Noname manuscript No.
(will be inserted by the editor)

Learning Real-Time Perspective Patch Rectification

Stefan Hinterstoisser · Vincent Lepetit · Selim Benhimane · Pascal Fua · Nassir Navab

Received: date / Accepted: date

Abstract We propose two learning-based methods to patch rectification that are faster and more reliable than state-of-the-art affine region detection methods. Given a reference view of a patch, they can quickly recognize it in new views and accurately estimate the homography between the reference view and the new view. Our methods are more memory-consuming than affine region detectors, and are in practice currently limited to a few ten patches. However, if the reference image is a fronto-parallel view and the internal parameters known, one single patch is often enough to precisely estimate an object pose. As a result, we can deal in real-time with objects that are significantly less textured than the ones required by state-of-the-art methods.

The first method favors fast run-time performance while the second one is designed for fast real-time learning and robustness, however they follow the same general approach: First, a classifier provides for every keypoint a first estimate of its transformation. Then, the estimate allows carrying out an accurate perspective rectification using linear predictors. The last step is a fast verification—made possible by the accurate perspective rectification—of the patch identity and its sub-pixel precision position estimation. We demonstrate the advantages of our approach on real-time 3D object detection and tracking applications.

Keywords Patch Rectification · Tracking by Detection · Object Recognition · Online Learning · Real-Time Learning · Pose Estimation

Stefan Hinterstoisser, Selim Benhimane, Nassir Navab
Computer Aided Medical Procedures (CAMP)
Technische Universität München (TUM)
Boltzmannstrasse 3, 85748 Munich, Germany
Tel.: +49-89-28919400
Fax: +49-89-28917059
E-mail: {hinterst,benhiman,navab}@in.tum.de

Vincent Lepetit, Pascal Fua
Computer Vision Laboratory (CVLab)
École Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
Tel.: +41-21-6936716
Fax: +41-21-6937520
E-mail: {pascal.fua,vincent.lepetit}@epfl.ch

1 Introduction

Retrieving the poses of patches around keypoints in addition to matching them is an essential task in many applications such as vision-based robot localization [9], object recognition [25] or image retrieval [8,24] to constrain the problem at hand. It is usually done by decoupling the matching process from the keypoint pose estimation: The standard approach is to first use some affine region detector [19] and then rely on SIFT [15] or SURF [3] descriptors on the rectified regions to match them.

Recently, it has been shown that taking advantage of a training phase, when possible, greatly improves the speed and the rate of keypoint recognition tasks [16,22]. Such a training phase is possible when the application relies on some *database* of keypoints, such as object detection or SLAM. By contrast with [19], these learning-based approaches usually do not rely on the extraction of local patch transformations in order to handle larger perspective distortions but on the ability to generalize well from training data. The drawback is they only provide a 2–D location, while using an affine region detector provides additional constraints that proved to be useful [25,8].

To overcome this problem we introduce an approach illustrated in Fig. 1 and that can provide not only an affine transformation but the full perspective patch rectification and that is still real-time thanks to a learning stage. We show this is very useful for object detection and SLAM applications: Applying our approach on a single keypoint is often enough

Fig. 1 The advantages of learning for patch recognition and pose estimation. **(a)** Given a training images or a video sequence, our method learns to recognize patches and in the same time to estimate their transformation. **(b)** The results are very accurate and mostly exempt of outliers. Note we get the full perspective pose, and not only an affine transformation. **(c)** Hence a single patch is often sufficient to detect objects and estimate their pose very accurately. **(d)** To illustrate the accuracy, we use the 'Graffiti 1' image and the ICCV booklet cover respectively to train our method and detect patches in the 'Graffiti 6' image and in the real scene respectively. We then superimpose the retrieved transformations with the original patches warped by the ground truth homography. **(e)** Even after zooming, the errors are still barely visible. **(f)** By contrast, the standard methods retrieve comparatively inaccurate transformations, which are limited to the affine transformation group.

to estimate the 3–D pose of the object the keypoint lies on, provided that a fronto-parallel view of the keypoint is given for training. As a result, we can robustly handle very poorly textured or strongly occluded objects.

More exactly we propose two methods based on this approach. The first method was called LEOPAR in [11] where it was originally published, and will be refer as ALGO1 in this paper. It is the faster method, and still more accurate than affine region detectors. The second method was called GEPARD in [12], and will be referred as ALGO2. ALGO2 produces the more reliable results and requires only a very fast training stage. Choosing between the two methods depends on the application at hand.

Both methods are made of two stages. The first stage relies on a classifier to quickly recognize the keypoints and provide a first estimate of their poses to the second stage. This second stage uses relatively slower but much more accurate template matching techniques to refine the pose estimate. The difference between the two methods lies in the way the first stage proceeds.

ALGO1 first retrieves the patch identity and then a coarse pose using an extended version of the Ferns classifier [22]: To each keypoint in our database correspond several classes, where each class covers its possible appearances for a restricted range of poses. Due to the Fern structure the computations can be done in almost no time which results in a very fast runtime performance.

Unfortunately the Ferns require a long training stage and a large amount of memory. ALGO2 dramatically decreases the training time and the memory amount and is even more accurate; however, it is slower than ALGO1 at run-time. In ALGO2, the Ferns classifier is replaced by a simple nearest-neighbour classifier, as new classes can be added quickly to such a classifier. To retrieve the incoming keypoints identities and approximate poses, each keypoint in the database is characterized in the classifier by a set of "mean patches", each of them being the average of the keypoint appearances over a restricted range of poses. Our mean patches are related to Geometric Blur [6], but we show how to very quickly compute them, making our approach more efficient.

To retrieve an accurate full perspective transformation, the second stage uses linear regressors similar to the one described in [13] for template matching. We made this choice because our experiments proved they converge faster and more accurately than other least-squares optimization such as Gauss-Newton for this purpose. In addition, we show that these regressors can be trained efficiently The final pose es-

timate is typically accurate enough to allow a final check by simple cross-correlation and prune the incorrect results.

Compared to affine region detectors, their closest competitors in the state-of-the-art, our two methods have one important limitation: They do not scale very well with the size of the keypoints database, and our current limitations are limited to a few tens of keypoints to keep real-time the applications. Moreover, they need a frontal training view and the camera internal parameters to compute the camera pose with respect to the keypoint. However, as our experiments show, our two methods not only are much faster but they provide an accurate 3–D pose for each keypoint, by contrast with an approximate affine transformation. In practice, a single keypoint is often enough to compute the camera or the target pose, which compensates this limitation on the database size at least for the applications we present in this paper.

In the remainder of the paper, we first discuss related work. Then, we describe our two methods, and compare them against affine region detectors [19]. Finally, we present applications of tracking-by-detection and SLAM using our method.

2 Related Work

Many different approaches often called "affine region detectors" have been proposed to recognize keypoints under large perspective distortion. For example, [27] generalized the Förstner-Harris approach, which was designed to detect keypoints stable under translation, to small similarities and affine transformations. However, it does not provide the transformation itself. Other methods attempt to retrieve a canonical affine transformation without *a priori* knowledge. This transformation is then used to rectify the image around the keypoint and make them easier to recognize. For example, [19] showed that the Hessian-Affine detector of [18] and the MSER detector of [17] are the most reliable ones. In the case of the Hessian-Affine detector, the retrieved affine transformation is based on the image second moment matrix. It normalizes the region up to a rotation, which can then be estimated for example by considering the peaks of the histogram of gradient orientations over the patch as in SIFT [15]. In the case of the MSER detector, other approaches exploiting the region shape are also possible [21], and a common approach is to compute the transformation from the region covariance matrix and solve for the remaining degree of freedom using local maximums of curvature and bitangents.

But besides helping the recognition, the estimated affine transformations can provide useful constraints. For example, [25] uses them to build and recognize 3–D objects in stereoscopic images. [8] uses them to add constraints between the different regions and help matching them more reliably.

Unfortunately, as the experiments presented in this paper show, the retrieved transformations are often not accurate. We will show that our learning-based methods can reach a much better accuracy.

Learning-based methods to recognize keypoints became quite popular recently, however all the previous methods provide only the *identity* of the points, not their pose. For example, in [14], Randomized Trees are trained with randomly warped patches to estimate a probability distribution over the classes for each leaf node. The non-terminal nodes contain decisions based on pairwise intensity comparisons which are very fast to compute. Once the trees are trained an incoming patch is classified by adding up the probability distributions of the leaf nodes that were reached and by identifying the class with the maximal probability. However, training the Randomized Trees is slow and performed offline, which is problematic for applications such as SLAM. [28] replaced the Randomized Trees by a simpler list structure and binary values instead of a probability distribution. These modifications allow them to learn new features online in real-time. Another approach based on the boosting algorithm presented in [10] to allow online feature learning in real-time was proposed in [16].

More recently, [26] introduced a learning-based approach that is closer to the approach presented in this paper. It is based on what is called "Histogrammed Intensity Patches" (HIP). The link with our approach is that the HIPs are reminiscent of our "mean patches" used in ALGO2: Each keypoint in the database is represented by a set of HIPs, each of them computed over a small range of poses. For fast indexation, a HIP is a binarized histogram of the intensities of a few pixels around the keypoint. However, while this is in theory possible, estimating the keypoint pose has not been evaluated nor demonstrated, and this method too provides only the keypoint identities.

Another work related to this paper is [20], which exploits the perspective transformation of patches centered on landmarks in a SLAM application. However, it is still very depending on the tracking prediction to match the landmarks and to retrieve their transformations, while we do not need any prior on the pose. Moreover, in [20], these transformations are recovered using a Jacobian-based method while, in our case, a linear predictor can be trained very efficiently for faster convergence.

In short, to the best of our knowledge, there is no method in the literature that attempts to reach the exact same goal as ours. Our two methods can estimate quickly and accurately the pose of keypoints of a database, thanks to a learning-based approach.

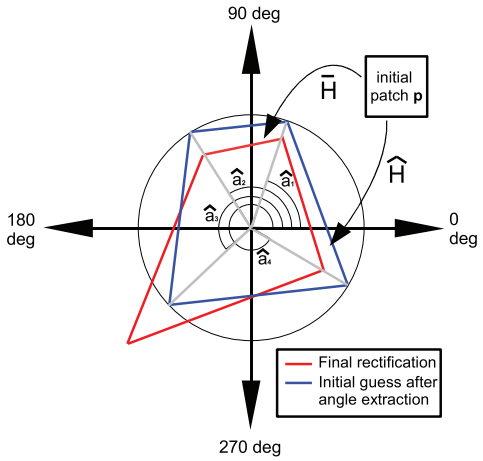
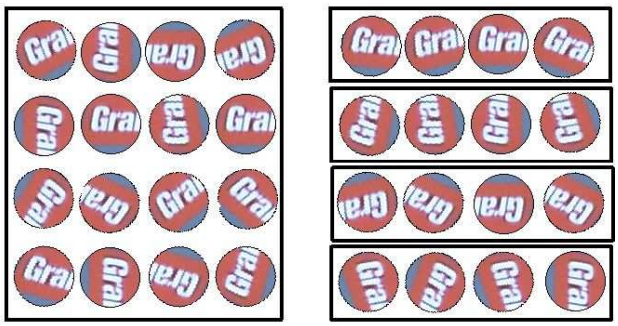


Fig. 2 A first estimate of the patch transformation is obtained using a classifier that provides the values of the angles $\mathbf{a}_i$ defined as the angles between the lines that go through the patch center and each of the four corners.



(a) (b)

Fig. 3 Examples of patches used for classification. (a) To estimate the keypoint identity, patches from the same keypoint are grouped in a single class. (b) To estimate the patch transformation, several classes for different transformations are created for each keypoint in the database.

3 ALGO1: A Classifier Favoring Run-Time Performance

We present in this section the first stage of our first method. It provides the identity and an approximate pose of a keypoint, given an image patch centered on this keypoint, using an extension of [22]. The second stage, the keypoint pose refinement and checking steps, is common to our two methods, and will be presented Section 5.

3.1 Finding the Keypoint's Identity

ALGO1 first recognizes the keypoint to which the patch corresponds to by using the Ferns classifier presented in [22]. Ferns are trained with patches centered on the keypoints in the database and seen under different viewing conditions as in Fig. 3(a). Formally, for a given patch $\mathcal{P}$ centered on a keypoint we want to recognize, it estimates:

$$\widehat{id} = \underset{id}{\operatorname{argmax}} P(Id = id \mid \mathcal{P}) \,, \tag{1}$$

where Id is a random variable representing the identity of the keypoint. The identity is simply the index of the corresponding keypoint in the database. The classifier represents the patch $\mathcal{P}$ as a set of simple image binary features that are grouped into subsets, and Id is estimated following a semi-Naive Bayesian scheme that assumes the feature subsets independent. This classifier is usually able to retrieve the patch identity Id under scale, perspective and lighting variations.

3.2 Discretizing and Estimating the Keypoint's Pose

Once Id is estimated, our objective is then to get an estimate of the transformation of the patch around the keypoint. Because we also want to use a classifier for that, we first need a way to quantize the transformations. We tried various approaches, and the best results were obtained with the parametrization described in Fig. 2. It is made of the four angles $\mathbf{a}_i$ between the horizontal axis and the semi-lines going from the patch center and passing through the patch corners. Each angle is quantized into 36 values, and to reduce the required amount of memory and to increase the speed at runtime, we estimate each angle independently as:

$$\forall i = 1 \ldots 4 \quad \widehat{\mathbf{a}_i} = \underset{\mathbf{a}_i}{\operatorname{argmax}} P(A_i = \mathbf{a}_i \mid Id = id, \mathcal{P}) \,, \tag{2}$$

using four Ferns classifiers specific to the keypoint of identity Id.

ALGO1 will be evaluated in Section 6. Before that, we present our second method and their common second stage.

4 ALGO2: A Classifier Favoring Real-Time Learning and Robustness

ALGO1 was designed for run-time speed, and it requires a slow training phase: It takes about 1 second to ALGO1 to learn one keypoint, and this makes it not suitable for online applications like SLAM. We therefore propose a second method, which is slower at run-time but can learn new keypoints much faster. It is also more accurate.

4.1 Finding the Keypoint's Identity and Pose

As depicted by Fig. 5, our starting idea to estimate the keypoint's identity and pose is to first build a set of "mean patches". Each mean patch is computed as the average of the

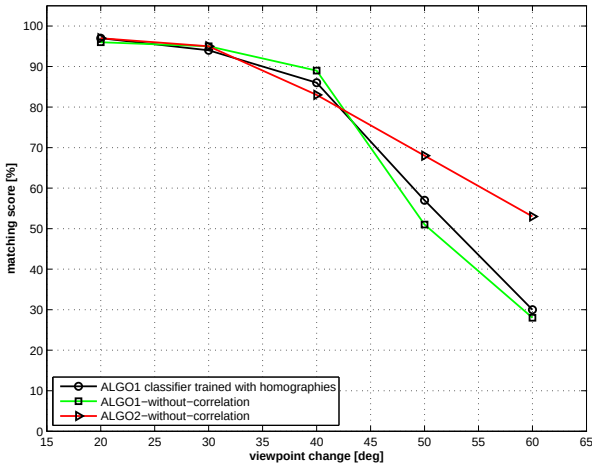


Fig. 4 We tried to use in ALGO1 the homography discretization used in ALGO2. As the graph shows, it does not result in any improvement in terms of matching score compared to the discretization described in Fig. 2. Since it requires more computation time and memory because the number of discrete homographies is larger, we used the method of Fig. 2 for ALGO1. The experiment was performed on the standard Graffiti test set [19].

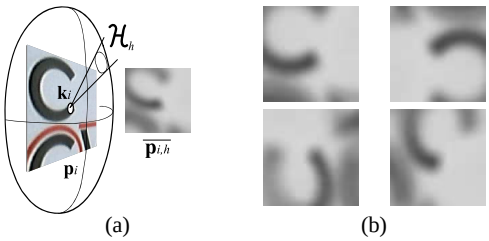


Fig. 5 The ALGO2 descriptor. (a) For a feature point $\mathbf{k}_i$, this descriptor is made of a set of mean patches $\{\overline{\mathbf{p}_{i,h}}\}$, each computed for a small range of poses $\mathcal{H}_h$ from a reference patch $\mathbf{p}_i$ centered on the feature point $\mathbf{k}_i$. (b) Some other mean patches for the same feature point. The examples shown here are full resolution patches for visibility, in practice we use downscaled patches.

keypoint appearance when the pose varies in the neighborhood of a reference pose. Then, we can use nearest-neighbor classification: We assign to an incoming keypoint the pose of the most similar mean patch as a first estimate of its pose.

Of course, computing a single mean patch over the full range of poses would result in a blurred irrelevant patch. Because we compute these mean patches over only a small range of poses, they are meaningful and allow for reliable recognition. Another advantage is that they are robust to image noise and blur. As the mean patches in Fig. 5 look like blurred image patches, one may wonder if using a uniform blur on warped patches would be enough. The answer is no: As Fig. 6 shows, using mean patches substantially improves the matching rate by about 20% compared to using blurred warped patches.

Compared to standard approaches [19], we do not have to extract an estimate of the keypoint's pose, nor compute a

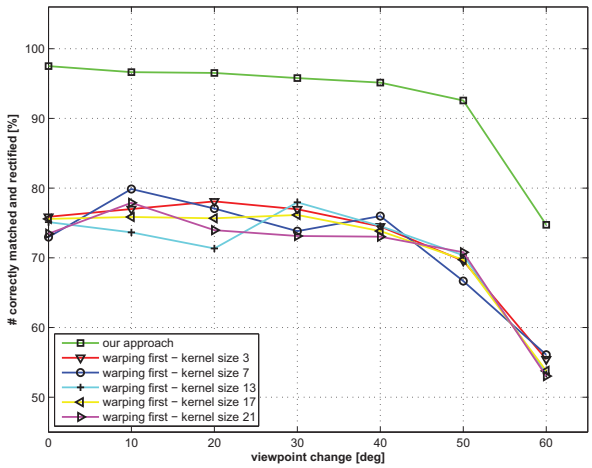


Fig. 6 Computing a set of mean patches clearly outperforms simple blurring of a set of warped patches. Different Gaussian smoothing kernels were tried and we show that our approach improves the matching rate constantly of about 20%.

descriptor for the incoming points, and that makes the approach faster at runtime. The set of means that characterizes a keypoint in the database can be seen as a descriptor, which we refer to as a "one-way descriptor" since it does not have to be computed for the new points. This approach increases the number of vectors that have to be stored in the database, but fortunately, efficient methods exist for nearest-neighbor search in large databases of high-dimensional vectors [4].

More formally, given a keypoint $\mathbf{k}$ in a reference image, we compute a set of mean patches $\{\overline{\mathbf{p}_h}\}$ where each mean $\overline{\mathbf{p}_h}$ can be expressed as

$$\overline{\mathbf{p}_h} = \int_{H \in \mathcal{H}_h} \mathbf{w}(\mathcal{P}^*, H) p(H) dH \qquad (3)$$

where

- H represents a pose, in our case a homography,
- $\mathcal{P}^*$ is the *reference patch*, the image patch centered on the keypoint $\mathbf{k}$ in the reference image. To be robust to light changes, the pixel intensities in $\mathcal{P}^*$ are normalized so their sum is equal to 0, and their standard deviation to 1,
- $\mathbf{w}(\mathcal{P}, H)$ returns the patch $\mathcal{P}$ seen under pose H,
- $p(H)$ is the probability that the keypoint will be seen under pose H, and
- $\mathcal{H}_h$ is a range of poses, as represented in Fig. 5. The $\mathcal{H}_h$'s are defined so that they cover small variations around a fixed pose H_h but together they span the set of possible poses $\bigcup_h \mathcal{H}_h$.

In practice, the integrals in Eq. (3) are replaced by finite sums, the distribution over the transformations H is assumed

uniform and the expression of $\overline{\mathbf{p}_h}$ becomes

$$\overline{\mathbf{p}_h} = \frac{1}{N}\sum_{j=1}^{N} \mathbf{w}(\mathcal{P}^*, H_{h,j}) \tag{4}$$

where the $H_{h,j}$ are N poses sampled from $\mathcal{H}_h$.

Once the means $\overline{\mathbf{p}_h}$ are computed, it is easy to match incoming keypoints against the database, and get a coarse pose. Given the normalized patch $\mathbf{p}$ centered on such an incoming point with assumed identity $\widehat{id}$, its coarse pose $\widehat{\mathbf{H}}_{i=\widehat{id},h=\hat{h}}$ indexed by $\hat{h}$ is obtained by finding:

$$\widehat{h} = \underset{i=\widehat{id},h}{\operatorname{argmax}}\, \mathbf{p}^\top \cdot \overline{\mathbf{p}_h}\,. \tag{5}$$

However, computing the $\overline{\mathbf{p}_h}$ using Eq. (4) is very inefficient because it would require the generation of too many samples $\mathbf{w}(\mathcal{P}^*, H_{h,j})$. In practice, to reach decent results, we have to use at least 300 samples, and this takes about 1.1 seconds to generate [1], which was not acceptable for interactive applications. We show below that the mean patches can actually be computed very quickly, independently of the number of samples used.

4.2 Fast Computation of the Mean Patches

We show here that we can move most of the computation cost of the mean patches to an offline stage, so that computing the mean patches at runtime can be done very efficiently. To this end, we first approximate the reference patch $\mathcal{P}^*$ as:

$$\mathcal{P}^* \approx \overline{\mathcal{V}} + \sum_{l=1}^{L} \alpha_l \mathcal{V}_l \tag{6}$$

where $\overline{\mathcal{V}}$ and the $\mathcal{V}_l$'s are respectively the mean and the L first principal components of a large set of image patches centered on keypoints. The α_l are therefore the coordinates of $\mathcal{P}^*$ in this eigenspace, and can be computed as $\alpha_l = \mathcal{V}_l^\top \cdot \mathcal{P}^*$.

Computing $\overline{\mathcal{V}}$ and the $\mathcal{V}_i$'s takes time but this can be done offline. Because we consider normalized patches, the mean $\overline{\mathcal{V}}$ is equal to 0, and the expression of $\overline{\mathbf{p}_h}$ from Eq.(3) becomes

$$\overline{\mathbf{p}_h} \approx \frac{1}{N}\sum_{j} \mathbf{w}(\sum_{l=1}^{L} \alpha_l \mathcal{V}_l, H_{j,h})\,. \tag{7}$$

This expression can be simplified by using the fact that the warping function $\mathbf{w}(\cdot, H)$ is a linear function: Warping is mostly a permutation of the pixel intensities between the original patch and the patch after warping, and therefore a linear transformation. This fact was used for example in [7] for transformation-invariant data modeling. In our case, because we use perspective transformations, parts that are not visible in the original one could appear in the generated patch. To solve this issue, we simply take the original patch larger than the warped patch, and large enough so that there are never parts in the warped patch which were not present in the original patch. The function $\mathbf{w}(., H)$ then becomes a permutation followed by a projection, and this composition remains a linear transformation.

Thanks to this property, Eq.(7) simplifies easily:

$$\overline{\mathbf{p}_h} \approx \frac{1}{N}\sum_{j=1}^{N} \mathbf{w}(\sum_{l=1}^{L} \alpha_l \mathcal{V}_l, H_{j,h}) \tag{8}$$

$$= \frac{1}{N}\sum_{j=1}^{N} \left(\sum_{l=1}^{L} \alpha_l \mathbf{w}(\mathcal{V}_l, H_{j,h})\right) \tag{9}$$

$$= \sum_{l=1}^{L} \frac{\alpha_l}{N} \sum_{j=1}^{N} \mathbf{w}(\mathcal{V}_l, H_{j,h}) \tag{10}$$

$$= \sum_{l=1}^{L} \alpha_l \overline{\mathbf{v}_{l,h}} \tag{11}$$

where the $\overline{\mathbf{v}_{l,h}}$'s are patches obtained by warping the eigenvectors $\mathcal{V}_l$ under poses in $\mathcal{H}_h$:

$$\overline{\mathbf{v}_{l,h}} = \frac{1}{N}\sum_{j=1}^{N} \mathbf{w}(\mathcal{V}_l, H_{j,h})\,. \tag{12}$$

Like the $\mathcal{V}_l$, the $\overline{\mathbf{v}_{l,h}}$'s patches can be computed offline. The number of samples N therefore does not matter for the runtime computations, and we can use a very large number.

In summary, when we have to insert a new keypoint in the database, we simply have to project it in the eigenspace, and compute its associated mean patches by linear combinations. The complete process can be written in matrix form:

$$\boldsymbol{\alpha} = \mathbf{P}_{\text{PCA}}\mathcal{P}^*\,, \text{ and} \tag{13}$$

$$\forall h \quad \overline{\mathbf{p}_h} = \mathbf{V}_h \boldsymbol{\alpha}\,, \tag{14}$$

where $\mathcal{P}^*$ is the reference patch seen as a vector, $\mathbf{P}_{\text{PCA}}$ the projection matrix into the eigenspace, $\boldsymbol{\alpha}$ the vector of the α_l coefficients, and the $\mathbf{V}_h$'s are matrices. The rows of $\mathbf{P}_{\text{PCA}}$ are the $\mathcal{V}_l$ vectors, and the columns of the $\mathbf{V}_h$'s matrices are the $\overline{\mathbf{v}_{l,h}}$ vectors. This approach saves significant computation time with respect to the naive way to compute Eq.(7).

To speed-up computation even further, mostly in the evaluation of the similarity between an incoming patch $\mathbf{p}$ and a mean patch $\overline{\mathbf{p}_h}$ as in Eq. (5), we downsample these patches. We keep the reference patch $\mathcal{P}^*$ and the eigenvectors $\mathcal{V}_l$ at the original resolution to avoid losing important details. Downscaling is applied only on the result of the sum in Eq. (12), which is performed off-line anyway.

[1] All times given in this paper were reached on a on a standard notebook (Intel(R) Centrino Core(TM)2 Duo with 2.6GHz and 3GB RAM and an NVIDIA quadro FX3600M with 512MB).

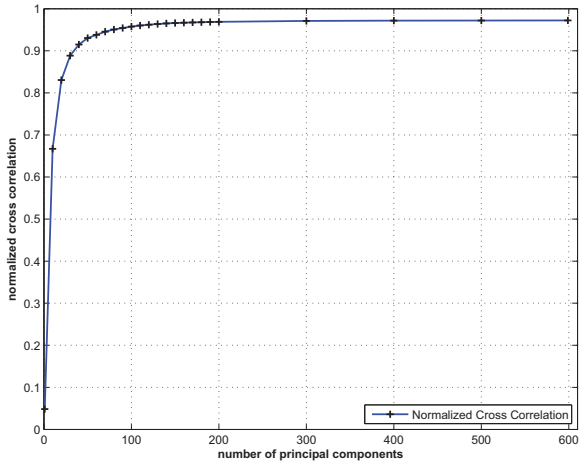


Fig. 7 Normalized cross-correlation between approximated mean patches and their exact computation as a function of the number of principal components. The values are averaged over 100 patches from the Graffiti image set. In this experiment, the patches are 120×120 pixels but we need only a small percentage of the principal components to get a good approximation.

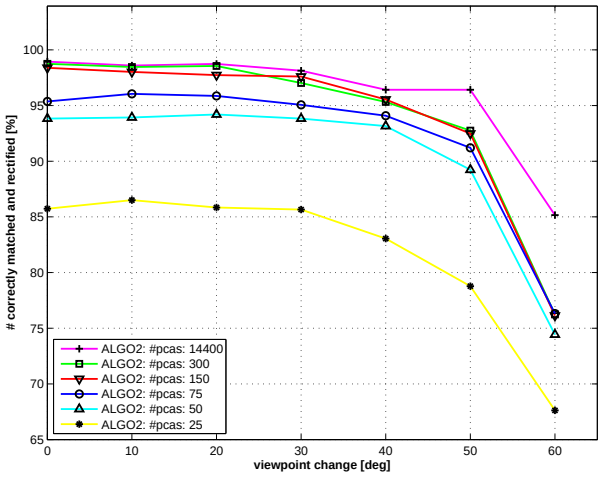


Fig. 8 Recognition rates as a function of the viewpoint angle for different number of principal components. The values are averaged over 100 patches from the Graffiti image set. We use synthesized images to generate more views than the original Graffiti image sequence. In this experiment, the patches are 120×120 pixels but using only 150 principal components over 14400 gives results comparable to the full method up to 40 degrees and is more than 70 times faster.

To handle scale efficiently, we compute the mean patches only on one scale level. Then, the evaluation of the similarity in Eq. (5) is done several times for each mean patch with different versions of the incoming patch $\mathbf{p}$, each version extracted at a different scale.

In practice, as shown in Fig. 7, we can keep only a small percentage of the first principal components and still get a good approximation of the mean patches. The graph of Fig. 8 shows that using only $L = 150$ principal components and 192 mean patches over 3 scales—giving a total of 576 mean patches—already gives reasonably good results. The computation time is then 15 milliseconds for patches downsampled from 71×71 to 12×12 including the computation of $\boldsymbol{\alpha}$ while using directly Eq. (4) takes 1.1 seconds. Using the GPU to compute the matrix form expressions in Eqs. (13) and (14), we can reduce the processing time even further to only 5.5 milliseconds [1].

4.3 Discretizing the Pose Space for ALGO2

The quantized pose space used for ALGO1 represented in Fig. 2 was designed to keep the number of discrete homographies small, because a finer discretization does not improve the matching score while slowing down the recognition and increasing the required memory. In ALGO2 however, we can afford a finer discretization and that results in better recognition rate.

This discretization is done based on the formula:

$$\mathbf{H} = \mathbf{K} \left(\Delta\mathbf{R} + \frac{\delta\mathbf{t} \cdot \mathbf{n}^\top}{d} \right) \mathbf{K}^{-1} , \qquad (15)$$

which is the expression of the homography $\mathbf{H}$ relating two views of a 3–D plane, where $\mathbf{K}$ is the matrix of the camera internal parameters, $[\mathbf{n}^\top, \ d]^\top$ the parameters of the plane in the first view, and $\Delta\mathbf{R}$ and $\delta\mathbf{t}$ the camera displacement between the two views. For simplification, we assume we have a frontal view of the reference patches.

We first tried discretizing the motion between the views by simply discretizing the rotation angles around the three axes. However, for the nearest-neighbor classification to work well, it must be initialized as close as possible to the correct solution, and we provide a better solution. As shown by the left image of Fig. 9, we found that the vertices of (almost) regular polyhedrons provide a more regular sampling that is useful to discretize the angle the second view in Eq. (15) makes with the patch plane.

However, there exists only a few convex regular polyhedrons —the Platonic solids— with the icosahedron the one with the largest number of vertices, 12. As the right image of Fig. 9 illustrates, we obtain a finer sampling by recursively substituting each triangle into four almost equilateral triangles. The vertices of the created polyhedron give us the two out-of-plane rotation angles for the sampled pose, that is around the x- and y-axes of Fig. 9. We discretize the in-plane rotation angle to cover the 360° range with 10° steps.

5 Pose Refinement and Final Check

Having the output of the first stage of ALGO1 or ALGO2, the keypoint's identity and approximate pose, we want to compute a better estimate of the pose, in the form of a ho-

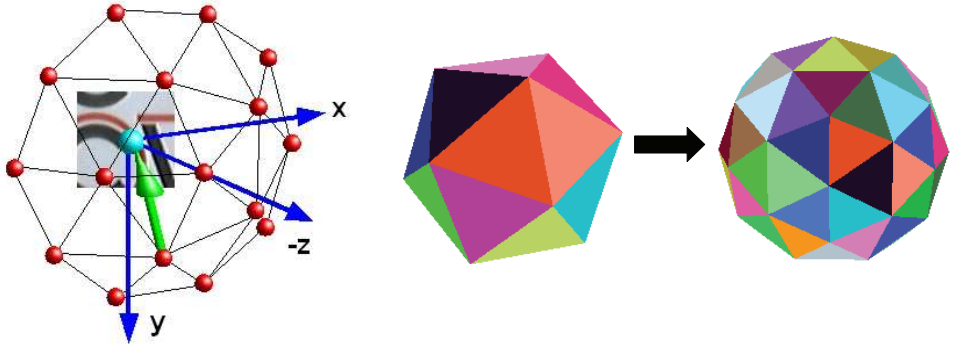


Fig. 9 Pose space sampling using almost regular polyhedrons. Left: The red dots represent the vertices of an (almost) regular polyhedron generated by our recursive decomposition and centered on a planar patch. The sampled directions of views are given by vectors starting from one of the vertices and pointing toward the patch center. The green arrow is an example of such a vector. Right: The initial icosahedron and the result of the first triangle substitution.

mography, without quantization anymore. This refinement is based on linear regression, and we show how the linear predictors can be computed incrementally and how we can improve the training speed. The refinement is followed by a final check step, to suppress keypoints that were incorrectly recognized.

5.1 Linear Prediction for Refinement

The homography $\widehat{\mathbf{H}}$ computed in the first stage is an initial estimate of the true homography $\overline{\mathbf{H}}$. We use the method presented in [13] and based on linear predictors to obtain the parameters $\widetilde{\mathbf{x}}$ of a corrective homography:

$$\widetilde{\mathbf{x}} = \mathbf{B}\left(\mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}}) - \mathbf{p}^*\right) , \tag{16}$$

where

- $\mathbf{B}$ is the matrix of our linear predictor, and depends on the retrieved patch identity $\widehat{id}$;
- $\mathcal{P}$ is the patch in the incoming image, centered on the keypoint to recognize;
- $\mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}})$ is the patch $\mathbf{p}$ warped by the current estimate $\widehat{\mathbf{H}}$ and downscaled for efficiency. Note that we do not actually warp the patch, we simply warp back the sampled pixel locations;
- $\mathbf{p}^*$ is the reference patch $\mathcal{P}^*$ after downscaling. $\mathcal{P}^*$ is the image patch centered on the keypoint $\widehat{id}$ in a reference image as in Section 4.

This equation gives us the parameters $\widetilde{\mathbf{x}}$ of the incremental homography that updates $\widehat{\mathbf{H}}$ to produce a better estimate of the true homography $\overline{\mathbf{H}}$:

$$\widehat{\mathbf{H}} \longleftarrow \widehat{\mathbf{H}} \circ \mathbf{H}(\widetilde{\mathbf{x}}) . \tag{17}$$

For more accuracy, we iterate Eqs. (16) and (17) using a series of linear predictors $\mathbf{B}_i$, each matrix being dedicated to smaller errors than its predecessor: Applying successively these matrices remains fast and gives a more accurate estimate than with a single level. In order to do an ultimate refinement the ESM algorithm [5] can be applied.

In practice, our vectors $\mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}})$ and $\mathbf{p}^*$ contain the intensities at locations sampled on a regular grid of 13×13 over image patches of size 75×75 pixels, and we normalize them to be robust to light changes. We parametrize the homographies by the 2D locations of the patch four corners since this parametrization is proved to be more stable than others in [1]. In practice, for one patch we train four to ten [2] matrices $\mathbf{B}$ with different ranges of variation from coarse to fine, using downscaled patches of $13 \times 13 = 169$ pixels and 300 to 5000 [2] training samples.

For online applications, the $\mathbf{B}$'s matrices must be computed for each new keypoint inserted in the database at runtime. Thus, learning the $\mathbf{B}_i$s which consists in computing a set of couples made of small random transformations H_s and the corresponding warped patches $\mathbf{w}(\mathcal{P}, H_s^{-1})$ as discussed below must be fast enough to fulfill the real-time constraints. In order to do so we precompute the transformations H_s and the warped pixel locations in order to obtain very quickly the $\mathbf{w}(\mathcal{P}, H_s^{-1})$ patches at run-time for an arbitrary incoming feature point. The whole process thus can be speed up to 29 milliseconds [1] using 300 samples and four $\mathbf{B}$ matrices.

5.2 Incrementally Learning the Linear Predictor

For some applications it is desirable to improve the tracking by performing online learning. Since the classification steps in ALGO1 as well as in ALGO2 can easily be extended to do online learning, we only have to concentrate on the linear predictors.

[2] Depending on the application. Using more $\mathbf{B}_i$ matrices improves the accuracy but the computation time for this step increases linearly with the number of matrices.

The linear predictor $\mathbf{B}$ in Eq. (16) can be computed as the pseudo-inverse of the analytically derived Jacobian matrix of a correlation measure [2,5]. However, the hyperplane approximation [13] computed from several examples yields a much larger region of convergence. The matrix $\mathbf{B}$ is then computed as:

$$\mathbf{B} = \mathbf{X}\mathbf{D}^\top \left(\mathbf{D}\mathbf{D}^\top\right)^{-1} , \tag{18}$$

where $\mathbf{X}$ is a matrix made of $\mathbf{x}_i$ column vectors, and $\mathbf{D}$ a matrix made of column vectors $\mathbf{d}_i$. Each vector $\mathbf{d}_i$ is the difference between the reference patch $\mathbf{p}^*$ and the same patch after warping by the homography parametrized by $\mathbf{x}_i$: $\mathbf{d}_i = \mathbf{w}(\mathbf{p}, \mathbf{H}(\mathbf{x}_i)) - \mathbf{p}^*$.

Eq. (18) requires all the couples $(\mathbf{x}_i, \mathbf{d}_i)$ to be simultaneously available. If it is applied directly, this prevents incremental learning but this can be fixed. Suppose that the matrix $\mathbf{B} = \mathbf{B}_n$ is already computed for n examples, and then a new example $(\mathbf{x}_{n+1}, \mathbf{d}_{n+1})$ becomes available. We want to update the matrix $\mathbf{B}$ into the matrix $\mathbf{B}_{n+1}$ that takes into account all the $n+1$ examples. Let us introduce the matrices $\mathbf{Y}_n = \mathbf{X}_n\mathbf{D}_n^\top$ and $\mathbf{Z}_n = \mathbf{D}_n\mathbf{D}_n^\top$. We then have:

$$\begin{aligned}
\mathbf{B}_{n+1} &= \mathbf{Y}_{n+1}\mathbf{Z}_{n+1}^{-1} \\
&= \mathbf{X}_{n+1}\mathbf{D}_{n+1}^\top \left(\mathbf{D}_{n+1}\mathbf{D}_{n+1}^\top\right)^{-1} \\
&= [\mathbf{X}_n|\mathbf{x}_{n+1}][\mathbf{D}_n|\mathbf{d}_{n+1}]^\top \left([\mathbf{D}_n|\mathbf{d}_{n+1}][\mathbf{D}_n|\mathbf{d}_{n+1}]^\top\right)^{-1} \\
&= \left(\mathbf{X}_n\mathbf{D}_n^\top + \mathbf{x}_{n+1}\mathbf{d}_{n+1}^\top\right)\left(\mathbf{D}_n\mathbf{D}_n^\top + \mathbf{d}_{n+1}\mathbf{d}_{n+1}^\top\right)^{-1} \\
&= \left(\mathbf{Y}_n + \mathbf{x}_{n+1}\mathbf{d}_{n+1}^\top\right)\left(\mathbf{Z}_n + \mathbf{d}_{n+1}\mathbf{d}_{n+1}^\top\right)^{-1}
\end{aligned} \tag{19}$$

where $\mathbf{x}_{n+1}$ and $\mathbf{d}_{n+1}$ are concatenated to $\mathbf{X}_n$ and $\mathbf{D}_n$ respectively to form $\mathbf{X}_{n+1}$ and $\mathbf{D}_{n+1}$. Thus, by only storing the *constant size* matrices $\mathbf{Y}_n$ and $\mathbf{Z}_n$ and updating them as:

$$\mathbf{Y}_{n+1} \longleftarrow \mathbf{Y}_n + \mathbf{x}_{n+1}\mathbf{d}_{n+1}^\top \tag{20}$$

$$\mathbf{Z}_{n+1} \longleftarrow \mathbf{Z}_n + \mathbf{d}_{n+1}\mathbf{d}_{n+1}^\top , \tag{21}$$

it becomes possible to incrementally learn the linear predictor without storing the previous examples, and allows for an arbitrary large number of examples.

Since the computation of $\mathbf{B}$ has to be done for many locations in each incoming image and $\mathbf{Z}_n$ is a large matrix in practice, we need to go one step further in order to avoid the computation of $\mathbf{Z}_n^{-1}$ at every iteration. We apply the Sherman-Morrison formula to $\mathbf{Z}_{n+1}^{-1}$ and we get:

$$\begin{aligned}
\mathbf{Z}_{n+1}^{-1} &= \left(\mathbf{Z}_n + \mathbf{d}_{n+1}\mathbf{d}_{n+1}^\top\right)^{-1} \\
&= \mathbf{Z}_n^{-1} - \frac{\mathbf{Z}_n^{-1}\mathbf{d}_{n+1}\mathbf{d}_{n+1}^\top\mathbf{Z}_n^{-1}}{1 + \mathbf{d}_{n+1}^\top\mathbf{Z}_n^{-1}\mathbf{d}_{n+1}} .
\end{aligned} \tag{22}$$

Therefore, if we store $\mathbf{Z}_n^{-1}$ instead of $\mathbf{Z}_n$ itself, and update it using Eq. (22), no matrix inversion is required anymore, and the computation of matrix $\mathbf{B}_{n+1}$ becomes very fast.

5.3 Correlation-based Hypothesis Selection and Verification

In ALGO2, for each possible keypoint identity i, we use the method explained above to estimate a fine homography $\widehat{\mathbf{H}}_{i,\text{final}}$. Thanks to the high accuracy of the retrieved transformation, we can select the correct pair of keypoint identity i and pose $\widehat{\mathbf{H}}_{i,\text{final}}$ based on the normalized cross-correlation between the reference patch $\mathcal{P}_i^*$ and the warped patch $\mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}}_{i,\text{final}})$ seen under pose $\widehat{\mathbf{H}}_{i,\text{final}}$. The selection is done by

$$\underset{i}{\operatorname{argmax}}\, \mathcal{P}_i^{*\top} \cdot \mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}}_{i,\text{final}}) , \tag{23}$$

In ALGO1, the keypoint identity i is directly provided by the Ferns classifier.

Finally, we use a threshold $\tau_{\text{NCC}} = 0.9$ in order to remove wrong matches:

$$\mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}}_{i,\text{final}})^\top \cdot \mathcal{P}_i^* > \tau_{\text{NCC}} , \tag{24}$$

Thus, each patch $\mathbf{w}(\mathcal{P}, \widehat{\mathbf{H}}_{i,\text{final}})$ that gives the maximum similarity score, which exceeds τ_{NCC} at the same time, yields an accepted match.

6 Experimental Validation

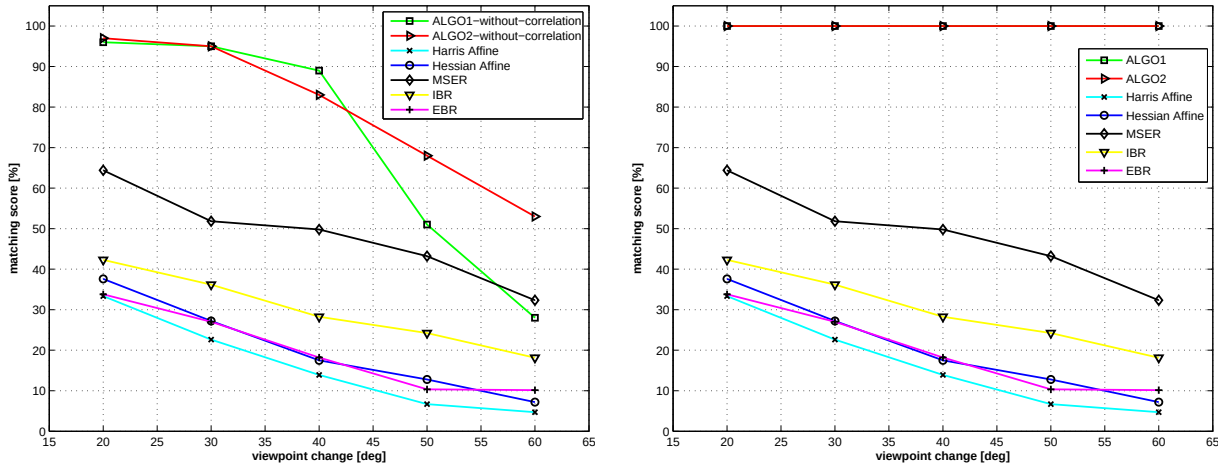
We compare here our approach against affine region detectors on the Graffiti image set from [19] towards robustness and accuracy. At the end of this section, we also evaluate the performance of our algorithms with respect to training time, running time and memory consumption. For each experiment we give a detailed discussion about the specific advantages of each of our two methods.

6.1 Evaluation on the Graffiti Image Set

We first built a database of the most stable 100 Harris keypoints from the first image of the Graffiti set. These keypoints were found by synthetically rendering the image under many random transformations, adding artificial image noise and extracting Harris keypoints. We then kept the 100 keypoints detected the most frequently.

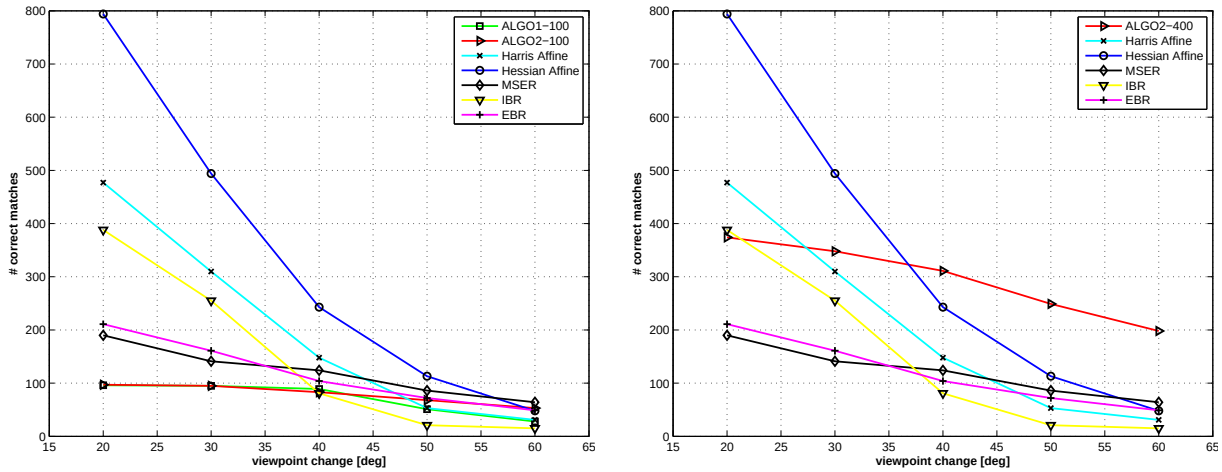
The Ferns classifiers in ALGO1 were trained with synthetic images as well, by scaling and rotating the first image for changes in viewpoint angle up to 65 degrees and adding noise. In the case of ALGO2 only the first image is needed. We then extracted Harris keypoints in the other images of the set, and run ALGO1 and ALGO2 to recognize them and estimate their poses.

We also run the different region detectors over the set images and matched the regions in the first image against the regions in the other images using the SIFT descriptor computed on the rectified regions.



(a) (b)

Fig. 10 Comparing the robustness of our methods and of affine region detectors on the Graffiti image set. **(a)**: Matching score as a function of the viewpoint angle. The curves of ALGO1 and ALGO2 plot our results with the correlation test of Section 5.3 disabled. Even then, our methods compare very favorably with the affine region detectors. **(b)**: Same with the correlation test turned on. No outlier is produced.



(a) (b)

Fig. 11 Comparing our method against affine region detectors on the Graffiti image set. **(a)** Number of correct matches for our approach and the affine region detectors. We trained our methods on 100 keypoints in the first image. **(b)** ALGO2 can manage more keypoints, and for this experiment we trained it on 400 keypoints. For the largest viewpoint we still obtain a matching rate of about 50%.

6.1.1 Robustness

In Fig. 10, we compare the matching scores for the different methods. The matching score is computed as the ratio between the number of correct matches and the smaller number of regions detected in one of the two images as defined in [19]. Two regions are said to be correctly matched if the overlap error is smaller than 40%. In our case, the regions are defined as the patch surfaces warped by the retrieved transformations

For a fair comparison, we first turned off our final check on the correlation since there is no equivalent for the affine regions in [19]. This yields the 'ALGO1/ALGO2 without Correlation' curves. Even then, our methods perform much better, at least up to an angle of 50° in the case of ALGO1. When we turn the final check on, not a single outlier is kept in this experiment. For completness, we also show the actual number of correct matches in Fig. 11(a). Note, that we can manually choose how many patches we initially want to track while affine region detectors can not.

Because the Ferns classifier consumes a large amount of memory that grows linearly with the number of classes, it is difficult to handle more than 100 keypoints with ALGO1. ALGO2 by contrast uses a nearest-neighbour classifier and can deal with more keypoints. We therefore performed in Fig. 11(b) the same kind of experiment as in Fig. 11(a), but with 400 keypoints for ALGO2. In that case, for the large

viewpoint angles, we get more correctly matches keypoints than regions with the affine region detectors.

ALGO2 is also more robust to scale and perspective distortions than ALGO1. In practice we found out that the limiting factor is by far the repeatability of the keypoint detector. However, once a keypoint is correctly detected, it is very frequently correctly matched at least by ALGO2.

6.1.2 2–D Accuracy

In Figs. 13(a)-(d), we compare the 2–D accuracy for the different methods. To create these graphs, we proceed as shown in Fig. 12. We first fit a square tangent to the normalized region, take into account the canonical orientation retrieved by SIFT and warp these squares back with the inverse transformation to get a quadrangle. To account for different scales, we proceed as in [19]: We normalize the reference patch and the back-warped quadrangle such that the size of the reference patch is the same for all the patches.

Two corresponding regions should overlap if one of them is warped using the ground truth homography. A perfect overlap for the affine regions cannot be expected since their detectors are unable to retrieve the full perspective. Since in SIFT several orientations were considered when ambiguity arises, we decided to keep the one that yields the most accurate correspondence. In the case of our method, the quadrangles are simply taken to be the patch borders after warping by the retrieved transformations.

Fig. 13(a) evaluates the error based on the overlap between the quadrangles and their corresponding warped versions. This overlap is between 90% and 100% for our methods, about 5-10% better than MSER and about 15-25% better for the other methods. Fig. 13(b) evaluates the error based on the distances between the quadrangle corners. Our methods also perform much better than the other methods. The error of the patch corner is less than two pixels in average for ALGO1 and slightly more for ALGO2. Figs. 10(c) and (d) show the same comparisons, this time when taking only the best 100 regions into account. The results are very similar.

6.2 3–D Pose Evaluation for Low-Textured Objects

In order to demonstrate the usefulness of our approach especially for low textured objects and for outdoor environments, we did two other quantitative experiments.

For the first experiment we run different methods to retrieve the camera pose using the outlet of Fig. 14 in a sequence of 398 real images. To obtain ground truth data we attached an artificial marker next to the outlet and tracked this marker. The marker itself was hidden in the reference image. We consider errors on the camera center larger than 50 units as not correctly matched. For clarity we do not display these false results. For our approaches we learned only one single patch on the outlet to track in order to emphasize the possibility to track an object with only a single patch. ALGO2 achieved a successful matching rate in over 98%, directly followed by ALGO1 with 97%.

For the affine region detectors we tried two different methods to estimate the pose of the outlet:

- Method A: For the first row of Fig. 14 we computed the pose from the 2–D locations of all the correctly matched affine regions. The correct matches were obtained by computing the overlap error between the regions that were matched by SIFT. In order to compute the overlap error we used the ground truth transformations. Note that this gives a strong advantage to the affine region detectors since the ground truth is usually not available. Each pair of regions was labeled as correctly matched if the overlap error was below 40%. The IBR detector obtained the best results with a 18% matching rate.
- Method B: For the second row, we used the shape of two matched affine regions in order to determine the current pose of the object. In order to obtain the missing degree of freedom, the orientation was obtained by determining the dominant orientation within the patch [15]. Since for each image several transformations are available due to several extracted affine regions, we finally took the transformation that corresponds best to the ground truth. The MSER and the Hessian-Affine detector perform best with a matching rate of 43% and 35%.

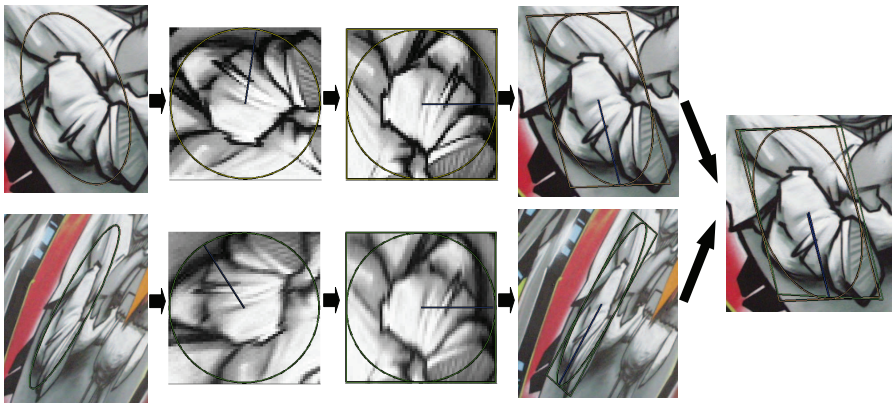
For the second experiment, shown in Fig. 15, we tracked a foot print in a snowy ground in a sequence of 453 images. The results are very similar to the first experiment's results. The success rate of our algorithms is around 88%. Again, Method A with the IBR detector performs best among the affine region detectors with a matching rate of 12%. All other detectors had a success rate of below 1%. For Method B, all affine region detectors performed around 5% except the EBR detector which had a matching rate below 1%.

6.3 Speed

We give below the computation times for training and runtime for both of our methods. All times given were obtained on a standard notebook [1]. Our implementations are written in C++ using the Intel OpenCV and IPP libraries.

6.3.1 Training

Table 1 shows the advantage of ALGO2 over ALGO1: It is much faster than ALGO1. When the GPU is used, learning time drops to 5.5 milliseconds, which is largely fast enough for frame rate learning, for SLAM applications for example. Computing the $\mathbf{B}$'s matrices for the refinement stage can be done in additional 29 ms on the CPU.



(a) (b) (c) (d) (e)

Fig. 12 Measuring the overlapping errors and the corners distances. (a) Two matched affine regions. (b) The same regions, after normalization by their affine transformations displayed with their canonical orientations. (c) Squares are fitted to the final normalized regions. (d) The squares are warped back into quadrangles in the original images. (e) The quadrangle of the second region is warped back with the ground truth homography and compare with the quadrangle of the first image. Ideally the two quadrangles should overlap. For comparison of different region detectors we normalize the reference region to a fixed size and scale the warped region correspondingly.

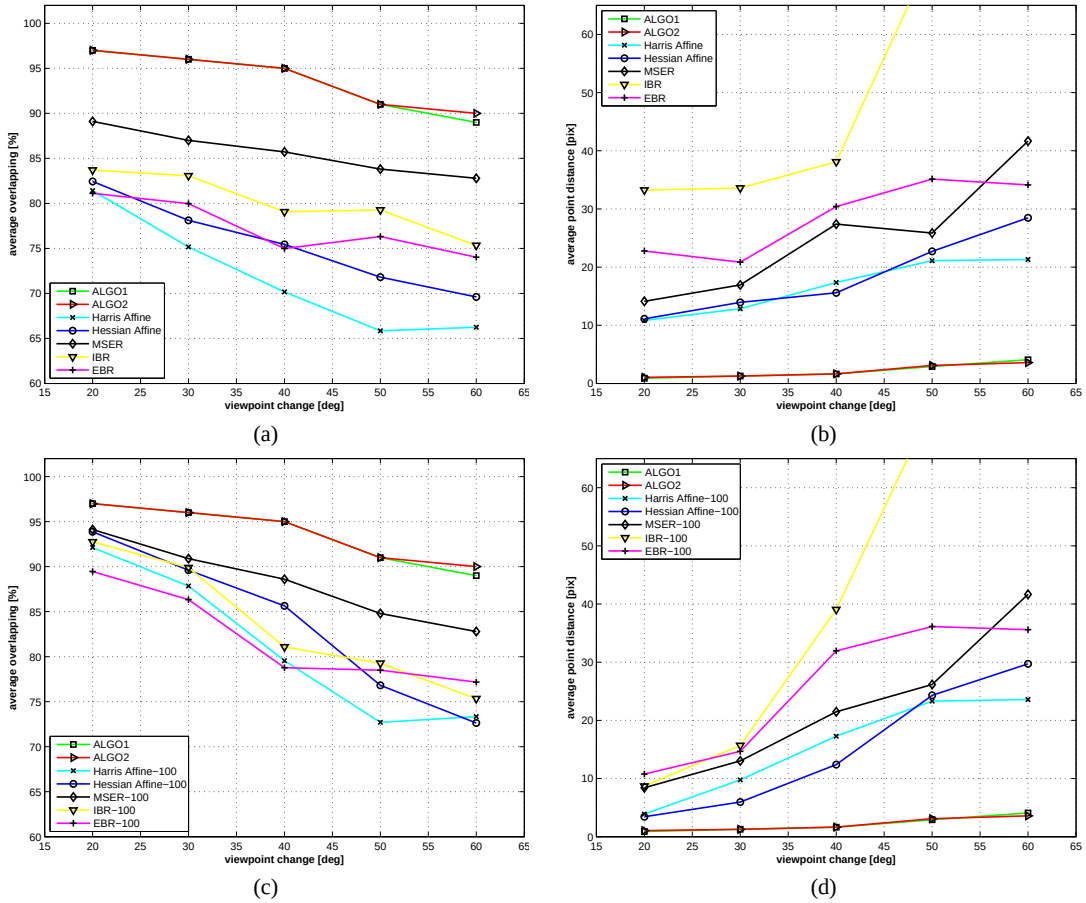


Fig. 13 Comparing the accuracy of our methods and of affine region detectors on the Graffiti image set. **(a)**: Average overlapping area of all correctly matched regions. Our method is very close to 100% and always more accurate than the other methods. **(b)**: Average sum of the distances from the ground truth for the corner points. Our method is also more accurate in that case. **(c)**,**(d)** Same experiments as **(a)**, **(b)** but with only the best 100 regions kept.

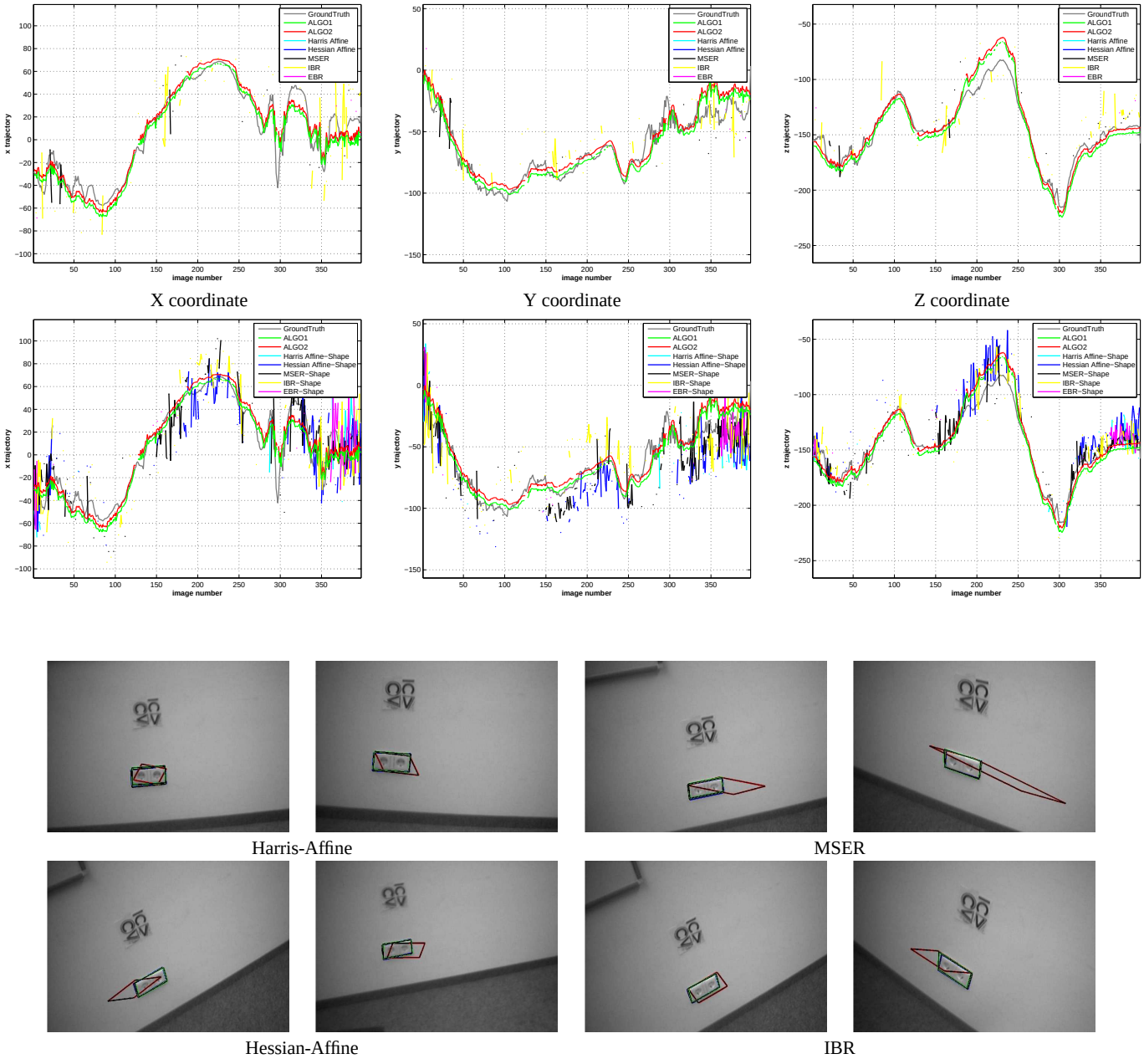


Fig. 14 Camera trajectories retrieved by different methods for a video sequence of the outlet of Fig. 22. For clarity we do not display results if the error on the camera center is larger than 50 units. **First row**: X, Y and Z coordinates of the camera center over the sequence in a coordinates system centered on the outlet. For the affine region detectors, the camera pose was retrieved using Method A as explained in Section 6.2. **Second row**: Same but using Method B. **Last two rows**: Typical results for different methods. the blue quadrangle is the ground truth, the green one was retrieved using ALGO2, the red one using one of the affine region detectors.

ALGO1 [11]	1.05 seconds
ALGO2 (CPU) [12]	15 miliseconds
ALGO2 (GPU) [12]	5.5 miliseconds

Table 1 Average learning time per feature for the first step in different approaches. ALGO2 is more than 70 times faster when the GPU is used.

6.3.2 Run-Time

Our current implementation of ALGO2 runs at about 10 frames per second using 10 keypoints in the database and 70 candidate keypoints. A better run-time performance is achieved with ALGO1: Our implementation runs at about 10 frames per second using a database of 50 keypoints and 400 candidate keypoints. Note that for ALGO1 the run-time is almost

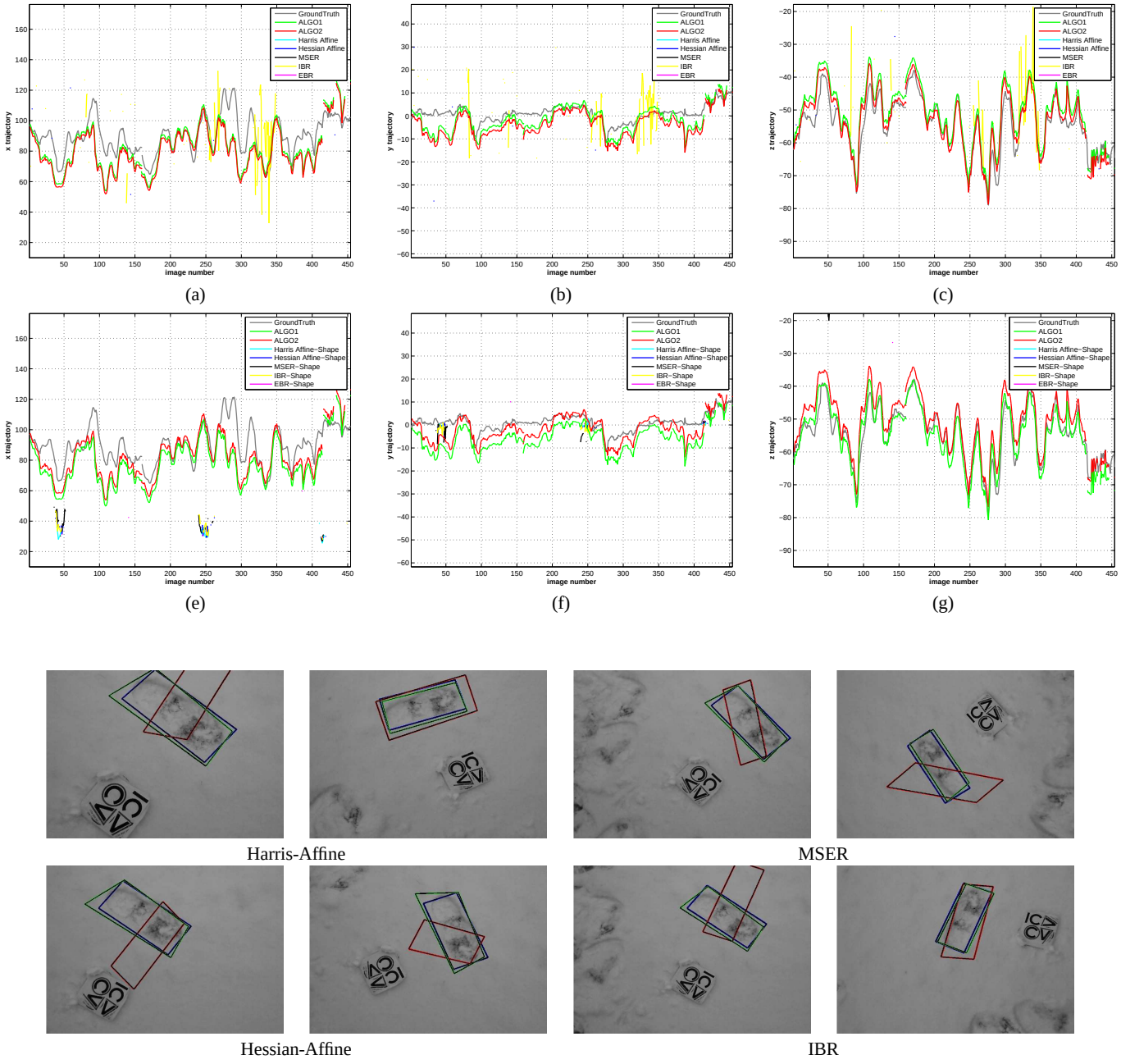


Fig. 15 Camera trajectories retrieved by different methods for a video sequence of a footprint in the snow. For clarity we do not display results if the error on the camera center is larger than 50 units. **First row**: X, Y and Z coordinates of the camera center over the sequence in a coordinates system centered on the outlet. For the affine region detectors, the camera pose was retrieved using Method A as explained in Section 6.2. **Second row**: Same but using Method B. **Last two rows**: Typical results for different methods. the blue quadrangle is the ground truth, the green one was retrieved using ALGO2, the red one using one of the affine region detectors.

constant with respect to the size of the database and only depends on the number of candidate keypoints. For ALGO2 the run-time is not only influenced by the number of candidate keypoints but also behaves linearly in the number of patches in the database. The single times for one patch in the database with respect to the number of candidate keypoints in the current image are given in Fig. 16. We do not use any special data structure for nearest neighbor search and using for example KD-trees [4] would speed it up. However, due to the methods robustness and accuracy, one detected keypoint is enough to detect the target object and to estimate its pose reliably. This can considerably speed up the processing time if the object is seen in the image and the result of only one extracted patch is enough to start a non-linear optimiza-

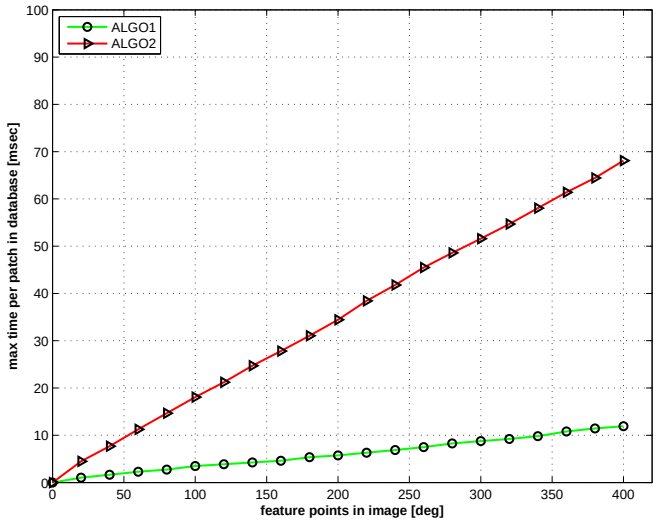


Fig. 16 We compare the maximal run-time per keypoint of both of our methods with respect to the number of keypoints extracted in the image. For ALGO1 we give two different run-times: The first one uses the same matching schema as ALGO2 which is more robust but slower. If we use the patch pre-classification described in Eq. 1 the run-time is decreased even more. Few hundreds of patches can be handled in real-time if the pre-classification is switched on.

tion process. Thus, the processing of all remaining keypoints in an image can be skipped as soon as one keypoint is extracted.

6.4 Memory

Typically, ALGO1 uses about 8 MB per keypoint, while ALGO2 uses only 350 KB. The actual amount depends on several parameters, but these values are representative. In particular the ratio between the two methods is typical: ALGO1 trades a large amount of memory for run-time speed.

7 Applications

7.1 Training Framework

Our methods can be trained using either a small set of training images or a video sequence. In the first case, we synthesize images by warping the original patches with random homographies and adding noise to train the classifiers and the linear predictors. A video sequence and a 3D model could also be used if available. In that case we proceed as proposed in [23]: The first image is registered manually and approximately. It is used to partially train the classifiers and linear predictors. Assuming a small interframe displacement in the training sequence, this is enough to recognize feature points in the next image, and register it. The process is iterated to process the whole sequence as shown in Fig. 17.

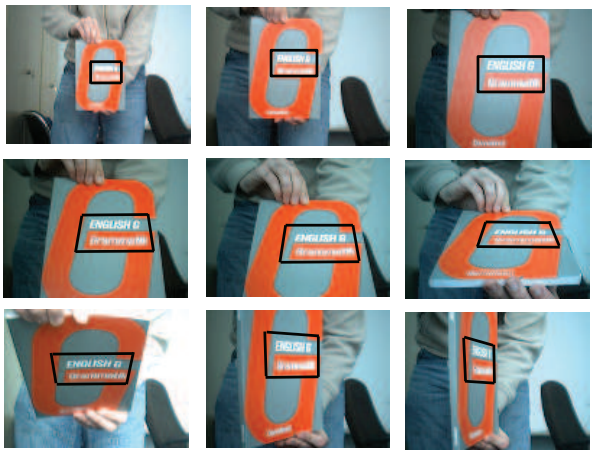


Fig. 17 Training framework. We incrementally train the classifiers and the linear predictors over the frames of a training sequence. To this end, the object is automatically registered in each incoming frame using the current state of these classifiers and linear predictors.

7.2 Examples

In Figs. 18, 19, 20, and 21, we apply ALGO1 to object detection and pose estimation application using a low-quality camera. ALGO1 is robust and accurate even in presence of drastic perspective changes, light changes, blur, occlusion, and deformations. For each of these objects we learned the patches from an initial frontal view. In Figs. 20 and 21 we used the template matching-based ESM algorithm [5] to refine the pose obtained from a single patch. As one can see, one extracted patch is already good enought to obtain the pose of the object reliably.

Several applications using ALGO2 are shown Figs. 22, 23, 24, and 25, respectively SLAM re-localisation using a single keypoint, poorly textured object detection, deformable object detection, and SLAM re-localisation in a room.

In Fig. 22 we learned the outlet from a frontal view with one patch and tracked it throughout 372 images as shown in the corresponding trajectory. While the outlet is a hard to track object we can still track it reliably despite of large scale and view point changes. Note, that SLAM usually uses apriori information about its pose in order to be fast while we do redetection in each frame in real-time.

In Fig. 23 we show ALGO2 working with poorly textured objects i.e. a telephone and a soldering station. Structured and weakly textured objects like them are often found in human environments and it is especially important to be able to interact with them. Although they do not exhibit very discriminative texture they still can be reliably recognized by our approach.

In Fig. 24 we tried our approach on matching deformable surfaces. We learned five patches on the book from an initial frontal view. Although the book is then strongly deformed

and we do not model the deformation within our recognition pipline, most of the learned patches are reliably recognized. The returned poses — represented by the visualized normals shown in blue — very often fit well to the deformation. We also found out that not the deformation but the specular reflection on the book cover is very often harder to overcome.

In Fig. 25 we show another example of a possible SLAM application. We walk around in an arbitrary office space and learn key landmarks which are reliably redetected in each successive frame if available. While scalability is currently still an issue and avoids the excessive use of patches, we see a possible application of our approach in learning few key landmarks which help to relocalize the SLAM system fast in difficult areas where is not much texture around and state-of-the-art relocalization approaches would fail.

8 Conclusion

We showed that including pose estimation within the recognition process considerably improves the robustness and the accuracy of the results of object detection, and this makes our approach highly desirable. Thanks to a two-step algorithm, it is possible to get matching sets that do usually contain no outliers. Even low-textured objects can therefore be well detected and their pose can be well estimated.

We showed in the paper that a Fern based classifier is able to recognize the keypoints in a very fast manner that allows to track several hundred patches very accurately in real-time. We also showed that the simultaneous estimation of keypoint identities and poses is more reliable but slower than the two separate steps undertaken consecutively. Finally, we showed how to build in real-time an one-way descriptor based on geometric blur that quickly, robustly and accurately estimates the pose of feature points and therefore is appropriate for applications where real-time learning is mandatory.

We demonstrated in various experiments the improved performance compared to previous state-of-the-art methods and demonstrated our approach on many applications including simple 3D tracking-by-detection, SLAM applications, low-textured object detection and deformable objects registration. However, many other applications could benefit from it, such as object recognition, image retrieval or robot localization.

Acknowledgements This project was granted partly by BMBF (AVILUS-plus: 01IM08002) and by the Bayrische Forschungsstiftung BFS.

References

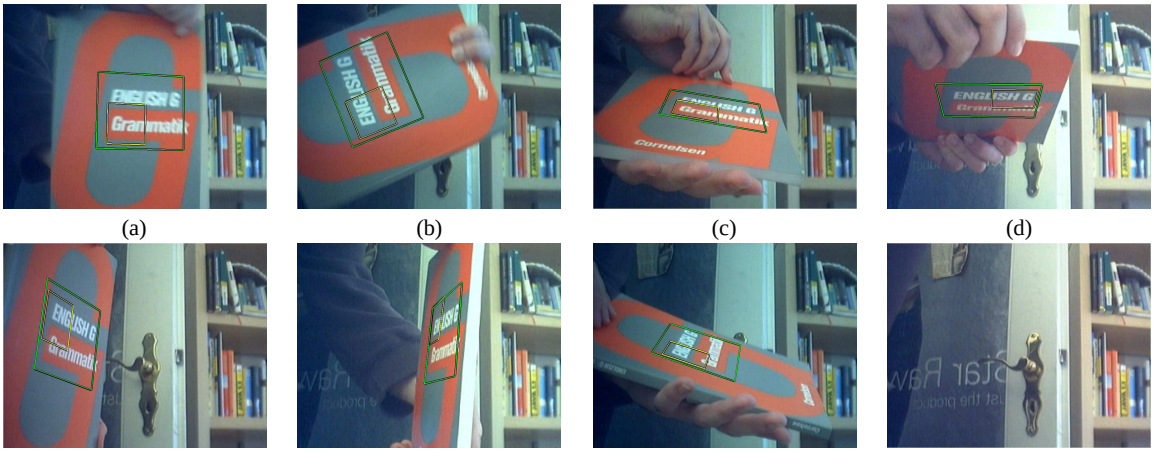
1. Baker, S., Datta, A., Kanade, T.: Parameterizing homographies. Tech. Rep. CMU-RI-TR-06-11, CMU (2006)
2. Baker, S., Matthews, I.: Lucas-kanade 20 years on: A unifying framework. IJCV **56**, 221–255 (2004)
3. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. In: European Conference on Computer Vision (2006)
4. Beis, J., Lowe, D.: Shape Indexing using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In: Conference on Computer Vision and Pattern Recognition, pp. 1000–1006. Puerto Rico (1997)
5. Benhimane, S., Malis, E.: Homography-based 2d visual tracking and servoing. IJRR **26**(7), 661–676 (2007)
6. Berg, A., Malik, J.: Geometric blur for template matching. In: Conference on Computer Vision and Pattern Recognition (2002)
7. B.J.Frey, Jojic, N.: Transformation invariant clustering using the em algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence (2003)
8. Chum, O., Matas, J.: Geometric Hashing with Local Affine Frames. In: Conference on Computer Vision and Pattern Recognition, pp. 879–884 (2006)
9. Goedeme, T., Tuytelaars, T., Van Gool, L.: Fast wide baseline matching for visual navigation. In: Conference on Computer Vision and Pattern Recognition (2004)
10. H. Grabner and H. Bischof: On-line boosting and vision. In: Conference on Computer Vision and Pattern Recognition (2006)
11. Hinterstoisser, S., Benhimane, S., Navab, N., Fua, P., Lepetit, V.: Online Learning of Patch Perspective Rectification for Efficient Object Detection. In: Conference on Computer Vision and Pattern Recognition (2008)
12. Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., Lepetit, V.: Real-Time Learning of Accurate Patch Rectification. In: Conference on Computer Vision and Pattern Recognition (2009)
13. Jurie, F., Dhome, M.: Hyperplane approximation for template matching. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(7), 996–100 (2002)
14. Lepetit, V., Lagger, P., Fua, P.: Randomized Trees for Real-Time Keypoint Recognition. In: Conference on Computer Vision and Pattern Recognition. San Diego, CA (2005)
15. Lowe, D.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision **20**(2), 91–110 (2004)
16. M. Grabner and H. Grabner. and H. Bischof: Learning features for tracking. In: Conference on Computer Vision and Pattern Recognition (2007)
17. Matas, J., Chum, O., Martin, U., Pajdla, T.: Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. In: British Machine Vision Conference, pp. 384–393. London, UK (2002)
18. Mikolajczyk, K., Schmid, C.: Scale and affine invariant interest point detectors. IJCV (2004)
19. Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Van Gool, L.: A comparison of affine region detectors. International Journal of Computer Vision **65**(1), 43–72 (2005)
20. Molton, N., Davison, A., Reid, I.: Locally Planar Patch Features for Real-Time Structure from Motion. In: British Machine Vision Conference (2004)
21. Obdržálek, Š., Matas, J.: Toward Category-Level Object Recognition, chap. 2, pp. 85–108. J. Ponce, M. Herbert, C. Schmid, and A. Zisserman (Editors). Springer-Verlag, Berlin Heidelberg, Germany (2006)
22. Ozuysal, M., Calonder, M., Lepetit, V., Fua, P.: Fast Keypoint Online Learning and Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (2009)
23. Ozuysal, M., Lepetit, V., Fleuret, F., Fua, P.: Feature Harvesting for Tracking-by-Detection. In: European Conference on Computer Vision (2006)
24. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: CVPR (2007)

(a) (b) (c) (d)

Fig. 18 Robustness of ALGO1 to deformation and occlusion. **(a)** Patches detected on the book in a frontal view. **(b)** Most of these patches are detected even under a strong deformation. **(c)** The book is half occluded but some patches can still be extracted. **(d)** The book is almost completely hidden but one patch is still correctly extracted. No outliers were produced.

(a) (b) (c) (d)

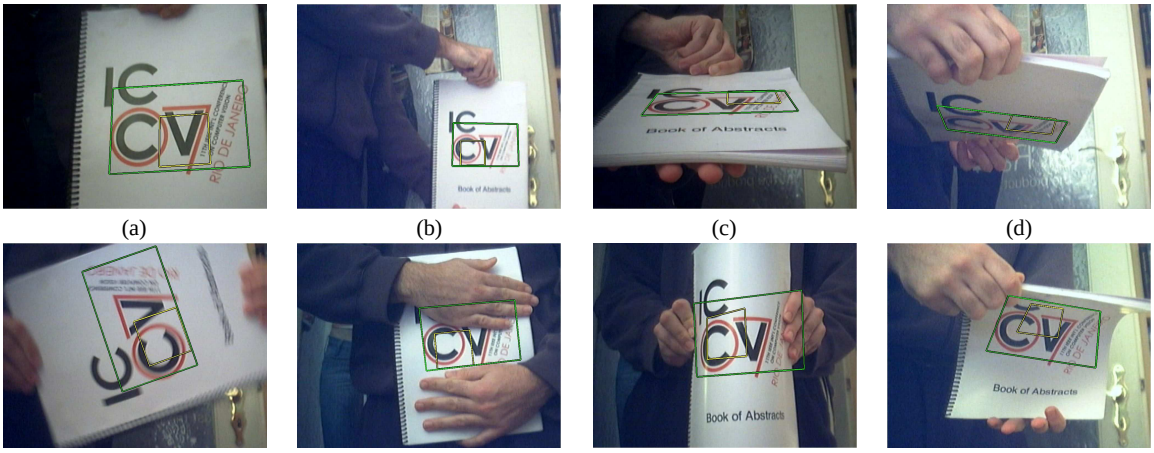
Fig. 19 Accuracy of ALGO1 of the retrieved transformation. For each of these images, we draw the borders of the book estimated from a single patch. This is made possible by the fact we estimate a full perspective transform instead of only an affine one.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 20 Some frames of a Tracking-by-Detection with ALGO1 sequence shot with a low-quality camera. **(a)-(g)** The book pose is retrieved in each frame independently at 10fps. The yellow quadrangle is the best patch obtained by ALGO1. The green quadrangle is the result of the ESM algorithm [5] initialized with the pose obtained from this patch. The retrieved pose is very accurate despite drastic perspective and intensities changes and blur. **(h)** When the book is not visible, our method does not produce a false positive.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 21 Another example of a Tracking-by-Detection sequence with ALGO1. The book pose is retrieved under **(b)** scale changes, **(c-d)** drastic perspective changes, **(e)** blur, **(f)** occlusion, and **(g-h)** deformations.

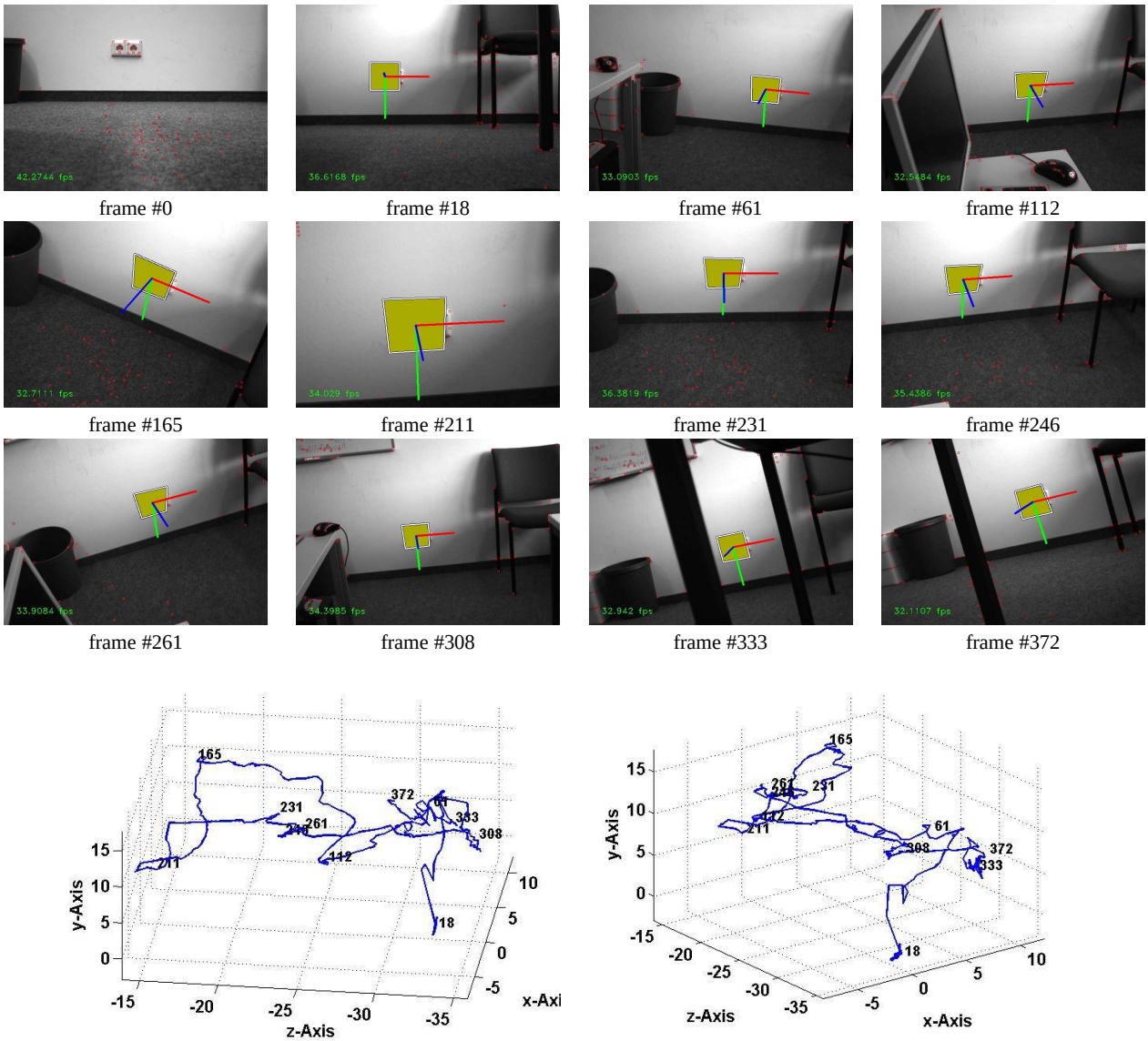


Fig. 22 Tracking an outlet with ALGO2. We can retrieve the camera trajectory through the scene despite very limited texture and large viewpoint changes. Since the patch is detected and its poses estimated in every frame independently, the method is very robust to fast motion and occlusion. The two graphs show the retrieved trajectory.

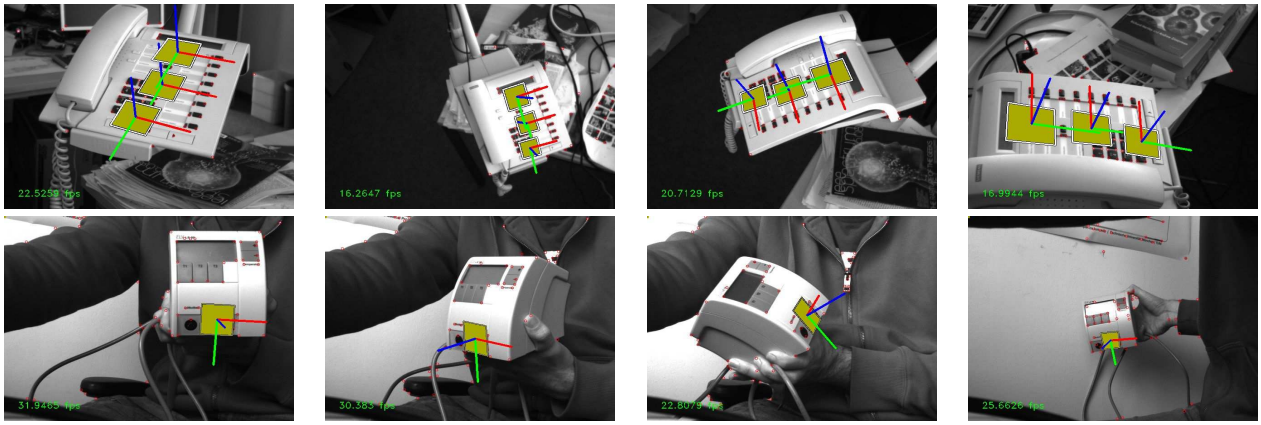


Fig. 23 Application to tracking-by-detection of poorly textured objects under large viewing changes with ALGO2.

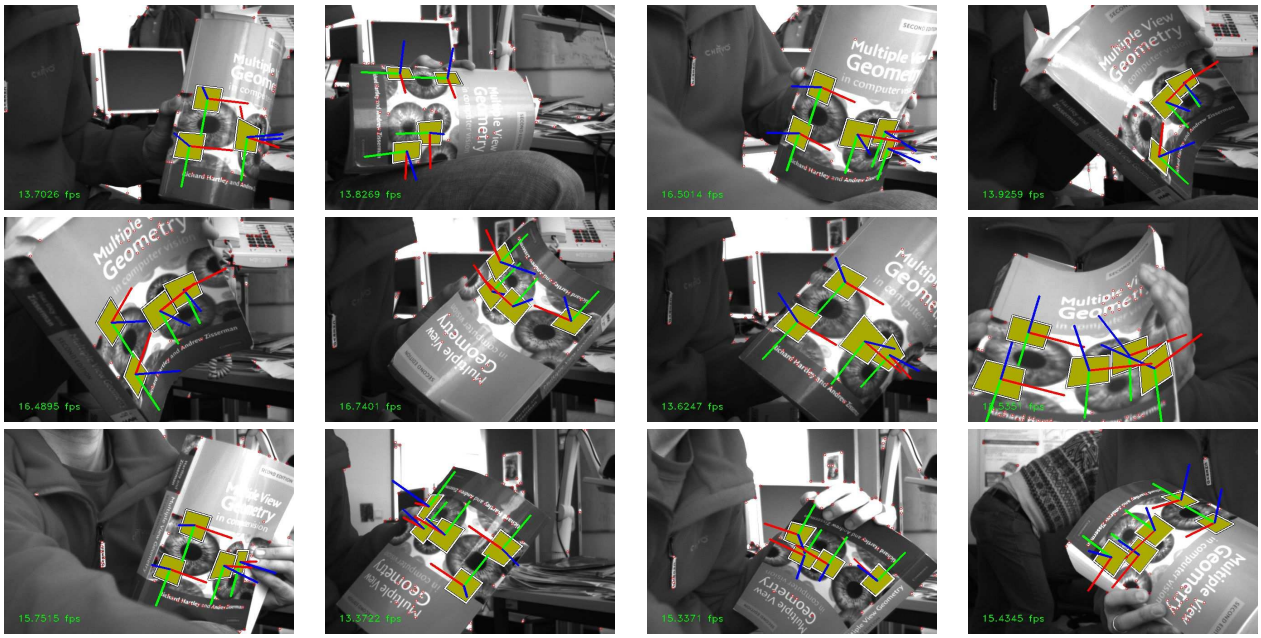


Fig. 24 Application to a deformable object with ALGO2. We can retrieve an accurate pose even under large deformations. While it is not done here, such cues would be very useful to constrain the 3D surface estimation.

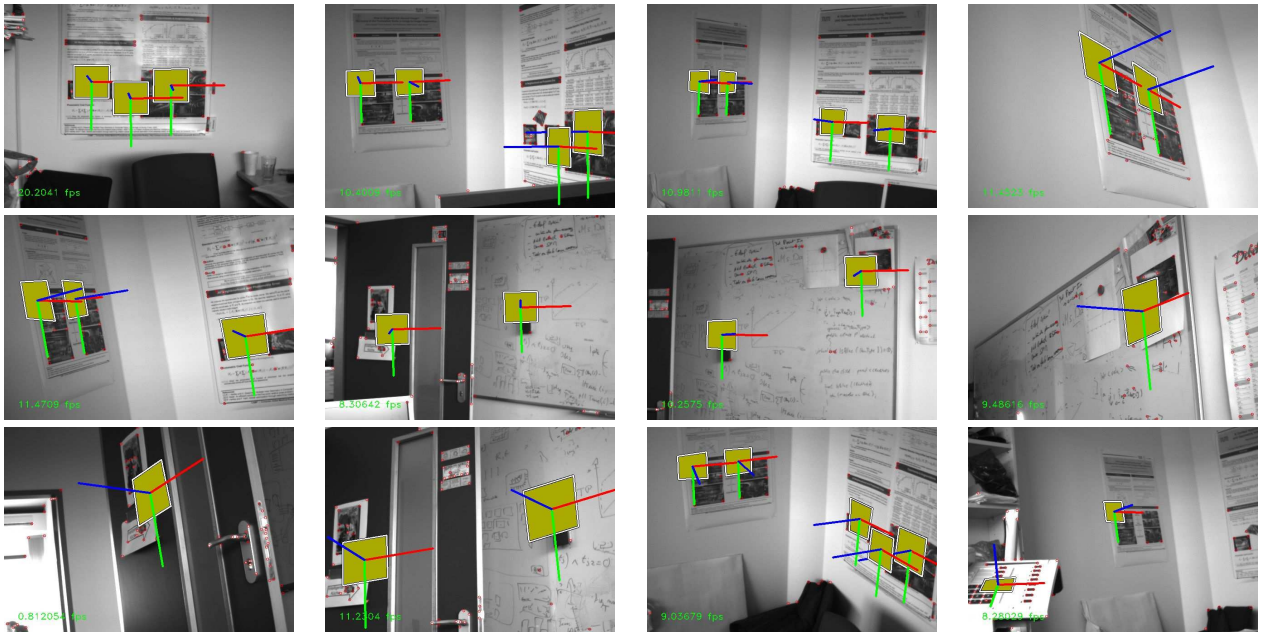


Fig. 25 Another example of SLAM re-localisation with ALGO2, using 8 different patches.

25. Rothganger, F., Lazebnik, S., Schmid, C., Ponce, J.: Object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. International Journal of Computer Vision **66**(3), 231–259 (2006)
26. Taylor, S., Drummond, T.: Multiple target localisation at over 100 fps. In: BMVC (2009)
27. Triggs, B.: Detecting Keypoints with Stable Position, Orientation and Scale under Illumination Changes. In: European Conference on Computer Vision (2004)
28. Williams, B., Klein, G., Reid, I.: Real-time slam relocalisation. In: International Conference on Computer Vision (2007)