a certain pose have already been proposed [46, 98], however they have been handcrafted. Our approach is motivated by the success of recent work on feature point descriptor learning [30, 223, 147], which shows that it is possible to learn compact descriptors that significantly outperform handcrafted methods such as SIFT or SURF.

However, the problem we tackle in this chapter is more complex: While feature point descriptors are used only to find the points' identities, we here want to find both the object's identity and its pose. We therefore seek to learn a descriptor with the two following properties: a) The Euclidean distance between descriptors from two different objects should be large; b) The Euclidean distance between descriptors from the same object should be representative of the similarity between their poses. This way, given a new object view, we can recognize the object and get an estimate of its pose by matching its descriptor against a database of registered descriptors. New objects can also be added and existing ones removed easily. To the best of our knowledge, our method is the first one that learns to compute descriptors for object views.

Our approach is related to manifold learning, but the key advantage of learning a direct mapping to descriptors is that we can use efficient and scalable Nearest Neighbor search methods. This is not possible for previous methods relying on geodesic distances on manifolds. Moreover, while previous approaches already considered similar properties to a) and b), to the best of our knowledge, they never considered both simultaneously, while it is critical for efficiency. Combining these two constraints in a principled way is far from trivial, but we show it can

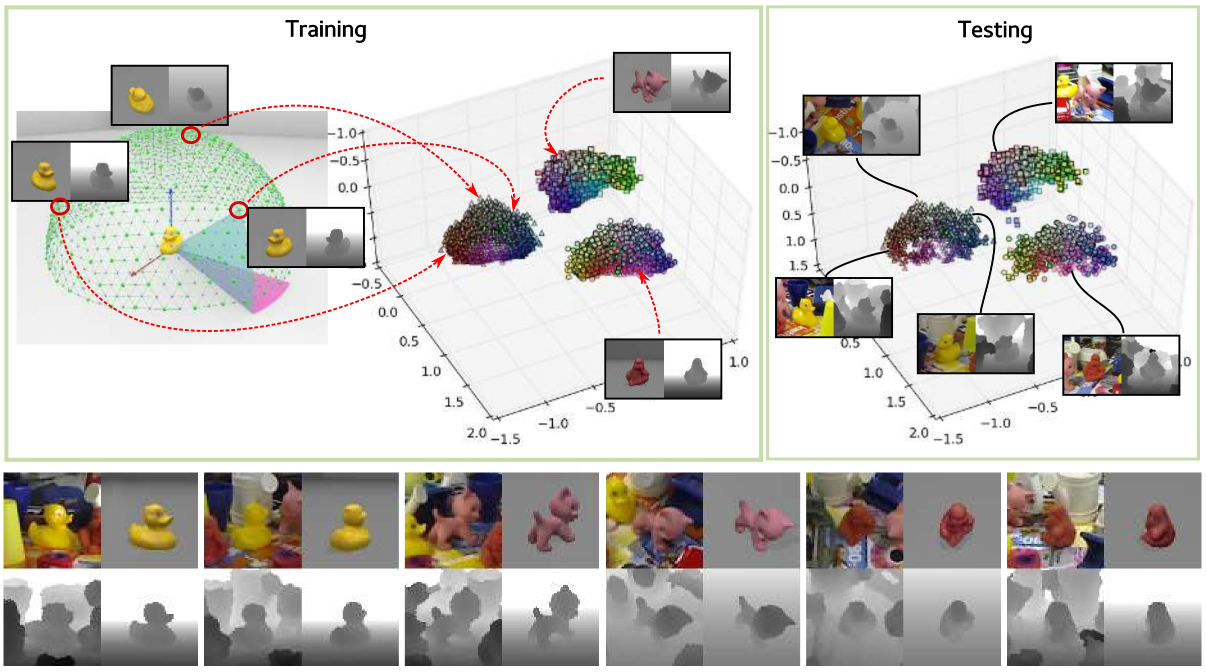


Figure 7.1: Three-dimensional descriptors for several objects under many different views computed by our method on RGB-D data. **Top-left:** The training views of different objects are mapped to well-separated descriptors, and the views of the same object are mapped to descriptors that capture the geometry of the corresponding poses, even in this low dimensional space. **Top-right:** New images are mapped to locations corresponding to the object and 3D poses, even in the presence of clutter. **Bottom:** Test RGB-D views and the RGB-D data corresponding to the closest template descriptor.

be done by training a Convolutional Neural Network [129] using simple constraints to compute the descriptors. As shown in Fig. 7.1, this results in a method that nicely untangles the views of different objects into descriptors that capture the identities and poses of the objects.

We evaluate our approach on instance recognition and pose estimation data with accurate ground truth and show significantly improved results over related methods. Additionally we perform experiments assessing the ability of the method to generalize to unseen objects showing promising results.

7.2 Related Work

Our work is related to several aspects of Computer Vision, and we focus here on the most relevant and representative work.

Our approach is clearly in the framework of 2D view specific templates [101], which is conceptually simple, supported by psychophysical experiments [218], and was successfully applied to various problems and datasets over the last years [157, 145, 95, 81, 50, 176].

However, most of these works rely on handcrafted representations of the templates, for example HOG [46] or LineMOD [98]. In particular, LineMOD was designed explicitly in the context of object detection and pose estimation. However these handcrafted representations are suboptimal compared to statistically learned features. [145, 81, 176] show how to build discriminative models based on these representations using SVM or boosting applied to training data. [145, 176] do not consider the pose estimation problem, while [81] focuses on this problem only, with a discriminatively trained mixture of HOG templates. Exemplars were recently used for 3D object detection and pose estimation in [15], but still rely on a handcrafted representation.

As mentioned in the introduction, our work is influenced by work developed for keypoint descriptor learning. Some of these methods are applied to existing descriptors to make them more discriminative, such as in [78, 207], but others are trained directly on image data. [30] introduces datasets made of “positive pairs” of patches corresponding to the same physical points and “negative pairs” of patches corresponding to different points. It is used for example in [223] to learn a binary descriptor with boosting. [147] uses a “siamese” architecture [42] to train a neural network to compute discriminative descriptors. Our approach is related to this last work, but the notion of pose is absent in their case. We show how to introduce this notion by using triplets of training examples in addition to only pairs.

Instead of relying on rigid templates as we do, many works on category recognition and pose estimation rely on part-based models. [187] pioneered this approach, and learned canonical parts connected by a graph for object recognition and pose estimation. [173] extends the Deformable Part Model to 3D object detection and pose estimation. [172] uses contours as parts. One major drawback of such approaches is that the complexity is typically linear with the number of objects. It is also not clear how important the “deformable” property really is for the recognition, and rigid templates seem to be sufficient [51].

Our approach is also related to manifold learning [174]. For example, [184] learns an embedding that separates extremely well the classes from the MNIST dataset of digit images, but the notion of pose is absent. [86] learns either for different classes, also on the MNIST dataset, or for varying pose and illumination, but not the two simultaneously. More recently, [16] proposes a method that separates manifolds from different categories while being able to predict the object poses, and also does not require solving an inference problem, which is important for efficiency. However, it relies on a discretisation of the pose space in a few classes, which limits the possible accuracy. It also relies on HOG for the image features, while we learn the relevant image features.

Finally, many works focus as we do on instance recognition and pose estimation, as it has important applications in robotics for example. [98] introduced LineMOD, a fast but handcrafted presentation of template for dealing with poorly textured objects. The very recent [26, 220] do not use templates but rely on recognition of local patches instead. However they were demonstrated on RGB-D images, and local recognition is likely to be much more challenging on poorly textured objects when a depth information is not available. [127] also expects RGB-D images, and uses a tree for object recognition, which however still scales linearly in the numbers of objects, categories, and poses.

7.3 Method

Given a new input image x of an object, we want to correctly predict the object’s class and 3D pose. Because of the benefits discussed above, such as scalability and ability to easily add and remove objects, we formulate the problem as a k-nearest neighbor search in a descriptor space: For each object in the database, descriptors are calculated for a set of template views and stored along with the object’s identity and 3D pose of the view. In order to get an estimate for the class and pose of the object depicted in the new input image, we can compute a descriptor for x and search for the most similar descriptors in the database. The output is then the object and pose associated with them.

We therefore introduce a method to efficiently map an input image to a compact and discriminative descriptor that can be used in the nearest neighbor search according to the Euclidean distance. For the mapping, we use a Convolutional Neural Network (CNN) that is applied to the raw image patch as input and delivers the descriptor as activations of the last layer in one forward pass.

We show below how to train such a CNN to enforce the two important properties already discussed in the introduction: a) The Euclidean distance between descriptors from two different objects should be large; b) The Euclidean distance between descriptors from the same object should be representative of the similarity between their poses.

7.3.1 Training the CNN

In order to train the network we need a set $\mathcal{S}_{\text{train}}$ of training samples, where each sample $s = (x, c, p)$ is made of an image x of an object, which can be a color or grayscale image or a depth map, or a combination of the two; the identity c of the object; and the 3D pose p of the object relative to the camera.

Additionally, we define a set $\mathcal{S}_{\text{db}}$ of templates where each element is defined in the same way as a training sample. Descriptors for these templates are calculated and stored with the classifier for k-nearest neighbor search. The template set can be a subset of the training set, the whole training set or a separate set. Details for the creation of training and template data are given in the implementation section.

7.3.2 Defining the Cost Function

We argue that a good mapping from the images to the descriptors should be so that the Euclidean distance between two descriptors of the same object and similar poses are small and in every other case (either different objects or different poses) the distance should be large. In particular, each descriptor of a training sample should have a small distance to the one template descriptor from the same class with the most similar pose and a larger distance to all descriptors of templates from other classes, or the same class but less similar pose.

We enforce these requirements by minimizing the following objective function over the parameters w of the CNN:

$$\mathcal{L} = \mathcal{L}_{\text{triplets}} + \mathcal{L}_{\text{pairs}} + \lambda ||w'||_2^2 \ . \tag{7.1}$$

The last term is a regularization term over the parameters of the network: w' denotes the vector made of all the weights of the convolutional filters and all nodes of the fully connect layers, except the bias terms. We describe the first two terms $\mathcal{L}_{\text{triplets}}$ and $\mathcal{L}_{\text{pairs}}$ below.

Triplet-wise terms

We first define a set $\mathcal{T}$ of triplets (s_i, s_j, s_k) of training samples. Each triplet in $\mathcal{T}$ is selected such that one of the two following conditions is fulfilled:

- either s_i and s_j are from the same object and s_k from another object, or
- the three samples s_i, s_j, and s_k are from the same object, but the poses p_i and p_j are more similar than the poses p_i and p_k.

These triplets can therefore be seen as made of a pair of similar samples (s_i and s_j) and a pair of dissimilar ones (s_i and s_k). We introduce a cost function for such a triplet:

$$c(s_i, s_j, s_k) = \max\left(0, 1 - \frac{||f_w(x_i) - f_w(x_k)||_2}{||f_w(x_i) - f_w(x_j)||_2 + m}\right) \ , \tag{7.2}$$

where $f_w(x)$ is the output of the CNN for an input image x and thus our descriptor for x, and m is a margin. We can now define the term $\mathcal{L}_{\text{triplets}}$ as the sum of this cost function over all the triplets in $\mathcal{T}$:

$$\mathcal{L}_{\text{triplets}} = \sum_{(s_i, s_j, s_k) \in \mathcal{T}} c(s_i, s_j, s_k) \ . \tag{7.3}$$

It is easy to check that minimizing $\mathcal{L}_{\text{triplets}}$ enforces our two desired properties in one common framework.

The margin m serves two purposes. First, it introduces a margin for the classification. It also defines a minimum ratio for the Euclidean distances of the dissimilar pair of samples and the similar one. This counterbalances the weight regularization term, which naturally contracts the output of the network and thus the descriptor space. We set m to 0.01 in all our experiments.

The concept of forming triplets from similar and dissimilar pairs is adopted from the field of metric learning, in particular, the method of [246], where it is used to learn a Mahalanobis distance metric. Note also that our definition of the cost is slightly different from the one in [241], which uses

$$c(s_i, s_j, s_k) = \max\left(0, m + ||f_w(x_i) - f_w(x_j)||_2^2 - ||f_w(x_i) - f_w(x_k)||_2^2\right) \ ,$$

where m is set to 1. Our formulation does not suffer from a vanishing gradient when the distance of the dissimilar pair is very small. Also the increase of the cost with the distance of the similar pair is bounded, thus putting more focus on local interactions. In practice, however, with proper initialization and selection of m both formulations deliver similar results.

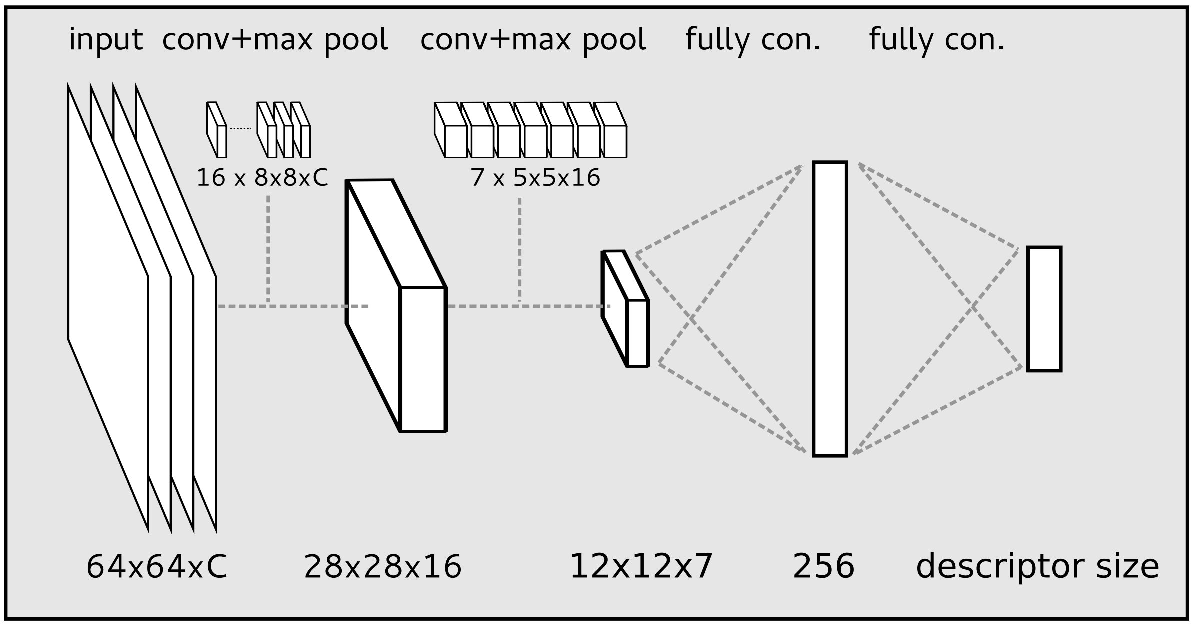


Figure 7.2: Network structure: We use a CNN made of two convolutional layers with subsequent 2×2 max pooling layers, and two fully connected layers. The activations of the last layer form the descriptor for the input image.

Pair-wise terms

In addition to the triplet-wise terms, we also use pair-wise terms. These terms make the descriptor robust to noise and other distracting artifacts such as changing illumination. We consider the set $\mathcal{P}$ of pairs (s_i, s_j) of samples from the same object under very similar poses, ideally the same, and we define the $\mathcal{L}_{\text{pairs}}$ term as the sum of the squared Euclidean distances between the descriptors for these samples:

$$\mathcal{L}_{\text{pairs}} = \sum_{(s_i,s_j)\in\mathcal{P}} ||f_w(x_i) - f_w(x_j)||_2^2 \ . \tag{7.4}$$

This term therefore enforces the fact that for two images of the same object and same pose, we want to obtain two descriptors which are as close as possible to each other, even if they are from different imaging conditions: Ideally we want the same descriptors even if the two images have different backgrounds or different illuminations, for example. As will be discussed in more detail in Section 7.4.1, this also allows us to use a mixture of real and synthetic images for training.

Note that we do not consider dissimilar pairs unlike work in keypoint descriptors learning for example. With dissimilar pairs the problem arises how strong to penalize a certain distance between the two samples, given their individual labels. Using triplets instead gives the possibility to only consider relative dissimilarity.

7.3.3 Implementation Aspects

The exact structure of the network we train to compute the descriptors is shown in Figure 7.2. It consists of two layers that perform convolution of the input with a set of filters, max-pooling and sub-sampling over a 2×2 area and a rectified linear (ReLU) activation function, followed by two fully connected layers. The first fully connected layer also employs a ReLU activation, the last layer has linear output and delivers the final descriptor.

We optimize the parameters w of the CNN by Stochastic Gradient Descent on mini-batches with Nesterov momentum [214]. Our implementation is based on Theano [20].

The implementation of the optimization needs some special care: Since we are working with mini-batches, the data corresponding to each pair or triplet has to be organized such as to reside within one mini-batch. The most straightforward implementation would be to place the data for each pair and triplet after each other, calculate the resulting gradient wrt. the network's parameters individually and sum them up over the mini-batch. However, this would be inefficient since descriptors for templates would be calculated multiple times if they appear in more than one pair or triplet.

To assemble a mini-batch we start by randomly taking one training sample from each object. Additionally, for each of them we add its template with the most similar pose, unless it was already included in this mini-batch. This is iterated until the mini-batch is full. However, this procedure can lead to very unequal numbers of templates per object if, for instance, all of the selected training samples have the same most similar template. We make sure that for each object at least two templates are included by adding a random one if necessary. Pairs are then formed by associating each training sample with its closest template. Additionally, for each training sample in the mini-batch we initially create three triplets. In each of them the similar template is set to be the one with the closest pose and the dissimilar sample is either another, less similar template from the same object or any template of a different object.

During the optimization, after the first set of epochs, we perform boot-strapping of triplets within each mini-batch to focus on the difficult samples: For each training sample we add two additional triplets. The similar template is again the closest one. The dissimilar ones are those templates that currently have the closest descriptors, one from the same object but different pose and one from all the other objects.

Another aspect to take care of is the fact that the objective function must be differentiable with respect to the parameters of the CNN, while the derivative of the square root—used in the triplet-wise cost—is not defined for a distance of 0. Our solution is to add a small constant ϵ before taking the square root. Another possible approach [241] is to take the square of the norm. However, this induces the problem (mentioned in Section 7.3.2) that for very small distances of the dissimilar pair, the gradient becomes very small and vanishes for zero distance.

7.4 Evaluation

We compare our approach to LineMOD and HOG on the LineMOD dataset [95]. This dataset contains training and test data for object recognition and pose estimation of 15 objects, with accurate ground truth. It comes with a 3D mesh for each of the objects. Additionally, it also provides sequences of RGB images and depth maps recorded with a Kinect sensor.

7.4.1 Dataset Compilation

We train a CNN using our method on a mixture of synthetic and real world data. As in [98], we create synthetic training data by rendering the mesh available for each of the objects in the dataset from positions on a half-dome over the object, as shown in Fig. 7.1 on the left. The viewpoints are defined by starting with a regular icosahedron and recursively subdividing each triangle into 4 sub-triangles. For the template positions the subdivison is applied two times. After removing the lower half-sphere we end up with 301 evenly distributed template positions. Additional training data is created by subdividing one more time, resulting in 1241 positions.

From each pose we render the object standing on a plane over an empty background using Blender[1]. We parameterize the object pose with the azimuth and elevation of the camera relative to the object. We store the RGB image as well as the depth map.

For the real world data we split the provided sequences captured with the Kinect randomly into a training and a test set. We ensure an even distribution of the samples over the viewing hemisphere by taking two real world images close to each template, which results roughly in a 50/50 split of the data into training and test. Preliminary experiments showed very little to no variance over the different train/test splits and, thus, all results presented here report runs on one random split, fixed for each experiment.

The whole training data set is augmented by making multiple copies with added noise. On both RGB and depth channel we add a small amount of Gaussian noise. Additionally, for the synthetic images, we add larger fractal noise on the background, to simulate diverse backgrounds.

Note that the template views, which are ultimately used in the classification are purely synthetic and noise-free renderings on clean backgrounds. The algorithm, thus, has to learn to map the noisy and real world input data to the same location in descriptor space as the clean templates.

[1]http://www.blender.org

As pointed out in [98] some of the objects are rotationally invariant, to different degrees. Thus, the measure of similarity of poses used for the evaluation and, in our case to define pairs and triplets, should not consider the azimuth of the viewing angle for those objects. We treat the *bowl* object as fully rotationally invariant. The classes *eggbox*, *glue* are treated as symmetric, meaning a rotation by $180°$ around the z-axis shows the same pose again. The *cup* is a special case because it looks the same from a small range of poses, but from sufficient elevation such that the handle is visible, the exact pose could be estimated. We also treat it as rotationally invariant, mainly to keep the comparison to LineMOD fair.

We extract a patch centered at the object and capturing a fixed size window in 3D at the distance of the object's center. In order to also address the detection part in a sliding window manner, it would be necessary to extract and test several scales. However, only a small range of scales needs to be considered, starting with a maximal one, defined by the depth at the center point, and going down until the center of the object is reached.

Before applying the CNN we normalize the input images. RGB images are normalized to the usual zero mean, unit variance. For depth maps we subtract the depth at the center of the object, scale down such that 20cm in front and behind the object's center are mapped to the range of $[-1, 1]$ and clip everything beyond that range.

The test sequences captured with the Kinect are very noisy. In particular, there are many regions with undefined depth, introducing very large jumps for which the convolutional filters with ReLU activation functions might output overly strong output values. Therefore, we pre-process the test data by iteratively applying median filters in a 3×3 neighborhood, but only on the pixels for which the depth is available, until all gaps are closed.

7.4.2 Network Optimization

For the optimization we use the following protocol: We initially train the network on the initial dataset for 400 epochs, an initial learning rate of 0.01 and a momentum of 0.9. Every 100 epochs the learning rate is multiplied by 0.9. Then we perform two rounds of bootstrapping triplet indices as explained in Section 7.3.3, and for each round we train the CNN for another 200 epochs on the augmented training set. In the end we train another 300 epochs with the learning rate divided by 10 for final fine-tuning. The regularization weight λ is set to 10^{-6} in all our experiments.

7.4.3 LineMOD and HOG

We compare our learned descriptors to the LineMOD descriptor and HOG as a baseline, as it is widely used as representation in the related work. For LineMOD we use the publicly available source code in OpenCV. We run it on the same data as our method, except for the median filter depth inpainting and normalization: LineMOD handles the missing values internally and performed better without these pre-processing operations.

For HOG we also use the publicly available implementation in OpenCV. We extract the HOG descriptors from the same data we use with our CNN. We use a standard setup of a 64×64 window size, 8×8 cells, 2×2 cells in a block and a block stride of 8, giving a 1764-dimensional descriptor per channel. We compute descriptors on each RGB and depth channel individually and stack them. For evaluation we normalize all descriptors to length 1 and take the dot product between test and template descriptors as similarity measure.

7.4.4 Manifolds

Fig. 7.1 plots the views of three objects after being mapped into 3-dimensional descriptors, for visualization purposes. As can be seen, not only the descriptors from the different objects are very well separated, but they also capture the geometry of the corresponding poses. This means that the distances between descriptors is representative of the distances between the corresponding poses, as desired.

For longer descriptors we show an evaluation of the relation between distances of descriptors and similarity between poses in Fig. 7.3. For each object, we computed the distances between every sample in the test set and every template for the same object in the training set, as well as the angles between their poses. We then plot a two-dimensional histogram over these angle/distance pairs. Correlation between small angles and large distances indicates the risk of missed target templates, and correlation between large angles and small distances the risk of incorrect matches. Ideally the histograms should therefore have large values only on the diagonal.

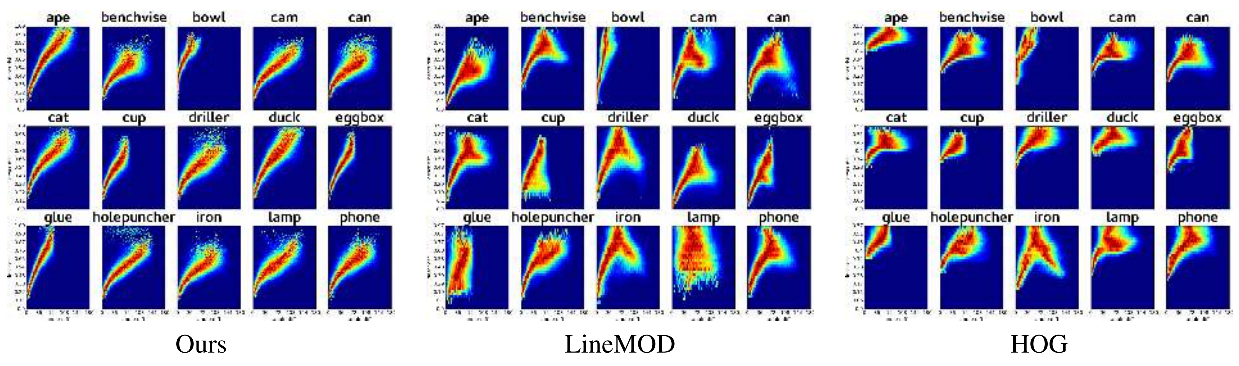


Figure 7.3: Histograms of the correlations between Pose Similarity (x-axis) and Descriptor Distance (y-axis) for each of the 15 objects of the LineMOD dataset on RGB-D data, as described in Section 7.4.4. Distances in the descriptor space are much more representative of the similarity between poses with our method than with LineMOD or HOG.

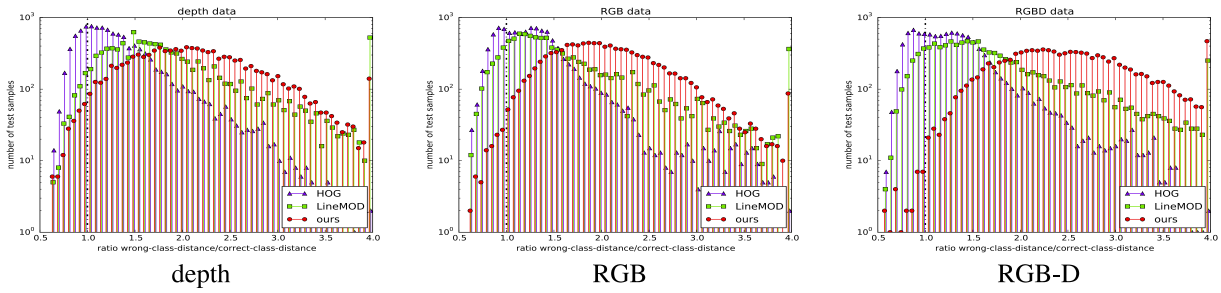


Figure 7.4: Evaluation of the class separation, over the 15 objects of the LineMOD dataset. The histograms plot the ratios between the distance to the closest template from the correct object and the distance to the closest template from any other object, for depth, RGB, and RGB-D data. Ratios above 4 are clipped; the y-axis is scaled logarithmically. Our method has considerably fewer samples for which the ratio is below one, which indicates less confusion between the objects.

The histograms for the descriptors computed with our method clearly show that the distance between descriptors increase with the angle between the views, as desired, while the histograms for LineMOD and HOG show that these descriptors are much more ambiguous.

Additionally, the ability of the descriptors to separate the different classes is evaluated in Fig. 7.4. For every test sample descriptor we compute the distance to the closest template descriptor of the same object and the closest from any other object and plot a histogram over those ratios. Clearly, descriptors obtained with our method exhibit a larger ratio for most samples and thus separate the objects better.

7.4.5 Retrieval Performance

What we ultimately want from the descriptors is that nearest neighbors are from the same class and have similar poses. In order to evaluate the performance we thus perform the following comparisons.

The scores reported for LineMOD in [98] represent the accuracy of the output of the whole processing pipeline, including the descriptor calculation, retrieval of similar templates, pruning the set with heuristics and refinement of the pose for a set of candidate matches by aligning a voxel model of the object. The contribution of this work is to replace the descriptors for the retrieval of templates with similar pose. Thus, we evaluate and compare this step in separation of the rest of the pipeline.

Evaluation Metric For each test sample we consider the k-nearest neighbors according to the descriptors and similarity metric of each method, the Euclidean distance in our case, the dot product for HOG, and the matching score of LineMOD. Among those k nearest templates we search for the one with the best closest pose to the test sample's pose, assuming that this one would perform best in the subsequent refinement process and thus finally

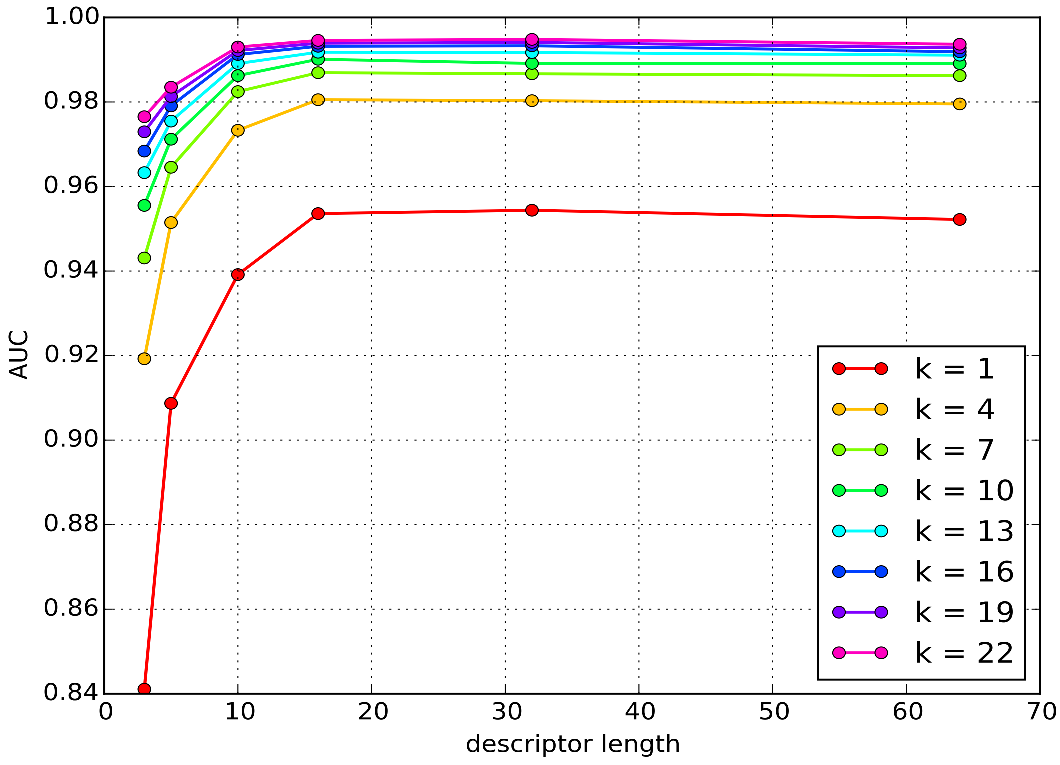


Figure 7.5: Evaluation of the descriptor length on depth data. Only 16 values are sufficient to reliably represent an object view, after which the performance plateaus.

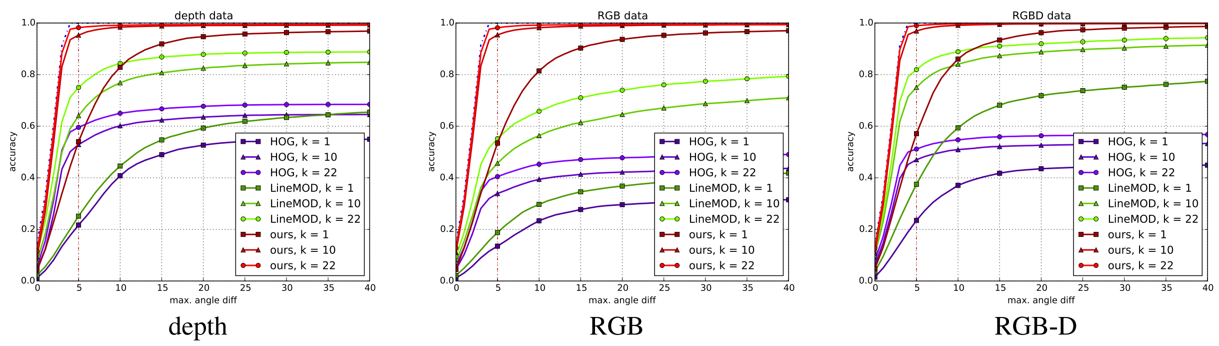


Figure 7.6: Performance evaluation and comparison to LineMOD and HOG on all 15 objects of the LineMOD dataset and depth, RGB, and RGB-D data. Each graph plots the accuracy over the y-axis for given a maximal allowed error of the resulting object's pose on the x-axis. Curves for different k are computed by taking k-nearest neighbors and selecting the one with the best matching pose. For our method, the descriptor length was set to 32 for depth, RGB, and RGB-D data. HOG uses 1764 values per channel, and LineMOD uses a length of 63 for depth and RGB data, and of 126 for RGB-D data.

be selected. The pose error is measured by the angle between the two positions on the viewing half-sphere. We define the accuracy as the percentage of test images for which the best angle error is below a certain threshold. The minimum angle error for which perfect accuracy can theoretically be reached is $5°$, because that is the maximal distance of a test image to its closest template.

Descriptor Length In Fig. 7.5 we evaluate the influence of the length of the descriptors learned on depth data. As can be seen the maximal performance is already reached with a 16 dimensional descriptor, while the length of the HOG descriptor is 1764. Thus, we use a 16 dimensional descriptor for all of the following experiments, including for the RGB and RGB-D data.

		Depth				RGB				RGB-D			
	k	5°	20°	40°	180°	5°	20°	40°	180°	5°	20°	40°	180°
ours	1	**54.4**	**94.7**	**96.9**	**98.1**	**53.4**	**93.7**	**97.0**	**99.1**	**57.1**	**96.2**	**98.7**	**99.8**
LineMOD	1	25.1	59.3	65.4	69.5	18.8	36.8	41.9	49.6	37.5	71.8	77.4	83.7
HOG	1	21.7	52.7	54.9	55.3	13.5	29.6	31.5	33.6	23.5	43.5	44.9	46.2
ours	22	**98.2**	**99.4**	**99.5**	**99.6**	**98.2**	**99.5**	**99.6**	**99.7**	**99.0**	**99.9**	**99.9**	**99.9**
LineMOD	22	75.0	87.9	88.8	89.5	55.2	74.0	79.3	83.5	81.9	92.0	94.3	96.4
HOG	22	59.5	67.7	68.5	68.9	40.5	47.8	49.0	50.9	51.1	56.3	56.8	57.5

Table 7.1: Performance comparison to LineMOD and HOG on all 15 objects of the LineMOD dataset and depth, RGB, and RGB-D data, for several tolerated angle errors. Our method systematically outperforms the other representations. The value at 180° indicates the object recognition rate when the pose is ignored.

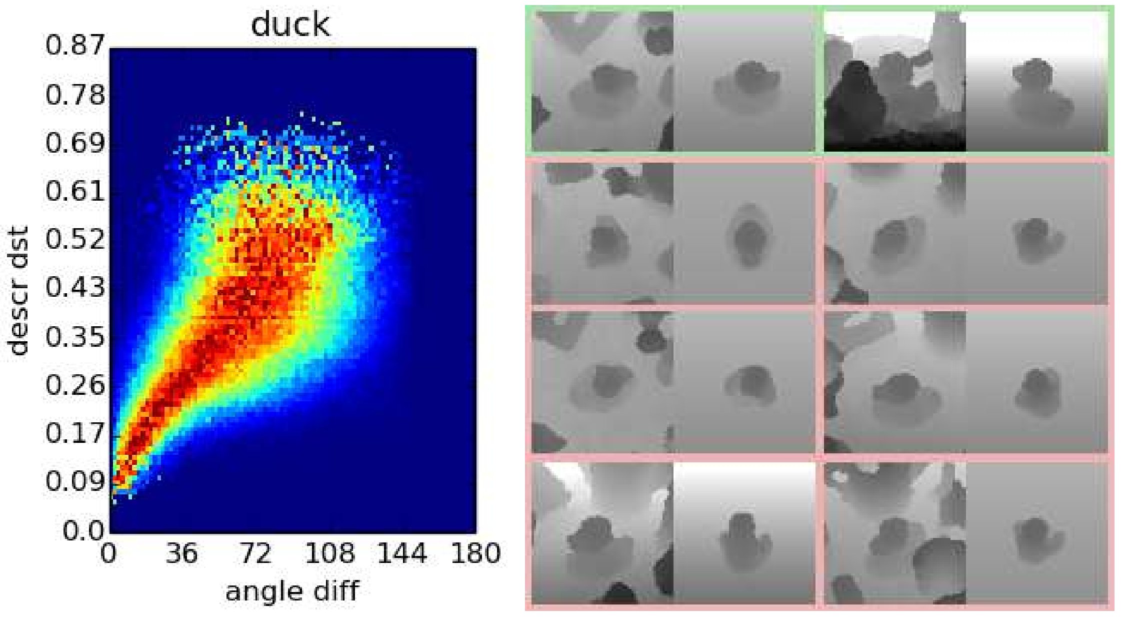


Figure 7.7: Generalization to objects not seen during training. **Left:** Histogram of correlation between Pose Similarity and Descriptor Distance for the *duck* that was not included during training for this experiment. The recognition rate is only slightly reduced compared to when the duck is used during training. **Right:** Difficult examples of correct and mis-classifications. The duck is mostly confused with the ape, which looks similar in the depth image under some angles.

Results We evaluate all three approaches on depth, RGB, and RGB-D data. Fig. 7.6 and Table 7.1 summarize the results. For *depth maps*, results are shown in Fig. 7.6-depth. When only considering 1 nearest neighbor we achieve a recognition rate of 98.1%, as opposed to the 69.5% achieved by the LineMOD descriptor and a pose error of less than 20° for 94.7% of the test samples (59.3% for LineMOD). Fig. 7.6-RGB shows results for training and testing on *color images*. While both LineMOD and HOG cannot reach the performance they obtain on the depth data on RGB alone, our descriptor performs almost identically in this setup. Finally, Fig. 7.6-RGBD shows results for training and testing on the combination of color images and depth maps. While LineMOD takes advantage of the combination of the two modalities, it is clearly outperformed by our descriptor, as taking the single nearest neighbor exhibits a pose error below 20° for 96.2% of the test samples and an overall recognition rate of 99.8%, an almost perfect score.

7.4.6 Generalization

In this last experiment, we show that our descriptor can generalize to unseen objects. This evaluation was performed using depth only. To do so, we train the CNN on 14 out of the 15 objects. We then perform the evaluation just as above by computing descriptors for the new object. As can be seen from the histogram of Fig. 7.7-left, our method generalizes well to this unseen object. The overall performance rate is slightly reduced since the network could not learn the subtle differences between the unseen object and the others. Most of the miss-classifications are with the ape, whose shape looks similar to the duck's under some viewpoints, as shown in Fig. 7.7-right.

7.5 Conclusion

We have shown how to train a CNN to map raw input images from different input modalities to very compact output descriptors using pair-wise and triplet-wise constraints over training data and template views. Our descriptors significantly outperform LineMOD and HOG, which are widely used for object recognition and 3D pose estimation, both in terms of accuracy and descriptor length. Our representation therefore replaces them advantageously. Tests on the capability to generalize to unseen objects also have shown promising results.

Chapter 8

Future Work

In the coming years, I plan to work on robust local image features, 3D object detection and tracking, but also on geo-localization and accurate object segmentation.

8.1 Short Term Future Work

8.1.1 Robust Local and Semi-Local Feature Points

Learning to extract straight-line segments

As already discussing before, feature points are the most commonly used image features in many problems including 3D registration, probably because they can be extracted more reliably than other features and they correspond to a clear 3D location. However many scenes including indoor scenes do not exhibit many feature points, and it is then interesting to consider other features such as straight line segments.

However it is difficult to extract straight line segments from images reliably. In fact, early registration and object detection works [139] relied on segments, which were then abandon in favor of feature points probably because of this reason. For example, the corner between two white walls can be extremely difficult to detect.

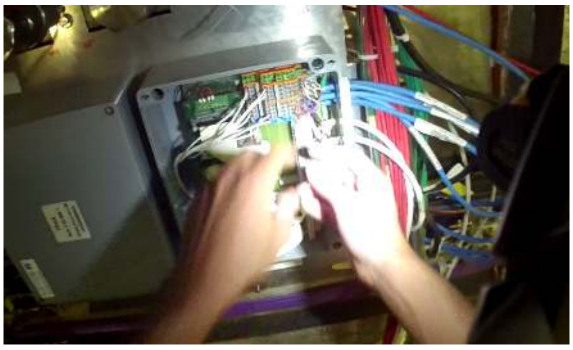
This problem can actually be set in the same formalism as in Chapter 5 and [199], where a feature map, with local maxima corresponding to the desired features, is predicted However a straightforward application would not be exploit the geometric nature of the segments. We will introduce an additional filtering step on the output of the regressors to remove the structures that are not straight. This will help to focus on only the straight-line segments over the next iterations. It will also be a good way to make statistical learning and *a priori* knowledge on the problem work together.

Learning to detect semi-local parts

Feature points and straight-line segments are the favorite features for estimating the 3D pose of the camera or an object. However they convey relatively little information on the pose, and it is necessary to combine the information from several of these features. This problem is exacerbated by the fact that some features can be incorrectly recognized, and a number of them, much larger than the theoretical minimal number, is required to compensate for these errors.

Consider the box of Fig. 8.1(a), from an actual Augmented Reality scenario in the ATLAS particle detector. In practice the inside and the surroundings of the box are not known in advance, only the geometry and appearance of the box are. When several of the corners are occluded, feature point- and straight-line segments-based detection methods fail. However, from a single corner, humans can have a very good idea of the orientation and location of the entire box.

To emulate the ability of humans to detect and estimate the pose of objects under extreme occlusions, we will develop what we refer to as "semi-local features". The key idea is to consider parts from which an estimate of the pose can be inferred. We already worked on a related idea in [94]. However this method relied on feature points


(a)

(b)

Figure 8.1: Some of the challenges we will consider in future work. (a) Detecting texture-less objects in real conditions like the box in the middle of the image and estimating its spatial orientation is very difficult for computers, but would enable novel Augmented Reality applications. (b) To be able to predict an object's 3D pose, current object detection approaches require extensive training for each new object: Each mug requires the capture of a specific training sequence, which is very cumbersome. We will develop practical methods that can generalize to multiple instances from categories of objects.

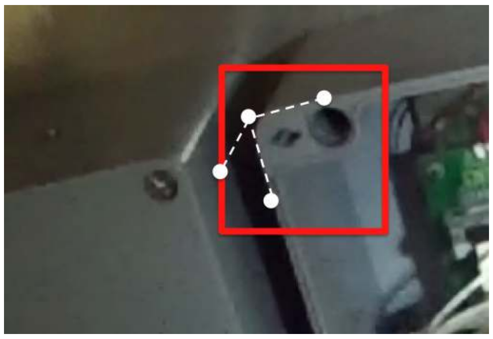


Figure 8.2: Detecting and estimating the pose of the box from Fig. 8.1(a). We propose to first detect discriminative parts, such as the visible corners, and predict the 3D orientation of each part. The part 3D orientation can be parameterised by control points, represented here with white dots.

to detect the parts, and template matching [116] to compute the 3D pose, which makes it still dependent on the presence of discriminative texture. It also only worked on planar surfaces.

Fig. 8.2 illustrates what we propose. We will train detectors to detect the different parts. For each part, we will train a regressor to predict its pose. Several options to represent this pose are possible, such as a quaternion to represent the rotation together with a 3-vector for the translation. However rotation representations have all singularities and periodicity [79], which would make the regression to their parameters difficult. Moreover it is not clear with these representations how to combine the poses predicted for different parts.

We therefore propose to instead predict the locations of "controls points". We already used such control points, which are 3D virtual points, to represent a 3D pose in [132]. By predicting their projections, we will avoid the problems mentioned above. Moreover, the predicted values will all be of the same nature, which will also facilitate training. It will also make trivial the combinations of the predictions from different parts. We will explore two directions: We can directly predict the 3D locations of the control points and align the control points with their predicted locations to compute the final pose; we can also predict their projections, and use a PnP algorithm [132, 120].

To obtain the parts and the corresponding training data, we will rely on training, calibrated video sequences. In a first stage, we will select the parts manually. We will capture images of the parts and the corresponding 3D poses using a Structure-from-Motion software such as [57]. In a second stage, we will develop a method to select these parts automatically. A possible direction for this is to apply our feature point detection algorithm on downscaled images to identify stable locations.

To demonstrate our approach, we will integrate our feature point detection method and our previous work on feature descriptor learning [225]. We will train the latter on our image database built for to obtain a local descriptor robust to complex light changes and perspective deformations.

8.1.2 3D object Detection Revisited

We will follow two different approaches to 3D object detection, both directions having their own advantages.

The first approach is in the framework of 2D view specific templates [101]. It will be adapted to relatively compact objects, and will scale to a large number of objects. The second approach will be based on the semi-local features developed in Work Package #1, and will be adapted to scenarios where large occlusions may occur, and large and elongated objects, such machines in a factory. On the long term, we will explore how these two approaches can be merged to get the benefits of both. For example, the semi-local features can be seen and rigid templates are not so different, and can be seen as two extremities of a spectrum.

We discuss these two approaches in more details below.

Template-based detection

We already developed a template-based approach to 3D object detection in [98]. In this work, we introduced a novel representation for the templates adapted to both color and depth cameras, which makes possible fast parsing of input images to find the objects. This approach gives very encouraging results. However, it can handle only a limited number of objects, while applications may have to deal with a very large number of objects.

Scalablity is actually a major issue in 3D object detection. As discussed in the Related Work section, many methods rely on one classifier per object, or multi-class classifiers such as Random Forests, whose complexity grows with the number of objects. So far the only recognition approaches that have been demonstrated to work on large scale problems are based on Nearest Neighbor (NN) classification [158, 113, 50], because extremely efficient methods for NN search exist with an average complexity of $O(1)$ [161, 155]. Moreover, a Nearest Neighbor (NN) approach would also offer the possibility to trivially add new objects, or remove old ones. which is not directly possible with neural networks, for example. However, to the best of our knowledge, such approach has not been applied to the 3D pose estimation problem, while it can potentially scale to many objects seen under large ranges of poses.

For NN approaches to perform well, a compact and discriminative description vector is required. Our representation from [46, 98] is not sufficient. For this project, we will:

- learn a suitable description of the templates;
- develop an efficient method to parse the input images based on this description and nearest-neighbor search.

Learning a template description is motivated by the success of recent work on feature point descriptor learning, including ours [30, 223, 198, 147], which shows that it is possible to learn compact descriptors that significantly outperform handcrafted methods such as SIFT or SURF. However, the problem we tackle here is more complex: While feature point descriptors are used only to find the points' identities, we here want to find both the object's identity and its pose.

In preliminary work, we use a Convolutional Neural Network (CNN) to learn a descriptor with the two following properties: a) The Euclidean distance between descriptors from two different objects should be large; b) The Euclidean distance between descriptors from the same object should be representative of the similarity between their poses. We currently do this by minimizing the following objective function over the parameters w of the CNN:

$$\mathcal{L} = \sum_{(s_i,s_j,s_k)\in\mathcal{T}} \max\left(0, 1 - \frac{||f_w(s_i) - f_w(s_k)||_2}{||f_w(s_i) - f_w(s_j)||_2 + m}\right) + \sum_{(s_i,s_j)\in\mathcal{P}} ||f_w(s_i) - f_w(s_j)||_2^2 + \lambda||w'||_2^2 \ , \qquad (8.1)$$

where $f_w(s)$ denotes the descriptor computed for sample s by the CNN with parameters w. The first term is a sum over triplets of training samples, and the second one a sum over pairs of training samples. To avoid going into technical details here, let us just say that together, they enforce both our a) and b) constraints. The last term is simply a regularization term over the parameters of the network.

The descriptor we obtained is more discriminative than previous handcrafted ones [46, 98]. However this first descriptor was trained on a relatively small number of objects, because the optimization problem is difficult and because training data is difficult to acquire, as we need calibrated images. We will therefore first build a larger database. We will also develop more efficient optimization algorithms to be able to handle this new database. One direction is to first relax the constraints between poses at the beginning of the optimization to focus on getting different descriptors for different objects, before injecting back the constraints on descriptors for different poses. We will also explore other possible, potential better objective functions than Eq. (8.1).

We also need to develop an efficient image parsing algorithm. This can be done by applying fast approximate nearest neighbor (ANN) search based on works such that [113] to our template descriptors. There is no exact algorithm for nearest-neighbor search for real-valued vectors faster than linear search. However our problem has a specific structure: In practice, it is acceptable if the ANN search returns a descriptor that corresponds to the correct object and a pose which is only close to the correct one: Instead of searching for the nearest-neighbor, we are only interesting in finding any vector among a set of acceptable ones. This observation can certainly be exploited to improve the performance in terms of speed and accuracy of the search.

Detecting extended objects

The previous approach is adapted to relatively compact objects. We will also develop an approach adapted to extended objects and scenarios where partial occlusions can occur, like in Fig. 8.2. For this, we will rely on the semi-local features developed for Work Package #1. Each detected feature of the target object will provide a set of control points, encoding the pose of the feature. We will then be able to estimate the object pose from these control points. However this has to be done with special care: Some detected features will be erroneous, and provide outlier measures.

Robust pose estimation can be performed using RANSAC [67] for example, except that control points from the same semi-local feature are either all inliers or all outliers together. Moreover in practice the features will be scarce as we are interested here in poorly textured objects; the number of inlier points will be limited and we want to make sure we use all of them, which is not guaranteed with RANSAC-like approaches.

We will therefore develop another approach for estimating the object pose. Since the number of features will be limited anyway, it is possible that a simple exhaustive approach considering each possible combinations of features being inlier or outlier will be sufficient. Otherwise a Branch-and-Bound method [162] seems appropriate here.

In any case, it will be important to take into account the uncertainty on the predicted control points, as the predictions are expected to be relatively inaccurate. While work on this aspect is limited, it is possible at least theoretically to estimate the uncertainty of values predicted by a Neural Network by linearizing the computations of the network [13]. Since a Convolutional Neural Network such as the one we will train to predict the locations of the control points is hardly linear, we propose to use a technique similar to the Unscented Filter [115], which was developed to avoid inaccuracies of the Extended Kalman Filter due to linearization. Our pose estimation algorithm will then be able to take into account the uncertainty of the predicted control points.

Learning to track an object in 3D

The two previous sections discussed 3D object detection, and are adapted when no prior knowledge about the object pose is available. Once the object pose is estimated however, it is more efficient to exploit temporal consistency to constrain the search for the object pose in the incoming frames of the video stream. A lot of work has been done on this tracking problem [131]. However existing approaches rely on the presence of clear features (typically feature points or edges) on the object, and the specific type of features needs to be chosen depending on the appearance of the target object.

Following the general approach of this project, we will explore the possibility to train a regressor to predict the motion of an object: Given two consecutive images of an object under motion, can we predict the relative motion of the object between these two images? Early experiments showed that it is possible to predict a 2D translation in challenging conditions, which is already interesting to constrain the full 3D motion. We will go further and aim to predict more complex 2D motions, such as an affine transformation, and actual 3D motions.

Another interesting direction here is the use of Dynamic Vision sensors. These sensors and work by producing asynchronous output in presence of motion *i.e.*he measurements are not organized in pixels and images, but as a list of events, where the order does not depend on the image locations. These sensors are very recent, and very attractive as they are extremely fast. However the atypical nature of their output still makes them challenging to exploit. An learning-based approach like ours should therefore be a good way to handle them. It is not clear how a statistical method can be applied on a list, by contrast with an image for example which is a common type of input. Maybe works on Natural Language Processing, which also consider will be adapted. We will explore this direction in the project.

8.1.3 Local Features for Geo-Localization with High-Level Data

We also would like to emulate the impressive ability of humans of localizing themselves based on 2D maps only, such as the one shown in Fig. 8.3(a). Currently, vision-based outdoor 3D localization requires registered images in outdoor 3D localization, which are cumbersome to acquire, and do not scale to large areas. Even the huge database of Google StreetView is not sampled enough to provide reliable results.

By contrast, 2D maps are widely available in practice and can be obtained for example from OpenStreetMap [1], which offers maps with a very broad coverage, and free to use under an open license. However, matching such a 2D map with a perspective view is clearly very challenging.

Our key idea is to extract and match high-level information instead of matching low-level image cues between images We will perform a semantic segmentation of the input image. This will provide an abstract representation of the image easier to match with the 2D map. In particular we should be able to identify the façades and the borders of the streets in the image. The camera pose can then be estimated as the 3D pose that aligns façades and street borders—and any other existing cue—from the 2D map with the ones extracted from the image. We will also estimate the *orientations* of the façades in the image, which would provide additional constraints, as the orientations in the image should be consistent with the 2D map and the camera pose.

Fig. 8.3 shows our preliminary work in this direction. We use an initial pose estimate from the camera's sensors for the input image to retrieve the 2D map of the surrounding buildings. We currently use a simple vanishing point method [88] based on horizontal and vertical straight line segments to first estimate the camera orientation.

The translation is found by aligning a 3D model generated from the 2D map with a segmentation of the image. We currently rely on a very simple pixel-wise segmentation of the image, obtained by applying a multi-class Support Vector Machine (SVM) [191, 40] trained on a dataset of manually segmented images, on the integral features introduced in [54].

Our first results shown in Fig. 8.3 and obtained with this very early version of our approach are very encouraging, however much more developments are required to reach a complete solution. We first need to develop a much more sophisticated segmentation method.

It can be noted that this segmentation problem can also be formalized as the keypoint detection problem in Chapter 5, but with constraints on local minima. Here we consider a multi-valued function, with for each image location the probabilities to belonging to different classes, such as façade, sky, or roof. We will use Convolutional Neural Network (CNN)s to learn to extract image features relevant to the task, and introduce spatial constraints, for example a sky region should be above a roof or a façade. This is often done with Random Fields. We also would like to experiment with Auto-Context [228] as well to learn such constraints.

However this is still a relatively classic and general semantic segmentation problem. For our registration problem, it is important to go further to avoid the failures of our preliminary implementation that are shown Fig. 8.4. further. For example, while usually regions of the same class are assigned the same label, it will be important to isolate each façade individually, as this will provide more constraints when matching with the 2D map. For that, we will also explicitly learn to extract the edges between façades. As mentioned above, it will also be important to estimate the orientations of the façades. This can probably be done by training a regressor to predict the orientation, based on training images.

Outdoor 3D tracking

Simultaneous Localization And Mapping (SLAM) methods are now able to reconstruct over time in 3D the environment from a handheld device such as a smartphone. It is based on feature points and straight line segments,

[1] `http://www.openstreetmap.org`

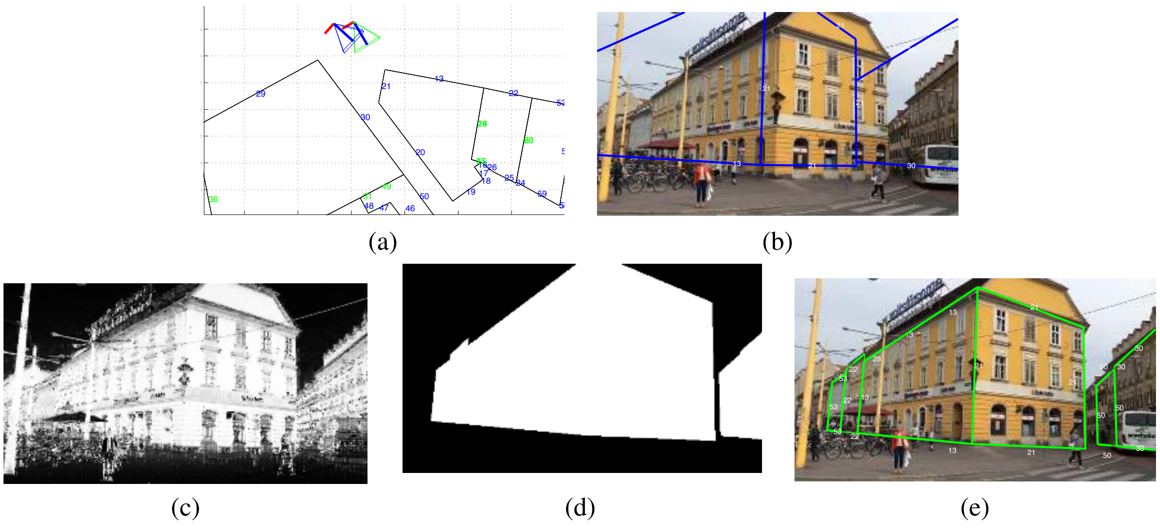


Figure 8.3: Localization with high-level data. (a) The 2D map used for 3D registration. (b) Reprojection of the map in the input image, given the pose estimate from the device's sensor. The reprojection is far from the expected location. (c) Heatmap for the Façade segmentation in the input image. (d) Reprojection of the map from a pose close to the correct one. (e) Correct pose, found by aligning the map reprojection (d) with the segmentation (c).

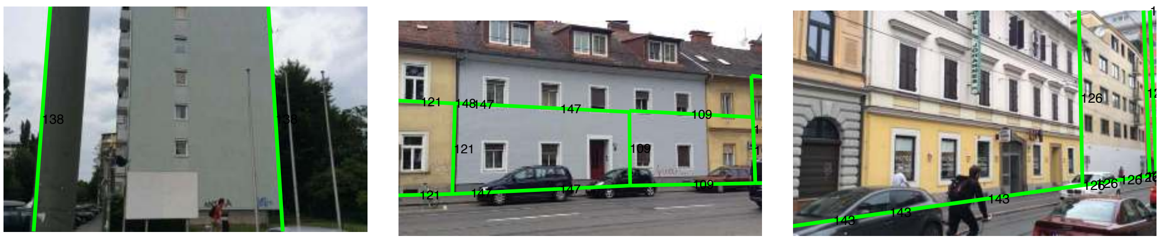


Figure 8.4: Failures by our current approaches. To solve these failures, we need to be able to extract more complex information from the input images, such as the edges of the buildings and the orientations of the façades.

which are matched over frames to estimate their 3D locations. However they are still fragile: The initialization is still challenging for a non-expert; it is easily disturbed by moving objects that violate the assumption of a static environment.

The semantic segmentation techniques we will develop for the previous part ("Using untextured maps") will also be useful here: For example, in urban environments, we can segment the façades, predict the orientations, and also their *depths*: This will be useful to better initialize the 3D locations of the image features lying on the façades, and make the initialization of the SLAM system more robust. We can also identify and segment the pedestrians, and other moving objects, to prune the image features lying on them, as they cannot be reconstructed by the API. Again, this will improve the API's robustness.

8.1.4 Accurate 3D Object Segmentation

Learning to segment known objects

In Augmented Reality, the parts of control objects hidden by real ones should not be rendered, for a consistent augmentation. To solve this, the silhouette of the occluding real objects present in the image should be estimated precisely.

An important case of occluding objects is the user's hands, which are likely to be between his or her point of view and the control objects. Simple cues such as the skin color have been used to segment hands, however it is neither reliably nor accurate. Even with depth cameras, the problem remains challenging as depth maps are typically inaccurate along occluding contours.

We will therefore focus first on hand segmentation in images captured by the Head-Mounted Display of the user, in Augmented Reality scenarios. This is related to the semantic segmentation discussed above, but with different requirements: We need here a fast—for real-time applications—but very accurate segmentation. On the other hand, we can focus on only one object to segment.

We will focus on an accurate segmentation of the contours of the hands. Regularization terms, typically used in segmentation problems, are not very useful here as they only make the segments smooth, without guarantee that the segment contours are at the right location. Our idea is to rely on an approach similar to our work developed for feature extraction, but instead of learning a function that peaks on the desired structures, we will consider a smooth function with zero-crossings at the contours.

Here the learned function should decrease with the distance to the contours outside of the object's silhouette, and increase with this distance inside the silhouette. This way, by extracting the zero-crossings of this function, we will get the contours of the hands. As for the linear structure extraction, it is expected that the localization will be much more accurate with this regression approach rather than a standard classification approach.

To speed up the prediction of the function, we will rely on a multi-scale approach: We will do a first prediction at a low scale, and use it with the image in an Auto-Context strategy to do a better prediction at a higher scale, and so on. We will process only the regions close to the predicted contours, and skip the rest of the image. This way we will speed up the segmentation process.

Note also that it is easy with Machine Learning to rely on both color and depth information provided by modern depth sensors. These two cues are complementary as occluding contours are present at least to some extent and well localized in color images, but look similar to other types of contours. In depth images, they correspond to depth discontinuities, but their localization is very poor.

Learning to identify occluding contours

After considering the extraction of occluding contours of a known object, we will go further and identify the occluding contours of the objects present in the scene. This is a very challenging topic, and there is very little work on the topic. A prominent exception is [100], which efficiently combines contour classification and novel local constraints. We can now benefit from depth cameras which provide a very useful information even if they do not provide reliable information precisely along occluding contours.

We propose to see this problem again as a regression problem. We will train a regressor to predict the actual depth of a pixel, given the image information and the depth measures provided by a depth sensor for its surrounding. By applying this regressor to all the pixels, we will obtain an improved depth map, with the occluding contours at their correct image locations. Again, we will perform Auto-Context-like iterations which will improve the initial depth estimates.

We therefore need color images and their—noisy—depth maps as captured by a depth sensor, and the corresponding actual depth maps. It is very difficult to obtain such perfect depth maps. To obtain at least a good approximation, we will apply an algorithm developed at ICG which already improves the depth map of a depth sensor by exploiting the corresponding color images [66]. We will feed this algorithm with manually drawn occluding contours, which will guarantee the quality of the output depth map.

8.2 Longer Term Future Work

8.2.1 Learning to Detect Objects from a 3D Model Only

As mentioned in Section 8.1.3, relying on calibrated images for 3D pose estimation as it is currently done is cumbersome, and in Work Package #3, we will develop methods for using only 2D maps, just as humans do. We would also like to do the same for 3D objects: For example, technicians can understand easily the orientation of an object from a 3D model or even a schematic representation only.

In fact, early Computer Vision systems were based only on a 3D model for estimating the 3D pose of an object. However they were assuming that 3D edges corresponded to 2D edges in images. The reality is much more complex: Some edges may not be visible in the images, or their appearance can vary tremendously.

The approach we propose to solve this long-pending problem extend our plans for Work Package #1: Can we learn to detect features not only in images but also on 3D models, such that these 3D features correspond to 2D features that can be detected reliably in images? One direction is to train an extension of the feature point detection developed for Work Package #1 on CAD models, and real images of instances of these CAD models.

8.2.2 3D Pose Estimation for Object Categories

The method we introduced in Chapter 7 for 3D object detection and pose estimation requires to capture additional images each time we want to learn a new object. For practical purposes, it can be important to be able to handle any object from a given category, for example any of the mugs of Fig. 8.1(b), without having to capture them beforehand.

To do so, we will collect images of different objects from the same category. In general, a canonical reference pose can be defined across the instances of the same category. For example, for mugs, it can be anchored to the mug's handle. Then, one direction is to extend the method we will develop for Section 8.1.2 to compute the same descriptor for different instances under the same pose. This can probably be done by adding to set $\mathcal{P}$ in Eq. (8.1) pairs of views (s_i, s_j) from different objects and similar poses, however this has to be evaluated.

8.2.3 Closing the Loop between Computer Vision and Robotics

Almost all the algorithms we propose to develop for the beginning of this project work are designed for still images. In the future, we will extend them to streams of images, which will provide much richer information. This can be done passively, in the sense that the algorithms have no control on the camera motion. In the case of Robotics applications, it can also be done actively: algorithms will control the camera and move it to obtain views that can disambiguate poses or object identities, for example. Many methods have been proposed in the past for this Active Vision problematic, and it will be very interesting to revisit them in the light of modern techniques.

Our general idea is to apply the proposed algorithms to detect and track the objects relevant to the application, and also the mobile parts of the robot, such as its hand, which is very challenging for classical techniques because of the lack of texture. Knowing control commands can help to improve the tracking, but tracking will also allow a feedback to the control commands and make them more precise. We developed a very interesting perspective in this direction [163], where we train a feedback loop to track the 3D pose of a hand, by learning to predict corrections to apply to the current pose estimate. While our recent work [163] is purely about computer vision, learning to predict corrections should be very interesting for robotics using computer vision.

Bibliography

[1] M. Agrawal, K. Konolige, and M.R. Blas. Censure: Center Surround Extremas for Realtime Feature Detection and Matching. In *European Conference on Computer Vision*, 2008.

[2] T. Ahonen, A. Hadid, and M. Pietikïinen. Face Description with Local Binary Patterns: Application to Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):2037–2041, 2006.

[3] A. Alahi, L. Jacques, Y. Boursier, and P. Vandergheynst. Sparsity Driven People Localization with a Heterogeneous Network of Cameras. *Journal of Mathematical Imaging and Vision*, 2011.

[4] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast Retina Keypoint. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[5] P. F. Alcantarilla, J. Nuevo, and A. Bartoli. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In *British Machine Vision Conference*, 2013.

[6] K. Ali, F. Fleuret, D. Hasler, and P. Fua. A Real-Time Deformable Detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):225–239, February 2012.

[7] Y. Amit. *2D Object Detection and Recognition: Models, Algorithms, and Networks.* The MIT Press, 2002.

[8] Y. Amit and D. Geman. Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 9(7):1545–1588, 1997.

[9] Y. Amit, D. Geman, and X. Fan. A Coarse-To-Fine Strategy for Multi-Class Shape Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[10] A. Andoni and P. Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM*, 51(1), 2008.

[11] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour Detection and Hierarchical Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2011.

[12] ARM. Realview Compilation Tools, 2010.

[13] K.O. Arras. An Introduction to Error Propagation: Derivation, Meaning and Examples of Equation $C_Y = F_X C_X F_x^t$. Technical report, EPFL, 1998.

[14] A. B. Ashraf, S. Lucey, and T. Chen. Reinterpreting the Application of Gabor Filters as a Manipulation of the Margin in Linear Support Vector Machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1335–1341, 2010.

[15] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3D Chairs: Exemplar Part-Based 2D-3D Alignement Using a Large Dataset of CAD Models. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[16] A. Bakry and A. Elgammal. Untangling Object-View Manifold for Multiview Recognition and Pose Estimation. In *European Conference on Computer Vision*, 2014.

[17] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding*, 10(3):346–359, 2008.

[18] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *European Conference on Computer Vision*, 2006.

[19] J. Beis and D.G. Lowe. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *Conference on Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.

[20] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-farley, and Y. Bengio. Theano: A CPU and GPU Math Expression Compiler. In *Python for Scientific Computing Conference*, June 2010.

[21] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[22] G. Blake, R.G. Dreslinski, and T. Mudge. A Survey of Multicore Processors. *IEEE Signal Processing Magazine*, 2009.

[23] A. Böcker, S. Derksen, E. Schmidt, A. Teckentrup, and G. Schneider. A Hierarchical Clustering Approach for Large Compound Libraries. *Journal of Chemical Information and Modeling*, 45(4):807–815, 2005.

[24] G. Borgefors. Hierarchical Chamfer Matching: A Parametric Edge Matching Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1988.

[25] A. Bosch, A. Zisserman, and X. Munoz. Image Classification Using Random Forests and Ferns. In *International Conference on Computer Vision*, pages 1–8, 2007.

[26] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6D Object Pose Estimation Using 3D Object Coordinates. In *European Conference on Computer Vision*, 2014.

[27] G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly Media, 2008.

[28] B. Brahnam and L. Nanni. High Performance Set of Features for Human Action Classification. In *International Conference on Image Processing*, 2009.

[29] L. Breiman. Hinging Hyperplanes for Regression, Classification, and Function Approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, 1993.

[30] M. Brown, G. Hua, and S. Winder. Discriminative Learning of Local Image Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[31] H. Cai, T. Hongping, and J. Matas. Fast Detection of Multiple Textureless 3D Objects. In *Computer Vision Systems*, 2013.

[32] M. Calonder, V. Lepetit, and P. Fua. Keypoint Signatures for Fast Learning and Recognition. In *European Conference on Computer Vision*, October 2008.

[33] M. Calonder, V. Lepetit, K. Konolige, J. Bowman, P. Mihelich, and P. Fua. Compact Signatures for High-Speed Interest Point Description and Matching. In *International Conference on Computer Vision*, September 2009.

[34] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, 2012.

[35] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary Robust Independent Elementary Features. In *European Conference on Computer Vision*, September 2010.

[36] J. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), 1986.

[37] Z. Cao, Q. Yin, X. Tang, and J. Sun. Face Recognition with Learning-Based Descriptor. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[38] Z. Cao, Q. Yin, X. Tang, and J. Sun. Face Alignment by Explicit Shape Regression. *International Journal of Computer Vision*, 2014.

[39] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod. CHoG: Compressed Histogram of Gradients a Low Bit-Rate Feature Descriptor. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[40] C.-C. Chang and C.-J. Lin. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):271–2727, 2011.

[41] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng. Boosted Multi-Task Learning. *Machine Learning*, 2010.

[42] S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *Conference on Computer Vision and Pattern Recognition*, pages 539–546, 2005.

[43] C. Chow and C. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[44] O. Chum and J. Matas. Matching with Prosac - Progressive Sample Consensus. In *Conference on Computer Vision and Pattern Recognition*, pages 220–226, June 2005.

[45] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[46] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[47] D. Damen, P. Bunnun, A. Calway, and W. Mayol-cuevas. Real-Time Learning and Detection of 3D Texture-Less Objects: A Scalable Approach. In *British Machine Vision Conference*, 2012.

[48] A. J. Davison, I. Reid, N. Molton, and O. Stasse. Monoslam: Real-Time Single Camera Slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, June 2007.

[49] A.J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *International Conference on Computer Vision*, 2003.

[50] T. Dean, J. Yagnik, M. Ruzon, M. Segal, J. Shlens, and S. Vijayanarasimhan. Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *Conference on Computer Vision and Pattern Recognition*, 2013.

[51] S. Divvala, A. Efros, and M. Hebert. How Important Are 'deformable Parts' in the Deformable Parts Model? In *European Conference on Computer Vision*, 2012.

[52] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral Channel Features. In *British Machine Vision Conference*, pages 1–11, 2009.

[53] P. Dollár, P. Welinder, and P. Perona. Cascaded Pose Regression. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[54] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian Detection: A Benchmark. In *Conference on Computer Vision and Pattern Recognition*, June 2009.

[55] P. Dollár and C. L. Zitnick. Fast Edge Detection Using Structured Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[56] P. Domingos, M. Pazzani, and G. Provan. On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss. In *Machine Learning*, pages 103–130, 1997.

[57] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *European Conference on Computer Vision*, 2014.

[58] M. Enzweiler, A. Eigenstetter, B. Schiele, and D. M. Gavrila. Multi-Cue Pedestrian Classification with Partial Occlusion Handling. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[59] A. Ess, B. Leibe, and L. Van Gool. Depth and Appearance for Mobile Scene Analysis. In *International Conference on Computer Vision*, 2007.

[60] M. Everingham, C.K.I. Williams L. Van Gool and, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge (VOC2010) Results, 2010.

[61] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[62] O.D. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.

[63] L. Fei-fei, R. Fergus, and P. Perona. One-Shot Learning of Object Categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[64] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[65] V. Ferrari, F. Jurie, and C. Schmid. From Images to Shape Models for Object Detection. *International Journal of Computer Vision*, 2009.

[66] D. Ferstl, C. Reinbacher, R. Ranftl, M. Rüther, and H. Bischof. Image Guided Depth Upsampling Using Anisotropic Total Generalized Variation. In *International Conference on Computer Vision*, 2013.

[67] M.A Fischler and R.C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications ACM*, 24(6):381–395, 1981.

[68] W. Förstner, T. Dickscheid, and F. Schindler. Detecting Interpretable and Accurate Scale-Invariant Keypoints. In *International Conference on Computer Vision*, September 2009.

[69] W. Förstner and E. Gülch. A Fast Operator for Detection and Precise Location of Distinct Points, Corners and Centres of Circular Features. In *ISPRS Intercommission Conference on Fast Processing of Photogrammetric Data*, 1987.

[70] Y. Freund and R.E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[71] J. H. Friedman and U.a Fayyad. On Bias, Variance, 0/1-Loss, and the Curse-Of-Dimensionality. *Data Mining and Knowledge Discovery*, 1:55–77, 1997.

[72] B. Froba and A. Ernst. Face Detection with the Modified Census Transform. In *International Conference on Automatic Face and Gesture Recognition*, pages 91–96, 17-19 2004.

[73] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng. Complete Solution Classification for the Perspective-Three-Point Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):930–943, 2003.

[74] D. Gavrila and V. Philomin. Real-Time Object Detection for “smart“ Vehicles. In *International Conference on Computer Vision*, pages 87–93, 1999.

[75] D. M. Gavrila and S. Munder. Multi-Cue Pedestrian Detection and Tracking from a Moving Vehicle. *International Journal of Computer Vision*, 2007.

[76] M. Godec, P. Roth, and H. Bischof. Hough-Based Tracking of Non-Rigid Objects. In *International Conference on Computer Vision*, 2011.

[77] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[78] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[79] F.S. Grassia. Practical Parameterization of Rotations Using the Exponential Map. *Journal of graphics tools*, 3(3):29–48, 1998.

[80] K. Grauman and T. Darrell. The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *International Conference on Computer Vision*, pages 1458–1465, 2005.

[81] C. Gu and X. Ren. Discriminative Mixture-Of-Templates for Viewpoint Classification. In *European Conference on Computer Vision*, 2010.

[82] W. Guan and S. You. Robust Image Matching with Line Context. In *British Machine Vision Conference*, 2013.

[83] R. Gupta and A. Mittal. SMD: A Locally Stable Monotonic Change Invariant Feature Descriptor. In *European Conference on Computer Vision*, 2008.

[84] R. Gupta, H. Patil, and A. Mittal. Robust Order-Based Methods for Feature Description. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[85] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning Rich Features from RGB-D Images for Object Detection and Segmentation. In *European Conference on Computer Vision*, 2014.

[86] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *Conference on Computer Vision and Pattern Recognition*, 2006.

[87] C.G. Harris and M.J. Stephens. A Combined Corner and Edge Detector. In *Fourth Alvey Vision Conference*, 1988.

[88] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[89] W. Hartmann, M. Havlena, and K. Schindler. Predicting Matchability. In *Conference on Computer Vision and Pattern Recognition*, June 2014.

[90] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[91] D.C. Hauagge and N. Snavely. Image Matching Using Local Symmetry Features. In *Conference on Computer Vision and Pattern Recognition*, June 2012.

[92] M. Heikkila, M. Pietikainen, and C. Schmid. Description of Interest Regions with Local Binary Patterns. *Pattern Recognition*, 42(3):425–436, March 2009.

[93] J. Heinly, E. Dunn, and J.-M. Frahm. Comparative Evaluation of Binary Features. In *European Conference on Computer Vision*, 2012.

[94] S. Hinterstoisser, S. Benhimane, N. Navab, P. Fua, and V. Lepetit. Online Learning of Patch Perspective Rectification for Efficient Object Detection. In *Conference on Computer Vision and Pattern Recognition*, 2008.

[95] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient Response Maps for Real-Time Detection of Textureless Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34, May 2012.

[96] S. Hinterstoisser, S. Ilic, N. Navab, P. Fua, and V. Lepetit. Dominant Orientation Templates for Real-Time Detection of Texture-Less Objects. In *Conference on Computer Vision and Pattern Recognition*, June 2010.

[97] S. Hinterstoisser, S. Ilic, N. Navab, P. Fua, and V. Lepetit. Learning Real-Time Perspective Patch Rectification. *International Journal of Computer Vision*, 91(1):107–130, 2011.

[98] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In *Asian Conference on Computer Vision*, 2012.

[99] G.E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14:1771–1800, 2002.

[100] D. Hoiem, A.A. Efros, and M. Hebert. Recovering Occlusion Boundaries from an Image. *International Journal of Computer Vision*, 91(3), 2011.

[101] D. Hoiem and S. Savarese. *Representations and Techniques for 3D Object Recognition and Scene Interpretation*. Morgan Claypool Publishers, 2011.

[102] D. Hoiem, R. Sukthankar, H. Schneiderman, and L. Huston. Object-Based Image Retrieval Using the Statistical Structure of Images. *Journal of Machine Learning Research*, 02:490–497, 2004.

[103] E. Hsiao and M. Hebert. Occlusion Reasoning for Object Detection Under Arbitrary Viewpoint. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(9):1803–1815, 2014.

[104] G. Hua, M. Brown, and S. Winder. Discriminant Embedding for Local Image Descriptors. In *International Conference on Computer Vision*, 2007.

[105] C. Huang, H. Ai, Y. Li, and S. Lao. Vector Boosting for Rotation Invariant Multi-View Face Detection. In *Conference on Computer Vision and Pattern Recognition*, 2005.

[106] D.H. Hubel. *Eye, Brain, and Vision*. W. H. Freeman, 1995.

[107] D.P. Huttenlocher, G.A. Klanderman, and W. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 850–863, 1993.

[108] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Symposium on Theory of Computing*, pages 604–613, 1998.

[109] intel. sse4 programming reference: software.intel.com/file/18187., april 2007.

[110] N. Jacobs, N. Roman, and R. Pless. Consistent Temporal Variations in Many Outdoor Scenes. In *Conference on Computer Vision and Pattern Recognition*, 2007.

[111] P. Jain, B. Kulis, J. Davis, and I. Dhillon. Metric and Kernel Learning Using a Linear Transformation. *Journal of Machine Learning Research*, 2012.

[112] H. Jégou, M. Douze, and C. Schmid. Improving Bag-Of-Features for Large Scale Image Search. *International Journal of Computer Vision*, 87(3):316–336, 2010.

[113] H. Jégou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[114] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating Local Descriptors into a Compact Image Representation. In *Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, June 2010.

[115] S.J. Julier and J.K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *IEEE*, 92(3):401–422, 2004.

[116] F. Jurie and M. Dhome. Hyperplane Approximation for Template Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):996–100, July 2002.

[117] Z. Kalal, J. Matas, and K. Mikolajczyk. P-N Learning: Bootstrapping Binary Classifiers from Unlabeled Data by Structural Constraints. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[118] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-Learning-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(07):1409–1422, July 2012.

[119] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

[120] L. Kneip, H. Li, and Y. Seo. UPnP: An Optimal O(n) Solution to the Absolute Pose Problem with Universal Applicability. In *European Conference on Computer Vision*, 2014.

[121] J. Koenderink. The Structure of Images. *Biological Cybernetics*, 50(5):363–370, August 1984.

[122] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1106–1114, 2012.

[123] E. Krupka, A. Vincentnikov, B. Klein, A. Bar Hillel, and D. Freedman. Discriminative Ferns Ensemble for Hand Pose Recognition. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[124] B. Kulis and T. Darrell. Learning to Hash with Binary Reconstructive Embeddings. In *Advances in Neural Information Processing Systems*, pages 1042–1050, 2009.

[125] M. Bartosz Kursa. Robustness of Random Forest-Based Gene Selection Methods. *BMC Bioinformatics*, 2014.

[126] P.-Y. Laffont, Z. Ren, X. Tao, C. Qian, and J. Hays. Transient Attributes for High-Level Understanding and Editing of Outdoor Scenes. *ACM Transactions on Graphics*, 33(4):149, 2014.

[127] K. Lai, L. Bo, X. Ren, and D. Fox. A Scalable Tree-Based Approach for Joint Object and Pose Recognition. In *AAAI*, 2011.

[128] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[129] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[130] V. Lepetit and P. Fua. Keypoint Recognition Using Randomized Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, September 2006.

[131] V. Lepetit, P. Lagger, and P. Fua. Randomized Trees for Real-Time Keypoint Recognition. In *Conference on Computer Vision and Pattern Recognition*, June 2005.

[132] V. Lepetit, F. Moreno-noguer, and P. Fua. EPnP: An Accurate $o(n)$ Solution to the PnP Problem. *International Journal of Computer Vision*, 2009.

[133] V. Lepetit, J. Pilet, and P. Fua. Point Matching as a Classification Problem for Fast and Robust Object Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*, June 2004.

[134] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints. In *International Conference on Computer Vision*, 2011.

[135] J. Liebelt and C. Schmid. Multi-View Object Class Detection with a 3D Geometric Model. In *Conference on Computer Vision and Pattern Recognition*, pages 1688–1695, 2010.

[136] C. J. Lin, R. C. Weng, and S. S. Keerthi. Trust Region Newton Method for Logistic Regression. *Journal of Machine Learning Research*, 9:627–650, 2008.

[137] T. Lindeberg. Scale-Space for Discrete Signals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):234–254, 1990.

[138] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised Hashing with Kernels. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[139] D. G. Lowe. Fitting Parameterized Three-Dimensional Models to Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, June 1991.

[140] D.G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 20(2), 2004.

[141] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *International Conference on Very Large Data Bases*, 2007.

[142] P. Mainali, G. Lafruit, K. Tack, L. Van Gool, and R. Lauwereins. Derivative-Based Scale Invariant Image Feature Detector with Error Resilience. *IEEE Transactions on Image Processing*, 23(5):2380–2391, 2014.

[143] P. Mainali, G. Lafruit, Q. Yang, B. Geelen, L. Van Gool, and R. Lauwereins. SIFER: Scale-Invariant Feature Detector with Error Resilience. *International Journal of Computer Vision*, 104(2):172–197, 2013.

[144] M. Malekesmaeili, R. Ward, and M. Fatourechi. A Fast Approximate Nearest Neighbor Search Algorithm in the Hamming Space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[145] T. Malisiewicz, A. Gupta, and A. Efros. Ensemble of Exemplar-SVMs for Object Detection and Beyond. In *International Conference on Computer Vision*, 2011.

[146] S. Marcel, Y. Rodriguez, and G. Heusch. On the Recent Use of Local Binary Patterns for Face Authentication. *International Journal on Image and Video Processing Special Issue on Facial Image Processing*, pages 469–481, 2007.

[147] J. Masci, D. Migliore, M. M. Bronstein, and J. Schmidhuber. *Registration and Recognition in Images and Videos*, chapter Descriptor Learning for Omnidirectional Image Matching. Springer, 2014.

[148] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. In *British Machine Vision Conference*, pages 384–393, September 2002.

[149] K. Mikolajczyk and C. Schmid. A Performance Evaluation of Local Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2004.

[150] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A Comparison of Affine Region Detectors. *International Journal of Computer Vision*, 65(1/2):43–72, 2005.

[151] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *American Association for Artificial Intelligence Conference*, 2002.

[152] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping That Provably Converges. In *International Joint Conference on Artificial Intelligence*, 2003.

[153] H. Moravec. Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. In *tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University, Stanford University*, September 1980.

[154] P. Moreels and P. Perona. Evaluation of Features Detectors and Descriptors Base on 3D Objects. In *International Journal of Computer Vision*, 2006.

[155] M. Muja and D. G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[156] L. Nanni and A. Lumini. Local Binary Patterns for a Hybrid Fingerprint Matcher. *Pattern Recognition*, 41(11):3461–3466, 2008.

[157] S. K. Nayar, S. A. Nene, and H. Murase. Real-Time 100 Object Recognition System. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(12):1186–1198, 1996.

[158] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *Conference on Computer Vision and Pattern Recognition*, 2006.

[159] M. Norouzi and D. Fleet. Minimal Loss Hashing for Compact Binary Codes. In *International Conference on Machine Learning*, 2011.

[160] M. Norouzi, A. Punjani, and D.J. Fleet. Fast Search in Hamming Space with Multi-Index Hashing. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[161] M. Norouzi, A. Punjanu, and D.J. Fleet. Fast Exact Search in Hamming Space with Multi-Index Hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1107–1119, 2014.

[162] S. Nowozin and C.H. Lampert. Structured Learning and Prediction in Computer Vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[163] M. Oberweger, P. Wohlhart, and V. Lepetit. Training a Feedback Loop for Hand Pose Estimation. In *International Conference on Computer Vision*, 2015.

[164] T. Ojala, M. Pietikäinen, and D. Harwood. A Comparative Study of Texture Measures with Classification Based on Feature Distributions. *Journal of Machine Learning Research*, 29:51–59, 1996.

[165] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

[166] C. F. Olson and D. P. Huttenlocher. Automatic Target Recognition by Matching Oriented Edge Pixels. *Journal of Machine Learning Research*, 6:103–113, January 1997.

[167] O. Oshin, A. Gilbert, J. Illingworth, and R. Bowden. Action Recognition Using Randomised Ferns. In *International Conference on Computer Vision*, 2009.

[168] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition Using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448–461, 2010.

[169] M. Ozuysal, P. Fua, and V. Lepetit. Fast Keypoint Recognition in Ten Lines of Code. In *Conference on Computer Vision and Pattern Recognition*, June 2007.

[170] M. Ozuysal, V. Lepetit, and P. Fua. Pose Estimation for Category Specific Multiview Object Localization. In *Conference on Computer Vision and Pattern Recognition*, June 2009.

[171] D. Park and D. Ramanan. N-Best Maximal Decoders for Part Mmdels. In *International Conference on Computer Vision*, 2011.

[172] N. Payet and S. Todorovic. From Contours to 3D Object Detection and Pose Estimation. In *International Conference on Computer Vision*, 2011.

[173] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D Geometry to Deformable Part Models. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[174] R. Pless and R. Souvenir. A Survey of Manifold Learning for Images. *IPSJ Transactions on Computer Vision and Applications*, 1:83–94, 2009.

[175] A. Richardson and E. Olson. Learning Convolutional Filters for Interest Point Detection. In *International Conference on Robotics and Automation*, pages 631–637, May 2013.

[176] R. Rios-cabrera and T. Tuytelaars. Boosting Masked Dominant Orientation Templates for Efficient Object Detection. *Computer Vision and Image Understanding*, 120:103–116, 2014.

[177] J.J. Rodrigues, J.-S. Kim, M. Furukawa, J. Xavier, P. Aguiar, and T. Kanade. 6D Pose Estimation of Textureless Shiny Objects Using Random Ferns for Bin-Picking. In *International Conference on Intelligent Robots and Systems*, 2012.

[178] A. Rodriguez, V. N. Boddeti, B. V. Kumar, and A. Mahalanobis. Maximum Margin Correlation Filter: A New Approach for Localization and Classification. *IEEE Transactions on Image Processing*, 22(2):631–643, 2013.

[179] S. Rosset, J. Zhu, and T. Hastie. Boosting as a Regularized Path to a Maximum Margin Classifier. *Journal of Machine Learning Research*, 2004.

[180] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. In *European Conference on Computer Vision*, 2006.

[181] E. Rosten, R. Porter, and T. Drummond. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:105–119, 2010.

[182] E. Rublee, V. Rabaud, K. Konolidge, and G. Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, 2011.

[183] W.J. Rucklidge. Efficiently Locating Objects Using the Hausdorff Distance. *International Journal of Computer Vision*, 1997.

[184] R. Salakhutdinov and G. Hinton. Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure. In *International Conference on Artificial Intelligence and Statistics*, 2007.

[185] R. Salakhutdinov and G. Hinton. Semantic Hashing. *International Journal of Approximate Reasoning*, 50(7), 2009.

[186] S. Salti, A. Lanza, and L. Di Stefano. Keypoints from Symmetries by Wave Propagation. In *Conference on Computer Vision and Pattern Recognition*, June 2013.

[187] S. Savarese and L. Fei-fei. 3D Generic Object Categorization, Localization and Pose Estimation. In *International Conference on Computer Vision*, 2007.

[188] R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence Rated Predictions. *Machine Learning*, 37(3):297–336, 1999.

[189] C. Scherrer, J. Pilet, P. Fua, and V. Lepetit. The Haunted Book. In *International Symposium on Mixed and Augmented Reality*, October 2008.

[190] C. Schmid and R. Mohr. Local Grayvalue Invariants for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534, May 1997.

[191] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[192] G. Shakhnarovich. *Learning Task-Specific Similarity*. PhD thesis, Massachusetts Institute of Technology, 2005.

[193] G. Shakhnarovich. *Learning Task-Specific Similarity*. PhD thesis, MIT, 2006.

[194] G. Shakhnarovich, P. Viola, and T. Darrell. Fast Pose Estimation with Parameter-Sensitive Hashing. In *International Conference on Computer Vision*, 2003.

[195] J. Shi and C. Tomasi. Good Features to Track. In *Conference on Computer Vision and Pattern Recognition*, June 1994.

[196] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient Human Pose Estimation from Single Depth Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2821–2840, 2013.

[197] K. Simonyan, A. Vedaldi, and A. Zisserman. Descriptor Learning Using Convex Optimisation. In *European Conference on Computer Vision*, 2012.

[198] K. Simonyan, A. Vedaldi, and A. Zisserman. Learning Local Feature Descriptors Using Convex Optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[199] A. Sironi, V. Lepetit, and P. Fua. Multiscale Centerline Detection by Learning a Scale-Space Distance Transform. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[200] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua. Learning Separable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1):94–106, 2015.

[201] J. Sivic and A. Zisserman. Video Google: Efficient Visual Search of Videos. In *Toward Category-Level Object Recognition*, pages 127–144. Springer, 2006.

[202] J. Sochman and J. Matas. Waldboost - Learning for Time Constrained Sequential Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 150–157, June 2005.

[203] J. Sochman and J. Matas. Learning a Fast Emulator of a Binary Decision Process. In *Asian Conference on Computer Vision*, pages 236–245, 2007.

[204] M. Stark, M. Goesele, and B. Schiele. Back to the Future: Learning Shape Models from 3D CAD Data. In *British Machine Vision Conference*, pages 1061–10611, 2010.

[205] C. Steger. Occlusion Clutter, and Illumination Invariant Object Recognition. In *ISPRS*, 2002.

[206] F. Stein. Efficient Computation of Optical Flow Using the Census Transform. In C. Rasmussen, H. Bülthoff, B. Schälkopf, and M. Giese, editors, *Pattern Recognition*, pages 79–86. Springer, 2004.

[207] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. LDAHash: Improved Matching with Smaller Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1), January 2012.

[208] C. Strecha, R. Fransens, and L. Van Gool. Wide-Baseline Stereo from Multiple Views: A Probabilistic Account. In *Conference on Computer Vision and Pattern Recognition*, 2004.

[209] C. Strecha, R. Fransens, and L. Van Gool. Combined Depth and Outlier Estimation in Multi-View Stereo. In *Conference on Computer Vision and Pattern Recognition*, 2006.

[210] C. Strecha, A. Lindner, K. Ali, and P. Fua. Training for Task Specific Keypoint Detection. In *DAGM Symposium on Pattern Recognition*, 2009.

[211] C. Strecha, L. Van Gool, and P. Fua. A Generative Model for True Orthorectification. In *International Society for Photogrammetry and Remote Sensing*, July 2008.

[212] H. Su, M. Sun, L. Fei-fei, and S. Savarese. Learning a Dense Multi-View Representation for Detection, Viewpoint Classification and Synthesis of Object Categories. In *International Conference on Computer Vision*, 2009.

[213] M. Sun, G. R. Bradski, B.-X. Xu, and S. Savarese. Depth-Encoded Hough Voting for Joint Object Detection and Shape Recovery. In *European Conference on Computer Vision*, 2010.

[214] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the Importance of Momentum and Initialization in Deep Learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[215] D.J. Tan and S. Ilic. Multi-Forest Tracker: A Chameleon in Tracking. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[216] D. Tang, Y. Liu, and T.-K. Kim. Fast Pedestrian Detection by Cascaded Random Forest with Dominant Orientation Templates. In *British Machine Vision Conference*, 2012.

[217] F. Tang, S.H. Lim, N.L. Chang, and H. Tao. A Novel Feature Descriptor Invariant to Complex Brightness Changes. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[218] M. Tarr and S. Pinker. Mental Rotation and Orientation-Dependence in Shape Recognition. *Cognitive Phycology*, 21(2):233–282, 1 1989.

[219] S. Taylor, E. Rosten, and T. Drummond. Robust Feature Matching in $2.3\mu S$. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[220] A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim. Latent-Class Hough Forests for 3D Object Detection and Pose Estimation. In *European Conference on Computer Vision*, 2014.

[221] E. Tola, V. Lepetit, and P. Fua. Daisy: An Efficient Dense Descriptor Applied to Wide Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830, 2010.

[222] A. Torralba, R. Fergus, and Y. Weiss. Small Codes and Large Databases for Recognition. In *Conference on Computer Vision and Pattern Recognition*, June 2008.

[223] T. Trzcinski, M. Christoudias, P. Fua, and V. Lepetit. Boosting Binary Keypoint Descriptors. In *Conference on Computer Vision and Pattern Recognition*, June 2013.

[224] T. Trzcinski, M. Christoudias, and V. Lepetit. Learning Image Descriptors with Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):597–610, 2015.

[225] T. Trzcinski, M. Christoudias, and V. Lepetit. Learning Image Descriptors with Boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):597–610, 2015.

[226] T. Trzcinski, M. Christoudias, V. Lepetit, and P. Fua. Learning Image Descriptors with the Boosting-Trick. In *Advances in Neural Information Processing Systems*, December 2012.

[227] T. Trzcinski and V. Lepetit. Efficient Discriminative Projections for Compact Binary Descriptors. In *European Conference on Computer Vision*, 2012.

[228] Z. Tu and X. Bai. Auto-Context and Its Applications to High-Level Vision Tasks and 3D Brain Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[229] T. Tuytelaars and C. Schmid. Vector Quantizing Feature Space with a Regular Lattice. In *International Conference on Computer Vision*, 2007.

[230] A. Vedaldi. `http://www.vlfeat.org/~vedaldi/code/siftpp.html`, 2005.

[231] A. Vedaldi. An Open Implementation of the SIFT Detector and Descriptor. Technical report, UCLA CSD, 2007.

[232] Y. Verdie, K. M. Yi, P. Fua, and V. Lepetit. TILDE: A Temporally Invariant Learned DEtector. In *Conference on Computer Vision and Pattern Recognition*, 2015.

[233] M. Villamizar, A. Garrell, A. Sanfeliu, and F. Moreno-noguer. Online Human-Assisted Learning Using Random Ferns. In *International Conference on Pattern Recognition*, 2012.

[234] M. Villamizar, H. Grabner, J. Andrade-Cetto, A. Sanfeliu, L. Van Gool, and F. Moreno-Noguer. Efficient 3D Object Detection Using Multiple Pose-Specific Classifiers. In *British Machine Vision Conference*, 2011.

[235] M. Viola, M. Jones, and P. Viola. Fast Multi-View Face Detection. In *Conference on Computer Vision and Pattern Recognition*, 2003.

[236] P. Viola and M. Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *Conference on Computer Vision and Pattern Recognition*, 2001.

[237] P. Viola and M. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[238] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose Tracking from Natural Features on Mobile Phones. In *International Symposium on Mixed and Augmented Reality*, September 2008.

[239] B. Wang, J. Tang, W. Fan, S. Chen, Z. Yang, and Z. Liu. Heterogeneous Cross Domain Ranking in Latent Space. In *Conference on Information and Knowledge Management*, 2009.

[240] J. Wang, S. Kumar, and S.-F. Chang. Sequential Projection Learning for Hashing with Compact Codes. In *International Conference on Machine Learning*, 2010.

[241] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jinbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[242] K. Wang, B. Babenko, and S. Belongie. End-To-End Scene Text Recognition. In *International Conference on Computer Vision*, 2011.

[243] S. Wang and X. Sun. Generalization of Hinging Hyperplanes. *IEEE Transactions on Information Theory*, 51(12):4425–4431, 2005.

[244] X. Wang, C. Zhang, and Z. Zhang. Boosted Multi-Task Learning for Face Verification with Applications to Web Image and Video Search. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[245] Z. Wang, B. Fan, and F. Wu. Local Intensity Order Pattern for Feature Description. In *International Conference on Computer Vision*, 2011.

[246] K.Q. Weinberger and L.K. Saul. Fast Solvers and Efficient Implementations for Distance Metric Learning. In *International Conference on Machine Learning*, 2008.

[247] Y. Weiss, R. Fergus, and A. Torralba. Multidimensional Spectral Hashing. In *European Conference on Computer Vision*, 2012.

[248] Y. Weiss, A. Torralba, and R. Fergus. Spectral Hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760, 2009.

[249] B. Williams, G. Klein, and I. Reid. Real-Time Slam Relocalisation. In *International Conference on Computer Vision*, 2007.

[250] S.A. Winder and M. Brown. Learning Local Image Descriptors. In *Conference on Computer Vision and Pattern Recognition*, June 2007.

[251] S.A. Winder, G. Hua, and M. Brown. Picking the Best Daisy. In *Conference on Computer Vision and Pattern Recognition*, June 2009.

[252] P. Wohlhart and V. Lepetit. Learning Descriptors for Object Recognition and 3D Pose Estimation. In *Conference on Computer Vision and Pattern Recognition*, 2015.

[253] C. Wojek, S. Walk, and B. Schiele. Multi-Cue Onboard Pedestrian Detection. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[254] R. Zabih and J. Woodfill. Non Parametric Local Transforms for Computing Visual Correspondences. In *European Conference on Computer Vision*, pages 151–158, May 1994.

[255] M. Zervos, H. BenShitrit, F. Fleuret, and P. Fua. Facial descriptors for identity-preserving multiple people tracking. Technical report epfl-article-187534, Ãcole Polytechnique FÃ©dÃ©rale de Lausanne (EPFL), 2013. `http://michal.is/projects/people-tracking-identification/`.

[256] G. Zhao and M. Pietikainen. Local Binary Pattern Descriptors for Dynamic Texture Recognition. In *International Conference on Pattern Recognition*, pages 211–214, 2006.

[257] F. Zheng and G.I. Webb. A Comparative Study of Semi-Naive Bayes Methods in Classification Learning. In *Australasian Data Mining Conference*, pages 141–156, 2005.

[258] A. Ziegler, E. Christiansen, D. Kriegman, and S. Belongie. Locally Uniform Comparison Image Descriptor. In *Advances in Neural Information Processing Systems*, 2012.

[259] C.L. Zitnick. Binary Coherent Edge Descriptors. In *European Conference on Computer Vision*, 2010.

[260] C.L. Zitnick and K. Ramnath. Edge Foci Interest Points. In *International Conference on Computer Vision*, 2011.