Fast Keypoint Recognition in Ten Lines of Code [*]

Mustafa Özuysal Pascal Fua Vincent Lepetit
Computer Vision Laboratory
École Polytechnique Fédérale de Lausanne (EPFL) 1015 Lausanne, Switzerland
Email: {Mustafa.Oezuysal, Pascal.Fua, Vincent.Lepetit}@epfl.ch

Abstract

While feature point recognition is a key component of modern approaches to object detection, existing approaches require computationally expensive patch preprocessing to handle perspective distortion. In this paper, we show that formulating the problem in a Naive Bayesian classification framework makes such preprocessing unnecessary and produces an algorithm that is simple, efficient, and robust. Furthermore, it scales well to handle large number of classes.

To recognize the patches surrounding keypoints, our classifier uses hundreds of simple binary features and models class posterior probabilities. We make the problem computationally tractable by assuming independence between arbitrary sets *of features. Even though this is not strictly true, we demonstrate that our classifier nevertheless performs remarkably well on image datasets containing very significant perspective changes.*

1. Introduction

The ability to recognize interest points across images that may have been taken from very different viewpoints is required to address many important Computer Vision problems. They range from image registration to object detection [7, 6] and often require real-time performance. The standard approach to addressing this problem is to build affine-invariant descriptors of local image patches and to compare them across images. This usually involves fine scale selection, rotation correction, and intensity normalization [11, 10]. It results in a high computational overhead and often requires handcrafting the descriptors to achieve insensitivity to specific kinds of distortion.

It has recently been shown that casting this wide-baseline matching problem as a more generic classification problem leads to solutions that are much less computationally demanding [9]. This approach relies on an offline training phase during which multiple views of the keypoints to be matched are used to train randomized trees [2] to recognize them based on a few pairwise intensity comparisons. This yields both fast run-time performance and robustness to viewpoint and lighting changes, which has proved very effective for real-time object detection.

In this paper, we show that using a classic Naive Bayesian framework yields an approach that is simpler, faster, and as robust as the state-of-the-art methods discussed above. We use non-hierarchical structures that we refer to as *ferns* to classify the patches. Each one 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 [9], 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 binary tests we use as classifier features are picked completely at random, which puts our approach firmly in the camp of techniques that rely on randomization to achieve good performance [1]. We will show that this is particularly effective for the specific classification task we are addressing, which requires scale and perspective invariance, involves 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 keypoints. Furthermore, our approach is particularly easy to implement, does not overfit, does not require *ad hoc* patch normalization, and allows fast and potentially incremental training.

2. Image Patch Classification

The importance of image patch recognition and matching across images is widely accepted for applications ranging from object recognition and image retrieval to pose estimation. Given feature points extracted from the images, two main classes of approaches have been proposed to achieve results such as those of Figure 6.

The first family involves computing local descriptors in-

[*]This work has been supported in part by the Swiss National Science Foundation.

variant to changes such as perspective and lighting [13, 10]. In particular the SIFT vector [10], computed from local histograms of gradients, works remarkably well, at least on textured images and we will use it as a benchmark for our own approach.

A second class relies on statistical learning techniques to model the set of possible appearances of a patch. The one-shot approach of [6] uses PCA and Gaussian Mixture Models but does not account for perspective distortion. This is addressed in [9] using Randomized Trees (RTs). Since the set of possible patches around an image feature under changing perspective and lightning conditions can be seen as a class, it is possible to train a set of RTs to recognize feature points by feeding it samples of their possible appearances, synthesized by warping the patches found in a training image using randomly chosen homographies.

This approach is fast and effective to achieve the kind of object detection depicted by Figure 6. Note that unlike in traditional classification problems, a close-to-perfect method is not required. Here it is enough to recognize some features succesfully and to use a robust estimator such as RANSAC to detect the object. However a scalable approach is still highly desirable for practical applications since the number of keypoints might become very large (typically $>$ 400). We will show that when this happens the performance of the RTs tends to drop whereas that of the ferns does not.

Recently [12] used keypoints as visual words [14] for image retrieval in very large image databases. Keypoints are labeled by hierarchical k-means [5] clustering based on their SIFT descriptors. This allows a very large number of visual words, but the performance measure is the number of correctly retrieved documents rather than number of correctly classified keypoints. In this work, we concentrate on localizing individual keypoints to obtain pose information which is required in tracking and augmented reality applications.

3. Naive Bayesian Classification

It has been shown that 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 [9]. 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.

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. The naive combination strategy lets us combine many more features, which is key to improved recognition rate.

3.1. Formulation

As discussed in Section 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

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

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

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}}\, P(f_1, f_2, \ldots, f_N \mid C = c_i)\,. \tag{1}$$

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 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 they are very simple, we require many ($N \approx 300$) for accurate classification. Therefore a complete representation of the joint probability in Eq. (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

$$P(f_1, f_2, \ldots, f_N \mid C = c_i) = \prod_{k=1}^{M} P(F_k \mid C = c_i)\,, \tag{2}$$

```
for $i = 1$ to $H$
    $\log P_{I \mid C}[i] \leftarrow 0$
end for
for all fern $F_k$ do
    index $\leftarrow 0$
    for $j = 1$ to $S$
        index $\leftarrow 2 \times$ index
        if $I\left(\mathbf{d}_{\sigma(k,j,1)}\right) < I\left(\mathbf{d}_{\sigma(k,j,2)}\right)$ then
            index $\leftarrow$ index $+ 1$
        end if
    end for
    for $i = 1$ to $H$
        $\log P_{I \mid C}[i] \leftarrow \log P_{I \mid C}[i] + \log P_{F_k}[\text{index}, i]$
    end for
end for
```

```
for(int i = 0; i < H; i++) P[i] = 0.;

for(int k = 0; k < M; k++) {
  int index = 0, * d = D + k * 2 * S;
  for(int j = 0; j < S; j++) {
    index <<= 1;
    if (*(K + d[0]) < *(K + d[1]))
      index++;
    d += 2;
  }
  p = PF + k * shift2 + index * shift1;
  for(int i = 0; i < H; i++) P[i]+=p[i];
}
```

Figure 1. **Left:** The pseudo-code of the run-time algorithm that computes $P(f_1, f_2, \ldots, f_N \mid C = c_i)$ as given by Eq. (2) to classify the image patch I, where *index* is an integer index computed from the binary features. No image rectification, illumination normalization, or parameter tuning are required. **Right:** A C++ implementation of the pseude-code. The code used for training is very similar.

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 [15] approach by modelling only some of the dependencies between features. The viability of such an approach has been shown by [8] in the context of image retrieval applications.

This form can now be handled easily since the it has $M \times 2^S$ parameters with M between 30-50, and we show in Section 4 that a fern size S around 10 gives good recognition rates, compared to the 2^N with $N \approx 300$ for the full joint probability representation. It is also flexible since performance/memory trade-offs can be made by changing the number of ferns and their sizes. The corresponding code is given as Figure 1 to highlight the simplicity of the resulting implementation.

3.2. Training

For our experiments, we start the training by constructing a set of H prominent keypoints lying on the object model. To each feature point corresponds a class.

The fern features, that is the locations $\mathbf{d}_{j,1}$ and $\mathbf{d}_{j,2}$, are picked at random. The terms

$$P(F_k \mid C = c_i),\ k = 1,\ \ldots,\ M$$

are estimated by computing the features on training samples of each class. We can exploit here our strong knowledge on the problem to create a virtually infinite training set: We use a small number of images and synthesize many new views of the object using simple rendering techniques as affine deformations, and extract training patches for each class. White noise is also added for more realism. For each keypoint on the model, this gives us a fine sampling of the set of all its possible appearances under different viewing conditions.

However even if each term $P(F_k \mid C = c_i)$ is only a part of the full joint probability of Eq. (1), their estimation still involves estimating an extremely large number of parameters, and they cannot be reliably estimated directly as empirical probabilities in practice. In order to explain how we estimate the $P(F_k \mid C = c_i)$, lets us introduce the event $\Theta(F_k)$ that states that "*the empirical probabilities for F_k are reliable*". We can then express a $P(F_k \mid C = c_i)$ term as:

$$\begin{aligned} P(F_k \mid C = c_i) \quad = \quad & P(F_k \mid C = c_i, \Theta(F_k)) P(\Theta(F_k)) + \\ & P(F_k \mid C = c_i, \overline{\Theta(F_k)}) P(\overline{\Theta(F_k)}) \end{aligned} . \tag{3}$$

$P(F_k \mid C = c_i, \Theta(F_k))$ is nothing more than the empirical probability of $P(F_k \mid C = c_i)$, and $P(F_k \mid C = c_i, \overline{\Theta(F_k)})$ should be taken as constant and is therefore equal to $\frac{1}{H}$. Let us now model $P(\Theta(F_k))$ as:

$$P(\Theta(F_k)) = \frac{\sum_i n_{k,i}}{\sum_i (n_{k,i} + u)} ,$$

where $n_{k,i}$ is the number of training samples that verify the set of features F_k. When the training set is truly representative of the actual variations within classes, this model makes sense since it tends to 1 when the number of training samples grows, and yields a simple way to estimate the $P(F_k \mid C = c_i, \Theta_k)$. It is easy to check that we have then:

$$P(F_k \mid C = c_i) = \frac{n_{k,i} + u}{\sum_k (n_{k,i} + u)} .$$

In practice, the value of u does not really influence the results as soon as it is higher than 0. In all our experiments, we use $u = 1$. This factor can be interpreted as a Dirichlet prior, since the class conditional probabilities are modelled as multinomials [4].

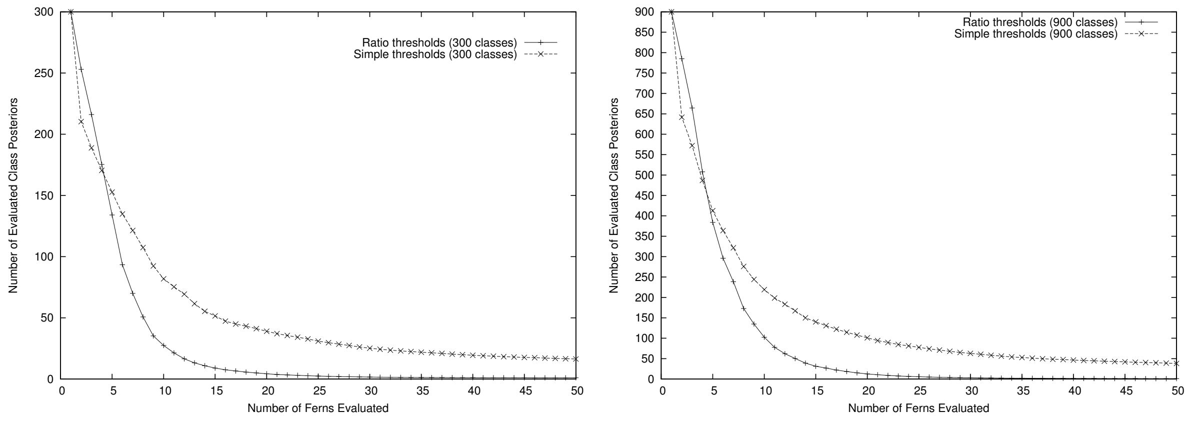


Figure 2. The number of class posteriors that needs to be evaluated decreases very rapidly when the probabilities are thresholded by their ratio to the maximum probability. Plots show these curves when 300 and 900 classes are trained respectively.

Figure 3. Two of the images used for evaluation.

	300 Classes	900 Classes
No Thresholding	93.2	87.2
Simple Thresholds	87.2	80.6
Ratio Thresholds	90.2	84.1

Table 1. Percentage of correctly classified image patches with and without thresholding. Since the thresholds are calculated using the training set they can cause misclassification on a test set. In the case of ratio thresholds this loss of performance is not significant.

3.3. Handling Many Classes

At run-time, computing the class probabilities takes a single lookup for each fern and their final multiplication. However this has to be repeated for each class and as the number of classes increases this quickly becomes burdensome. Furthermore some of the class posteriors reach very small values at the end of a multiplication of the first few terms and their final value becomes irrelevant for class selection. In principle we do not have to calculate the final value of the posterior for each class as long as the selected class does not change.

Here we consider two strategies for eliminating classes during posterior evaluation as each term coming from a fern is multiplied, so that the computation can be carried out much more quickly. The first strategy is to use a simple threshold, which can be learned from the training set, on the posteriors as each term gets multiplied. This eliminates classes that are unlikely to be the correct class. The second approach is to use a threshold on the ratio of the maximum posterior to the considered class posterior at each step. This is based on the observation that a class posterior that has fallen back by a large margin is unlikely to catch up. Note that this second strategy requires the computation of the maximum posterior at the end of each step.

Figure 2 shows the average number of class posteriors calculated at the end of each step for the two thresholding strategies. As the plots clearly show, thresholds on the ratios to maximum probability at each step decreases the number of necessary evaluations significantly. The thresholds were chosen to be as large as possible without causing a misclassification on the training set. For these experiments, we used the two images shown Figure 3, and the percentage of correctly classified image patches evaluated on randomly generated images of these images. As can be observed from Table 1 the ratio thresholds decrease the classification rate only slightly and therefore can generalize well.

4. Comparing Ferns and Trees

Ferns and Random Trees are very similar in spirit but differ in two important respects. In trees the binary tests are organized hierarchically and the posterior distributions are computed additively. By contrast, ferns are flat and compute posteriors multiplicatively. In this section, we first compare the two approaches. We then offer a theoretical insight into why ferns appear to outperform trees, but only when the training set is sufficiently large.

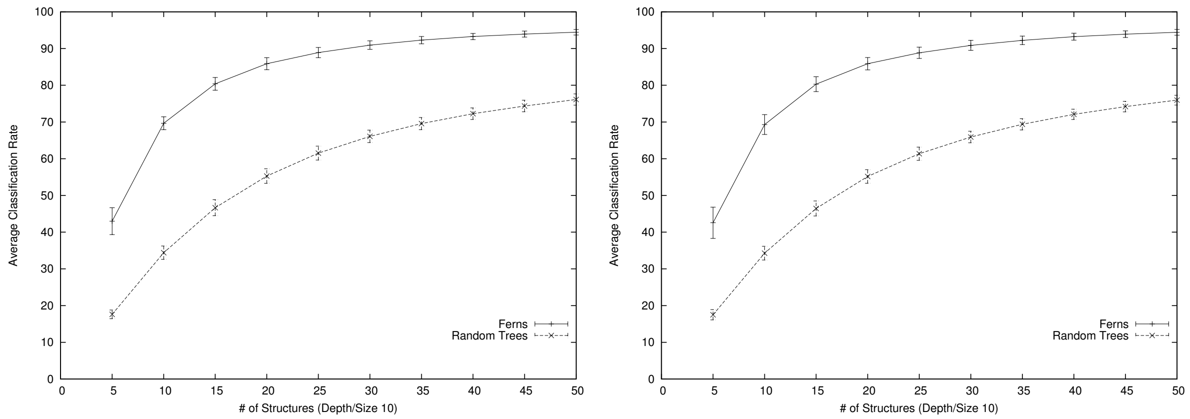


Figure 4. The percentage of correctly classified image patches is shown against different classifier parameters for the fern and tree based methods. The independence assumption between ferns allows the joint utilization of the features resulting in a higher classification rate. The error bars show the 95% confidence interval assuming a Gaussian distribution.

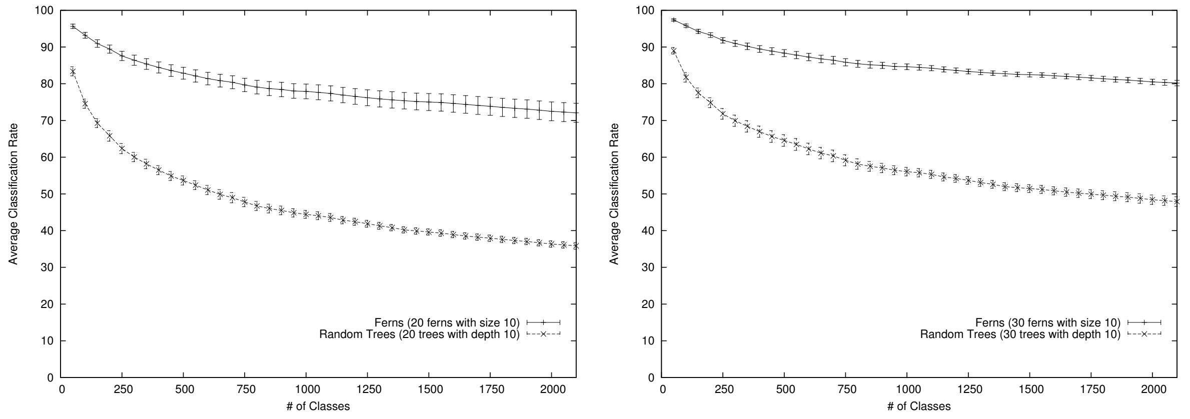


Figure 5. Classification using ferns can handle many more classes than Random Tree based methods. For both figures ferns with size 10 and trees with depth 10 are used.

4.1. Ferns Outperform Trees

We evaluate the performance of the proposed fern based approach by comparing to the results of a Random Tree based implementation. The number of tests in the ferns and the depth of the trees are taken to be equal, and we compare the classification rate when using the same number of structures, that is of ferns or Random Trees. In particular, the same number of tests is performed on each keypoint, and the same number of joint probabilities has to be stored.

We do our tests on the images presented 3 with 500 classes and calculate the average classification rate on randomly generated test images while eliminating false matches using object geometry. Since the feature selection is random, we repeat the test 10 times and calculate the mean and variance of the classification rate and we perform the test on the two images.

As depicted by Figure 4, despite the inaccuracy of the independence assumptions the fern based classifier outperforms the combination of trees. Furthermore as the number of ferns is increased the random selection method does not cause large variations on the classifier performance.

We also investigate the behavior of the classification rate as the number of classes increases. Figure 5 shows that a larger number of classes does not affect the performance of ferns much, while tree based methods can not cope with many classes. In both experiments we have trained the classifiers using classes from three different images up to 700 classes for each image.

4.2. Linking the Two Approaches

Here we show that the two approaches are equivalent when the training set is small and give some insights into why the ferns perform better when it is large.

Recall that we evaluate $P(F_k \mid C = c_i)$ of Eq. (3)

$$P(F_k \mid C = c_i) = P_e P(\Theta(F_k)) + \mu(1 - P(\Theta(F_k))) \,,$$

where P_e is the empirical probability and $\mu = \frac{1}{H}$. Since computing the product of such terms is the same as summing their logarithms, we write

$$\begin{aligned}
& \log P(F_k \mid C = c_i) \\
= \; & \log \left[\mu \left(P(\Theta(F_k)) \left(\frac{P_e}{\mu} - 1\right) + 1\right)\right] \\
= \; & \log \mu + \log \left[1 + P(\Theta(F_k)) \left(\frac{P_e}{\mu} - 1\right)\right] \\
\approx \; & \log \mu + P(\Theta(F_k)) \left(\frac{P_e}{\mu} - 1\right) \text{ when } P(\Theta(F_k)) \text{ is small} \\
= \; & \frac{P(\Theta(F_k))}{\mu} + (\log \mu - P(\Theta(F_k))) \text{ when } P(\Theta(F_k)) \text{ is small} \\
= \; & a P_e + b,
\end{aligned}$$

where a and b depend on $P(\Theta(F_k))$ and μ.

Therefore, when the training set is small and the empirical probabilities are only crude estimates of the true ones, there is not much difference in selecting the class either based on the maximum of the products of the $P(F_k \mid C = c_i)$ as the ferns do, or, as the maximum of the sum of the empirical probabilities, as the trees do. In this case, the two approaches are roughly equivalent.

By contrast, our experiments show that when the training set is sufficient large the ferns perform much better. This can be understood as follows: As soon as one single fern attributes a very low probability to one class, the final computed probability for the class is guaranteed to be low because of their multiplicative behavior. This does not occur with trees due to their additive behavior. This increases the discriminating power of ferns but requires posterior probabilities that can be trusted. This is why the evaluation method of Section 3.2 is required.

5. Results

We compare our ferns against SIFT [10], which is widely reported as one of the most effective approaches. It combines orientation estimation and SIFT descriptors to achieve viewpoint invariance. We will show that ferns let us omit costly preprocessing steps without loss in terms of performance and sometimes even yield better performance. We first compare classification rates and then computation times.

5.1. Comparing Matching Rates

We compared the match rates of SIFT and ferns on the sequence depicted by Figure 6, in which a mouse pad was moved in front of a moving camera over a cluttered background. The mouse pad was undergoing large displacements, which produces image blur and large scale, perspective, and illumination changes.

In the case of SIFT, we used the multi-resolution code kindly provided by David Lowe, which computes the Laplacian at several levels for each octave. By contrast, to test the ferns, we used a simple keypoint detector that considers extrema of the Laplacian over 3 octaves only. This implies that the patches our ferns have to work with are only roughly correctly scaled whereas those fed to SIFT are computed using a finely estimated scale. In other words, we use a more primitive and simpler to implement keypoint extraction method, which should handicap us, but does not really as we shall see.

We then train 20 ferns of size 14 as described in Section 3 and use them to establish matches between the model image and the sequence images selecting the most probable class for each keypoint in the test image. In parallel, we compute SIFT descriptors for the keypoints extracted from the model images and match each of them against the keypoints in the sequence images by selecting the one which has the nearest SIFT descriptor. We retain the 400 strongest keypoints in the reference image, and 1000 keypoints in the sequence images for the two methods. Then in both cases we use robust RANSAC estimation to compute the homography between the images, which is then refined using a non-linear estimation method using all matches that are compatible with it. All matches are checked against this homography and those who reprojection using this homography is within 10 pixels are retained as inliers.

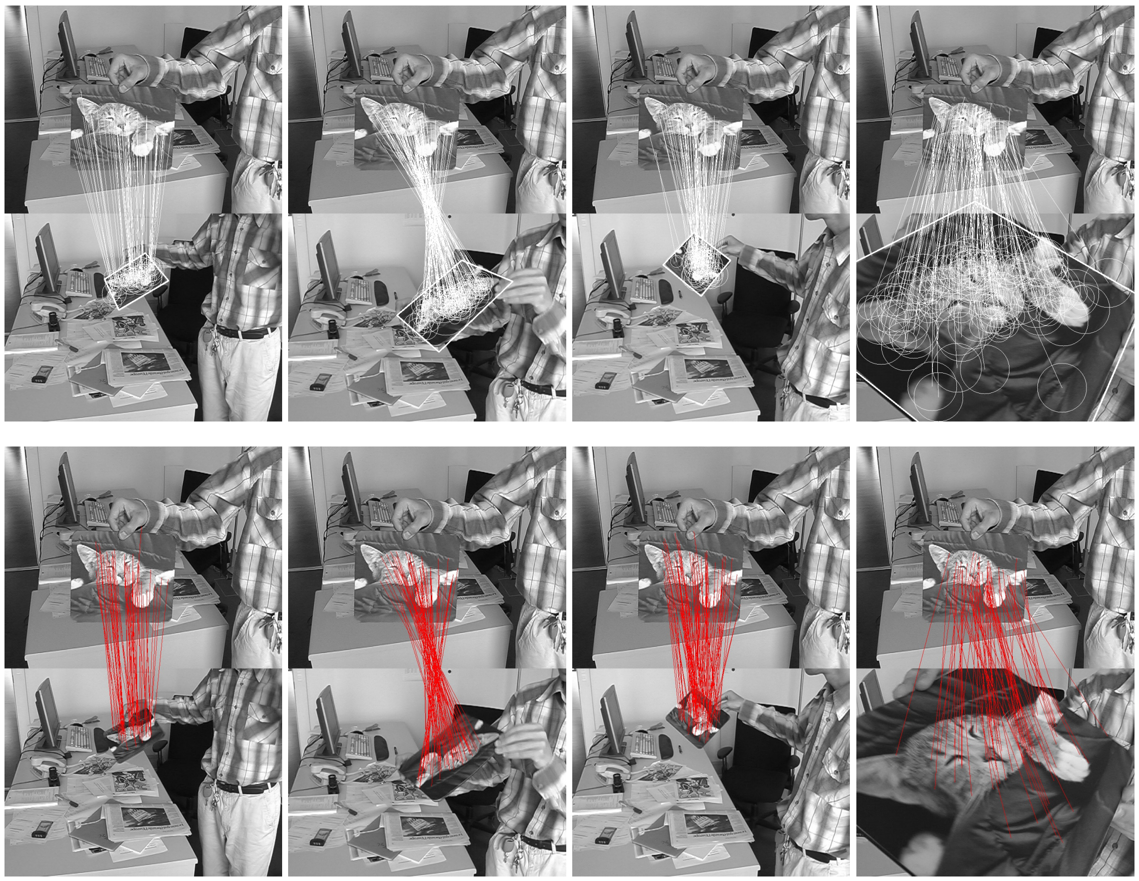
The graph of Figure 6 depicts the number of correct matches obtained by both methods for all frames in the sequence. Despite their simplicity, ferns match at least as many points as SIFT and often even more.

As shown Figure 7, we also applied ferns on face images to test the effect of non-planarity. Despite the planarity assumption made for simplicity when generating the synthetic training views, the ferns still perform well. The results obtained with SIFT are similar and not repeated here.

5.2. Comparing Computation Times

It is difficult to perform a completely fair 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 K-tree of [3] 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 3 requires very little and most of the time is spent computing the sums of the posterior probabilities. Without taking into account the strategies of Section 3.3 to speed things up, 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 K-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. Still note that in contrast to our method SIFT does not require a training phase.



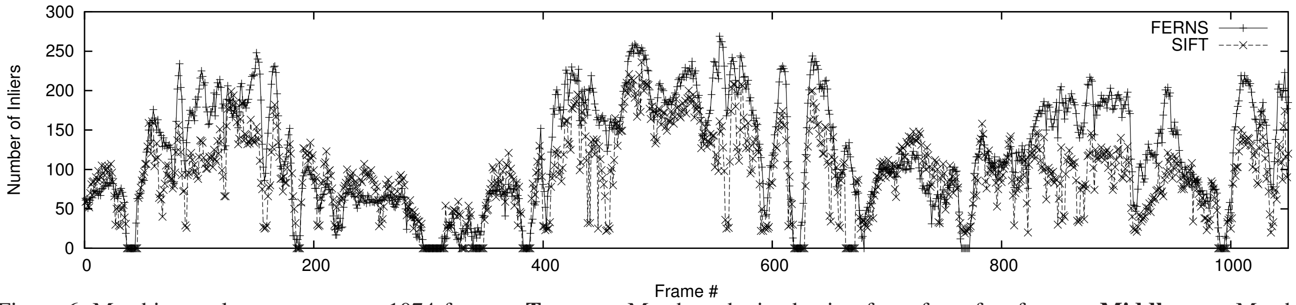


Figure 6. Matching a planar target over 1074 frames: **Top row.** Matches obtained using ferns for a few frames. **Middle row.** Matches obtained using SIFT for the same frames. **Bottom row.** Number of correct matches obtained by both methods for all frames in the sequence. Ferns match at least as many points as SIFT and often even more. The video sequence is available as supplemental material.

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 fifth of the time is taken by the extraction. This corresponds to a theoretical 50Hz frame rate if one does factor in the time required for frame acquisition. Training takes less than five minutes.

6. Conclusion

We have presented a general method for image patch recognition that is effective for object pose estimation despite severe perspective distortion. The "semi-naive" struc-

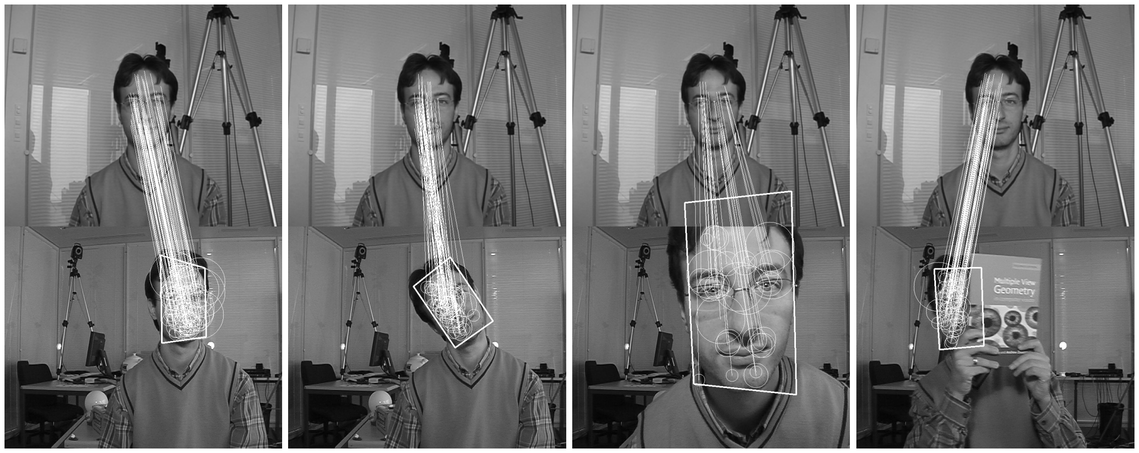


Figure 7. Matching face images with ferns. Despite the non-planarity of faces, ferns are still effective. Results with SIFT are similar and not shown here.

ture of ferns is well adapted and allows 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 computing and discriminative power. As computers become more powerful, we can add more ferns to improve the performance. Conversely, one can adapt to low computational power such those on hand-held systems by reducing the number of ferns.

References

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

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

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

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

[5] 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*, 45:807–815, 2005.

[6] 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.

[7] R. Fergus, P. Perona, and A. Zisserman. A Sparse Object Category Model for Efficient Learning and Exhaustive Recognition. In *Conference on Computer Vision and Pattern Recognition*, July 2005.

[8] D. Hoiem, R. Sukthankar, H. Schneiderman, and L. Huston. Object-based image retrieval using the statistical structure of images. *Conference on Computer Vision and Pattern Recognition*, 02:490–497, 2004.

[9] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, Sept. 2006.

[10] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 20(2):91–110, 2004.

[11] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1/2):43–72, 2005.

[12] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. In *Conference on Computer Vision and Pattern Recognition*, 2006.

[13] 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.

[14] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*, pages 127–144. Springer, 2006.

[15] F. Zheng and G. Webb. A comparative study of semi-naive bayes methods in classification learning. In *Proceedings of the Fourth Australasian Data Mining Conference (AusDM05)*, pages 141–156, Sydney, 2005.