Random Ferns for Keypoint Recognition, Image Matching and Tracking – Implementation and Experiments

Erik Derner, Tomáš Svoboda,
Karel Zimmermann

derneeri@fel.cvut.cz, svoboda@cmp.felk.cvut.cz,
zimmerk@cmp.felk.cvut.cz

CTU–CMP–2010–16

November 3, 2010

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/svoboda/Derner-TR-2010-16.pdf

The authors were supported by EC project FP7-ICT-247870 NIFTi
and by the Czech Science Foundation under project P103/10/1585.
Any opinions expressed in this paper do not necessarily reflect the
views of the European Community. The Community is not liable
for any use that may be made of the information contained herein.

Research Reports of CMP, Czech Technical University in Prague, No. 16, 2010

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Random Ferns for Keypoint Recognition, Image Matching and Tracking – Implementation and Experiments

Erik Derner, Tomáš Svoboda, Karel Zimmermann

November 3, 2010

Abstract

We have implemented a ferns-based classifier in MATLAB and applied it in image matching, object detection and tracking applications. We improve recognition performance, especially for scale changes. Additional modifications should help to overcome Harris corner detector inaccuracy and improve RANSAC homography estimation. We performed experiments with various real-world images and we show that our classifier is able to handle significant perspective changes.

Contents

1 Introduction

Image recognition, matching and tracking belong to the most frequent computer vision tasks. We have implemented a complete classifier framework in MATLAB capable of these applications and thoroughly tested it. Sample output of our classifier is on Figure 1.

At first, distinctive image locations – *keypoints* – are determined by the Harris corner detector [3]. Descriptors are based on comparison of image intensities in randomly chosen pixel pairs around these keypoints. Finding tentative matches is based on Naive Bayesian classifier. In object detection and tracking applications, RANSAC [2] is used for fitting of geometrical model.

The key component of the learning and classification process is the patch description. Patches around keypoints are evaluated by comparison of intensities in randomly formed pixel pairs called *binary features*. Many learning samples are generated for each class (patch) during offline learning phase by transforming the model image. The patches are evaluated in each transformation and probabilities of certain values of the features are used as patch descriptors. Joint probability of all binary features would require extreme amount of memory, as 2^N values (N represents number of binary features) would have to be stored for each class. Assuming independence of all binary features would rapidly reduce necessary amount of memory, but it completely ignores the correlation between features. The compromise consists in partitioning binary features into M groups of size $S = \frac{N}{M}$ called *ferns*. Further information can be found in [1] and in [4], section III.

2 Ferns: Learning and Classification

In this section, we will describe the algorithm we used for learning and classification. Source code description will follow in section 3. The algorithms were implemented in MATLAB version 7.8 and tested in MATLAB version 7.8 and 7.9.

2.1 Learning

At first, the model image is converted to greyscale (if necessary) and surrounded with free space (black pixels) to prevent index overflow when evaluating the patches in transformed image samples.

After that, Harris corner detector finds keypoints in the model image. Keypoints too close to the image borders (closer than half of the patch size

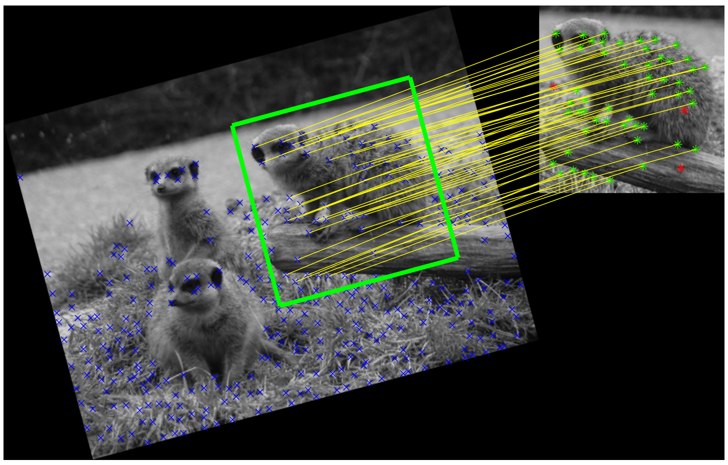
Figure 1: Finding correspondences (yellow lines), object detection (green rectangle). Blue crosses in the query image represent Harris keypoints. Green stars in the model image are correctly classified classes, while the red ones are incorrect.

in respective direction) are removed, the rest represent classes. Classes are identified by index in a variable containing their coordinates.

The model image is transformed in many ways to simulate possible appearance of the object in query images (scenes). Affine transformation matrices are generated for all combinations of given values of θ, ϕ and $\lambda_{1,2}$ in the formula

$$\mathtt{T} = \mathtt{R}_\theta \mathtt{R}_{-\phi} \operatorname{diag}(\lambda_1, \lambda_2) \mathtt{R}_\phi \,, \tag{1}$$

where $\mathtt{R}$ represents rotation by the subscripted angle, $\lambda_{1,2}$ represent scaling.

For a given patch size, the coordinates of pixel pairs forming binary features are randomly chosen in range from $-h_x$ to h_x horizontally and from $-h_y$ to h_y vertically (where h represents half size of a patch in respective direction) and randomly partitioned to ferns.

Now, we describe the main loop over all generated transformations. At first, the image and the keypoints' coordinates are transformed by the given transformation matrix. Gaussian noise with zero mean and large variance 25 (for intensity values from 0 to 255) is added to image intensities to increase robustness. Additionally, Gaussian smoothing with a mask of 7×7 is applied.

The core of the algorithm consists in classification of patches in the given transformation. Binary features within a fern are evaluated, i.e., the intensities of the two pixels within a binary feature are compared and evaluation is returned – 1 if the intensity of the first pixel in the pair is smaller than the intensity of the other one, 0 otherwise. Binary number formed by ‘concatenating’ evaluations of features within a fern is converted to decimal representation and 1 is added, so that the evaluation can be used as index. This is done with all ferns on all patches (classes). The number of occurences of particular fern evaluations is stored for each fern of each class and finally, after repeating the whole process for all transformations, recalculated to probability (all values are divided by a constant: number of samples + $2^{\text{fern size}}$). The evaluations that have never occured are treated as if they have occured once to prevent classification failure when calculating product of probabilities.

If requested, all classes’ coordinates are translated by a given value in all eight directions (horizontally, vertically and diagonally), thus the core of the learning algorithm is run eight more times. This improvement should help overcome possible Harris detector inaccuracy.

To improve classification, we added *post-scaling*, which extends and refines the scaling range given by $\lambda_{1,2}$ values in (1). To make the classification faster, the post-scale probabilities are calculated already in the learning phase. The scale change is simulated by resizing ferns. It can be also described as learning ferns on different patch sizes. Resizing is done by the same coefficient in both directions (i.e., aspect ratio is retained).

2.2 Classification

After learning the ferns probability model, classification can be run in one of the three modes: matching, finding object in a scene and tracking.

Matching mode is used to find correspondences between keypoints in the model (classes) and in the query image (yellow lines on Figure 1). By default, the most similar class is found for each patch around keypoints in the query image. Our classifier also allows matching in opposite direction, where the most similar keypoint in the query image is found for each class. As there are usually less classes than keypoints in the query image, the opposite approach is generally faster, but the recognition rate decreases.

Finding mode is intended for object detection applications. RANSAC is used to estimate homography (transformation matrix) from found correspondences between model classes and keypoints in the query image. As a by-product, correctly matched correspondences are determined as inliers, incorrect as outliers. The computed homography essentially finds exact location of the sought object, see the green rectangle in Figure 1.

Tracking mode is suitable for object tracking in image sequences, where the difference of successive images is rather small. This mode extends the finding mode by restriction of keypoints translation between two succeeding images (maximal translation may be set by user). Additionally, the object is first searched in surrounding of its position in the previous image in the sequence. If the object is not found in the restricted area, the whole image is searched through (as in the finding mode).

Now, we describe the classification process. Firstly, the image to be classified is converted to greyscale (if necessary) and keypoints – Harris corners are detected (in bounded area in the tracking mode according to previous object location, else in the whole image). Keypoints too close to the image borders (closer than half of the patch size in respective direction) are removed.

Correspondences between classes in the model image and keypoints in the query image are determined this way: Patches around all keypoints are evaluated using the same ferns that were used for learning. The prior (learned) probabilities of the acquired evaluations of all ferns are loaded for each class and multiplied, the particular keypoint (patch) is then assigned to the class with the largest product. This can be formulated as finding

$$\hat{c_i} = \underset{c_i}{\operatorname{argmax}} \prod_{k=1}^{M} P\left(f_k \mid C = c_i\right) \ , \tag{2}$$

where f_k represents evaluation of k^{th} fern, c_i represents i^{th} class and M is number of ferns. Maximum translation of each keypoint between the current and previously classified image is limited in the tracking mode, which restrains classes the keypoint can be assigned to.

In the finding and tracking mode, if not enough inliers were found to be sure that RANSAC returned correct homography (e.g. less than 10 inliers), the whole process is repeated for succeeding post-scales. In case that all post-scales were tried and the given number of inliers was achieved for none of them, the post-scale with the highest number of inliers is taken as the best. Classification of the next image in the tracking mode will begin with the best post-scale used for the last image.

If the *refine mode* is on, the classification may be re-run with higher RANSAC iteration limit to achieve better results when not sure about the correctness of the previously calculated homography. More about the refine mode can be found in section 3.2.

3 Implementation

As the process of image recognition consists of two main phases, learning and classification, there are two main functions, `f_learn` for learning and `f_find` for classification. Demonstration scripts are also distributed, detailed description can be found in the succeeding sections.

Brief overview The script `demo_ferns.m` provides a brief overview of object detection using our classifier. The image specified in `input_image` is displayed, user selects the object to be learned by dragging a rectangle with mouse. After the learning and classification process finishes, the result is displayed. The estimated object position is marked green. Blue crosses in the query image indicate found keypoints (Harris corners). Red stars in the model image represent classes that were determined by RANSAC as outliers, while green stars are inliers. Matching of correctly classified pairs (inliers) is drawn as yellow lines (Figure 1).

Sample data We recommend downloading ground truth image sequences[1] to take full advantage of the demonstration scripts. Each of the **_img* and **_gt* archives should be extracted to separate directory. Directory structure for downloaded archives extraction is already prepared in this package in the directory `testdata`. Images used for learning named `img_learn.jpg` are stored in **_img* directories in `testdata` in this package.

3.1 Learning

Function `f_learn` is used to learn ferns model on an input image. The syntax is
`model = f_learn(im, par)`,
where `im` is a model image (to be learned) represented as uint8 matrix. Color image will be automatically converted to greyscale. It is possible to set the pathname of the image instead, then the image is automatically read in the required format. Structure `par` contains parameters for learning. Description of the parameters can be found in the beginning of `f_learn.m`. All parameters are compulsory.

The fields of the output structure `model` are also described in the beginning of `f_learn.m`, but none of the fields has to be handled by user.

Sample configuration of learning is in `demo_learn.m`. The script consists of setting all parameters and calling the function `f_learn` in the end. The learned model is stored in the variable `model`. It can be saved by the MATLAB command
`save` *filename* `model`
and later reloaded and repeatedly used for classification by calling
`load` *filename*.
This is beneficial to save time.

[1]`http://cmp.felk.cvut.cz/demos/Tracking/linTrack/data/index.html`

3.1.1 Transformation parameters

The image is transformed in the learning phase in various ways to ensure robustness. Transformation parameters used in (1) are set in the structure `par.tform`. All combinations of values in the fields of `par.tform` (`theta_values`, `phi_values`, `lambda1_values`, `lambda2_values`) are generated. It means that the time necessary for learning rapidly increases when setting more values. Total number of learning samples can be calculated as the product of sizes of all `par.tform` fields.

If you would like to use random values in `par.tform` fields instead of setting them explicitly, you can generate them by the supplied function `gen_rand`. This function takes three parameters:
`rand_array = gen_rand(min_value, max_value, count)`,
where the number range is specified by `min_value` and `max_value`. The function generates 1-by-`count` array of random numbers in the given range.

3.1.2 Post-scaling

Post-scales used for better coping with the searched object's scale changes (see 2.1) are set in the field `par.post_resize` as a row vector. For instance, the value 0.5 represents resizing the image to half size. It is important to leave 1 as the first value in the vector. Sample setting can be found in `demo_learn.m`.

Post-scaling can be turned off by setting
`par.post_resize = 1`.

3.2 Classification

After learning ferns (or loading a previously learned model), it is possible to begin the classification. Function `f_find`, which is the main function for classification, supports three modes: matching, finding object in a scene and tracking.

The syntax of the function is
`[corners f_data] = f_find(im, model, par, par_cl)`,
where `im` is the image to be classified. As in `f_learn`, it can be either uint8 matrix or a pathname of the image, color image will be automatically converted to greyscale. Variable `model` is the learned ferns model (output from `f_learn`), `par` are parameters used for learning. If `par` is no more in the variable environment (e.g. when loading saved model), it can be fully restored from `model.par`.

Parameters for classification are set in the structure `par_cl`. Unlike in the learning phase, all the parameters are optional. Anyway, we strongly recommend setting the essential values manually. Detailed parameters description and default values, which are used if the parameters are not explicitly set, can be found in the beginning of `f_find.m`.

Function `f_find` has two output arguments: `corners` and `f_data`. The variable `corners` is a 4×2 matrix, in each row there are x- and y-coordinates of the corner of the estimated area, where the object should be located. If the object is not found (or in the matching mode), `corners` is filled by zeros.

Matched pairs are stored in the variable `f_data.matching_data` in form of 6-by-N matrix, where first two rows contain class coordinates (in the model), 4th and 5th row contain keypoint coordinates (in the query image). If not in the matching mode, it is possible to extract the pairs that were classified by RANSAC as inliers. These 'correct' pairs (correspondences) are stored in the variable `f_data.corresp`, which is of the same structure as `matching_data`.

Other fields of `f_data` are described in the beginning of `f_find.m`.

Refine mode In the refine mode, if less than `par_cl.ransac_inls_good` inliers are found, `f_find` is recalled with higher RANSAC iteration limit set in `par_cl.ransac_max_iter_fine`. If `par_cl.ransac_inls_refine_all` or less inliers were found in the first run, `f_find` tries all post-scales in the second run, otherwise it takes only the best post-scale from the first run (the one with the most inliers). The refine mode is turned on by explicitly setting the `par_cl.ransac_max_iter_fine` value. This mode increases recognition performance, but decreases speed. It is available in the finding and tracking mode.

3.2.1 Matching

One of the possible usages of `f_find` is matching, i.e., finding correspondences between keypoints in the model (classes) and in the query image.

Sample matching task is in `demo_match`. The image for classification is selected by setting path (`PATH_IMG`) and index (`IMG_INDEX`). The function `f_find` is run in matching mode by setting
`par_cl.matching_mode = true`.

The script `demo_match` shows a figure with found correspondences (Figure 2). Matched pairs can be displayed by calling
`disp(f_data.matching_data)`.

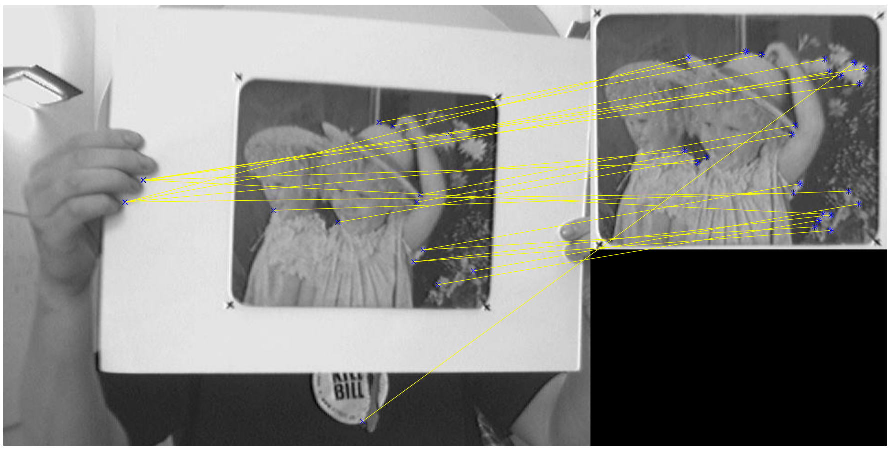

Figure 2: Matching mode – sample output.

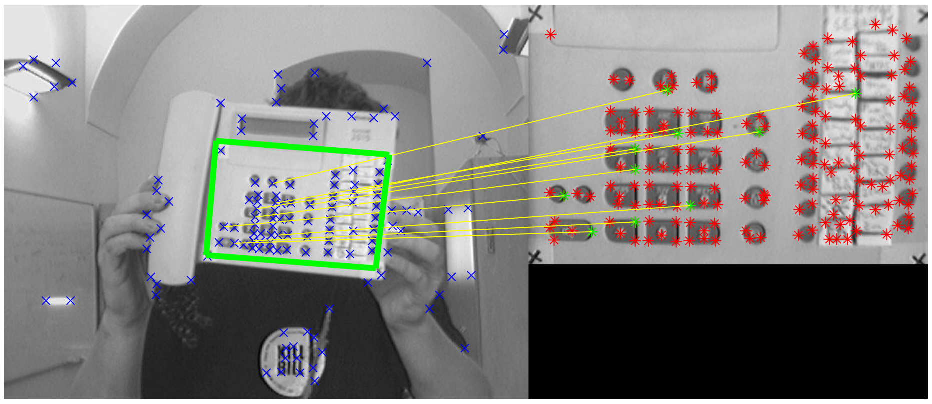

Figure 3: Finding mode – sample output.

3.2.2 Finding object in a scene

Using `f_find` to find an object in a scene is demonstrated in `demo_find`. This script also uses some features of tracking. Unlike the other two modes, no specification of finding mode usage in `par_cl` is necessary as this mode is taken as default.

The script runs a test sequence on images in directory `PATH_IMG` starting with `MIN_INDEX` and continuously increasing the index by `STEP` until `MAX_INDEX` is reached. The images are classified and `corners` from `f_find` are compared with ground truth stored in `PATH_GT` directory. The variable `CORR_DIST_LIMIT` is used to set the maximum position difference (both horizontally and vertically) of all corners in comparison to the ground truth. The area defined by `corners` is drawn green if it matches ground truth, otherwise the color is red. Blue crosses in the query image indicate found keypoints (Harris corners). Red stars in the model represent classes that were determined by RANSAC as outliers, while green stars are inliers. Matching of correctly classified pairs (inliers) is drawn as yellow lines (Figure 3).

Classification statistics are saved during the test sequence to the variable `results`. As the script finishes, brief summary is displayed and a figure shows, how many times was each class classified as inlier. Detailed results can be displayed by typing

```
parse_results(results).
```

Used post-scale indicates the index of the scale in `par.post_resize` that was finally used. If RANSAC returned enough inliers (`par_cl.ransac_inls_good` or more), the index is stored as it is. The value is increased by 100 to indicate that RANSAC found less than `par_cl.ransac_inls_good` inliers, i.e., `f_find` tried all post-scales and enough inliers were found in none of them.

It is possible to save displayed figures as EPS images and results as a text file for later inspection by setting

```
SAVE_OUTPUT = true.
```

All files are saved to the directory specified in `PATH_OUT`. Please note that the output directory has to be manually created.

3.2.3 Tracking

Tracking mode is similar to the finding mode. The demonstration script `demo_track` uses cell mode, which enables the user to easily re-run only the classification by evaluating the last cell ‘Classification’ with different parameters without repetitive learning. This script goes through the whole process of learning and recognition, i.e., no previous learning using `demo_learn` is necessary (unlike the preceding scripts).

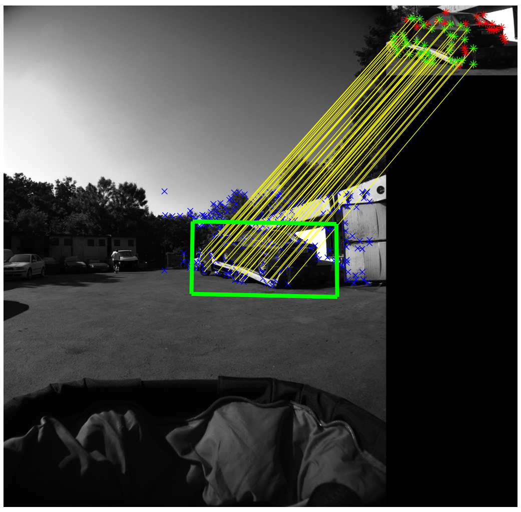

Figure 4: Tracking mode – sample output.

The script starts with a figure showing image defined in `im_getrect`, where the user has to drag a rectangle to choose the model for learning. After the learning finishes, the classification proceeds similarly as in the finding mode.

The main difference between this mode and the finding mode consists in restriction of keypoints translation between two succeeding images and restriction of the image area, where keypoints are searched for (as can be seen on Figure 4 – blue crosses representing keypoints are only in the close surrounding of the object). Tracking mode is enabled by `par_cl.tracking_mode = true`.

Tracking mode is temporarily turned off if the object is lost. This happens if `par_cl.ransac_inls_bad` or less inliers are found. The value should be selected carefully, because otherwise the classification will repeatedly fail when trying to find the object near the last correct position, where it is no more located though.

It is recommended to set the value `par_cl.tracking_dist`, which limits the maximum position difference of all keypoints (in both directions) between two succeeding images. If the value is not set, default value from `f_find` is used. Harris keypoints detection area has to be passed to `f_find` as a parameter in the `par_cl.tracking_crop` variable. Calculation of the area is implemented in the script `demo_track`. If the variable `par_cl.tracking_crop` is not set, Harris corners are searched in the whole image by default.

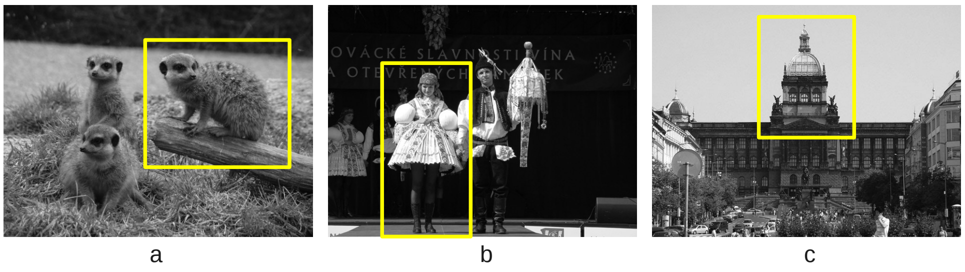

Figure 5: Generated transformations experiment – images: a – animals, b – costume, c – museum. Yellow rectangle specifies the object that is localized in transformed images.

4 Experiments

We performed experiments on various image sets to evaluate our classifier and the modifications not described in [4] that we introduced – post-scaling, refine mode, selecting pairs with the highest probability values for RANSAC, finding correspondences for model classes instead of query image keypoints and keypoints translation in the learning stage.

4.1 Tracking and object detection

4.1.1 Generated transformations

In this experiment, we wanted to evaluate our classifier on basic transformations as scale and rotation. Aditionally, we tested how selecting a number of pairs with the highest probability product for evaluation by RANSAC affects the recognition rate. At first, we selected objects in the images on Figure 5 (marked yellow). Then we generated several transformations and let our classifier determine the objects' positions in the transformed images.

The transformation matrix is calcuated using (1). This allows us to easily observe the influence of rotation and scaling on the recognition rate. Additionally, we can precisely determine the correct object position in the transformed image, because we know the transformed object coordinates.

Learning on 900 samples took about 5 minutes, training transformations consisted of rotation from 0 to 360° with step 10° and scaling from 0.5 to 1.5 with step 0.25 in both directions. Shear was not used in this experiment ($\phi = 0$ in (1)). 32×32 px patches were evaluated by 30 ferns of size 11. Using non-MEX Harris corner detector with threshold 10000, 51 classes were found on image *animals*, 62 classes on image *costume* and 39 classes on image *museum*. Post-scaling coefficients variable (`par.post_resize`) was set to `[1 0.5 0.75 1.25 1.5 1.75 2]`, translation of Harris corners was disabled (`par.surrounding = 0`).

Classification ran in finding mode with default parameters except for RANSAC maximum iterations thresholds that were set to `ransac_max_iter = 1000` and `ransac_max_iter_fine = 20000` (which implies that the refine mode was on). Query image was evaluated as correct if all four corners were found correctly with tolerance of 20 pixels (both horizontally and vertically).

Table 1 shows results of classification for rotation with no scaling (100 %), resizing to 75 % and 150 %. It is obvious that rotation has no negative effect on the object finding success. Some detection failures for the image *museum* in rotations combined with scaling could be caused by lower number of classes and by repetitive objects – windows, which make the classification more difficult.

Simple scaling (without rotation) yielded rather worse results (Table 2). The object was correctly found on none of the 3× enlarged images, similarly for the reduced images *animals* and *museum*. This is mainly caused by the Harris corner detector behavior, as the number and position of found Harris corners considerably differs for various scales. Better results may be achieved by changing the Harris detector threshold for extensive scale differences. Stable keypoints selection in the learning phase would help overcome this problem, which is one of the most important improvements to be implemented in the future.

We also tried choosing only the best correspondences for RANSAC. The number of pairs with the highest probability product to be evaluated by RANSAC is determined in the field `n_best_matches_for_ransac` of the parameters structure `par_cl`. The image *museum* was used for this experiment. Recognition performance rapidly decreased for very low values (about 1.5× number of classes and less), but for values about 2.5× number of classes, we achieved better results than without this feature. There is no general recommendation whether to use this feature and what value should be set, but recognition performance improvement may be achieved with carefully selected `n_best_matches_for_ransac` value.

4.1.2 Ground truth sequences

Three ground truth image sequences (Figure 6) available on[2] – *mouse pad* (6946 VGA images), *phone* (2300 VGA images) and *towel* (3231 VGA images) – were used for object finding and tracking experiments. The objects are ma-

[2] `http://cmp.felk.cvut.cz/demos/Tracking/linTrack/data/index.html`

Table 1: Generated transformations experiment results – rotation (with scaling). Symbol + represents correct classification, – incorrect.

Scale	Rotation (degrees counterclockwise)													
100 %	5	15	25	30	45	60	90	135	180	225	270	305	350	355
Animals	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Costume	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Museum	+	+	+	+	+	+	+	+	+	+	+	+	+	+
75 %	5	15	25	30	45	60	90	135	180	225	270	305	350	355
Animals	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Costume	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Museum	−	+	+	+	+	+	+	+	+	+	+	+	+	+
150 %	5	15	25	30	45	60	90	135	180	225	270	305	350	355
Animals	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Costume	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Museum	−	−	+	−	−	+	+	+	−	+	+	+	+	−

Table 2: Generated transformations experiments results – scaling. Symbol + represents correct classification, – incorrect.

Scale	33 %	50 %	75 %	100 %	150 %	200 %	300 %
Animals	−	−	+	+	+	−	−
Costume	+	+	+	+	+	+	−
Museum	−	−	+	+	−	−	−

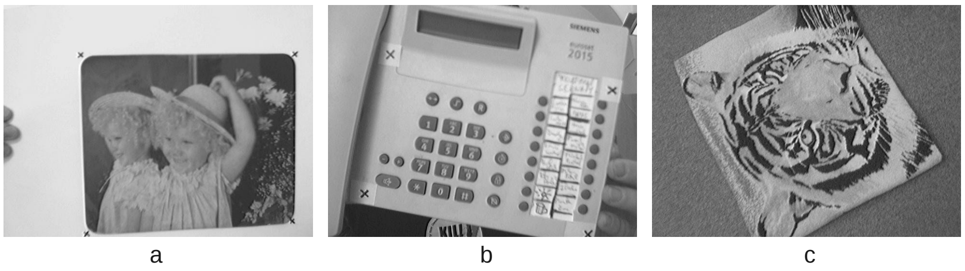

Figure 6: Ground truth data sets: a – mouse pad, b – phone, c – towel.

nipulated in many ways in the sequences – rotated, taken further and closer, tilted, partially covered etc. We wanted to evaluate the impact of tracking mode, refine mode, post-scaling and their combinations on the recognition performance. With the provided ground truth data (corner coordinates of the searched objects), it is easy to automatically determine whether the object was found correctly or not.

The classifier was learned on the `img_learn.jpg` image for each sequence, which can be found in respective **_img* directories under `testdata` distributed within this package. We used 30 ferns of size 11, Gaussian filter standard deviation was 2, patches were 32×32 pixels in size and we used non-MEX Harris corner detector with threshold 5000, which resulted in 60 classes on *mouse pad*, 183 classes on *phone* and 112 classes on *towel* data set. 13068 training samples were generated by rotation all around with step $10°$, shear (ϕ in (1)) 0 and two random values from 0 to $180°$ and scaling from 0.5 to 1.5 with step 0.1 both horizontally and vertically. The translation of Harris corners (`par.surrounding`) was set to 0 (no translation) and the post-scaling coefficients (`par.post_resize`) were `[1 0.5 0.75 1.25 1.5 1.75 2]`. Learning with these parameters takes about 90 minutes on a standard desktop computer.

All sequences were classified in five scenarios to compare results dependent on the usage of tracking/finding mode (t^+/t^-), refine mode on/off (r^+/r^-) and post-scaling on/off (p^+/p^-). Scenarios' parameters are summarized in Table 3. Supplied scripts `demo_find` (scenarios $t^-r^+p^+$, $t^-r^-p^+$ and $t^-r^-p^-$) and `demo_track` (scenarios $t^+r^+p^+$ and $t^+r^-p^+$) were used for classification. Each 5th image in the sequences was classified, which reduced the computation time to one fifth while the results were still enough reliable. We used default `par_cl` values except for RANSAC maximum iterations thresholds, which were set to `ransac_max_iter = 1000` and, in scenarios $t^-r^+p^+$ and $t^+r^+p^+$, `ransac_max_iter_fine = 20000`. The tracking was run with `tracking_dist = 80`. The script `demo_track` was modified to better meet the requirements of this experiment, i.e., the learning stage was removed and the initial corners' coordinates were set manually (instead of retrieving them automatically using `getrect` when selecting the object to learn). The value of `par_cl.ransac_inls_bad` was manually increased by one from the default value in the tracking mode (i.e., in scenarios $t^+r^+p^+$ and $t^+r^-p^+$), because we found out that 'resetting' the tracking mode (temporarily turning it off, when the object position is lost) even when 5 inliers are found produces much better results. Using the default value, it keeps 'wrong' tracking much longer, because with only 4 inliers, RANSAC often returns senseless homography.

Recognition performance of our classifier for each data set, i.e., in how many images was the object correctly found, is summarized in Table 4. An image is determined as correctly classified if all estimated corners' coordinates match the ground truth corners' coordinates with maximum difference of 50 pixels both vertically and horizontally. Such a high position tolerance is necessary because of the fact that the learned model image (`img_learn.jpg`) corners are being localized in the query images (as it would be useful in real applications), but the ground truth data contain crosses' coordinates in the *mouse pad* and *phone* sequences. This issue doesn't affect the *towel* sequence, but we want to keep the value the same for all experiments for better comparison.

Post-scaling is enabled in scenario $t^-r^-p^+$ and disabled in scenario $t^-r^-p^-$, while all other parameters are the same. Here we show that the post-scaling we used in our implementation improved the results by 52 % for *mouse pad*, by 359 % for *phone* and by 112 % for *towel* data set. Time consumption of re-running RANSAC with very limited number of iterations (1000, because the refine mode is off in scenarios $t^-r^-p^+$ and $t^-r^-p^-$) is quite low.

The refine mode provides even much better results, which is obvious from comparison of the values in columns $t^-r^+p^+$ and $t^-r^-p^+$ (finding mode), or in columns $t^+r^+p^+$ and $t^+r^-p^+$ (tracking mode). Whether to enable the refine mode or not depends on the speed requirements of a certain application. Using non-MEX Harris corner detector, one image is generally classified in hundreds of milliseconds if the refine mode is disabled, but it may take seconds if the refine mode is on and refine is requested. For instance, on the data set *mouse pad*, it takes about 200–400 ms if the first used scale returned enough inliers or about 800 ms if all post-scales were tried, but it takes about 5–6 seconds if refine is requested in the refine mode. (Only the computation time is measured, not the disk and graphics operations.)

The tracking mode is useful in combination with disabled refine mode (scenario $t^+r^-p^+$). It saves time (because Harris corners are generally searched for on a smaller area; even less than 100 ms may be necessary for an image) and produces slightly better results than the finding mode (scenario $t^-r^-p^+$). If the refine mode is on, even a bit worse results are returned, when the tracking mode is on (scenario $t^+r^+p^+$). This could be caused by tracking failure, which spoils some images that are classified correctly in the finding mode (scenario $t^-r^+p^+$), as RANSAC has enough iterations to estimate the correct homography even without the support of tracking mode restrictions.

4.1.3 Ladybug sequences

Coping with high resolution real world images, background changes, low light conditions and partial visibility of various objects were to be evaluated in

Table 3: Ground truth data sets experiments – parameters (+ on, – off).

	$t^-r^+p^+$	$t^-r^-p^+$	$t^-r^-p^-$	$t^+r^+p^+$	$t^+r^-p^+$
Tracking mode	−	−	−	+	+
Refine mode	+	−	−	+	−
Post-scaling	+	+	−	+	+

Table 4: Ground truth data sets experiments – recognition performance (%).

	$t^-r^+p^+$	$t^-r^-p^+$	$t^-r^-p^-$	$t^+r^+p^+$	$t^+r^-p^+$
Mouse pad	92.16	76.33	50.14	90.50	79.86
Phone	48.70	31.96	8.91	47.17	35.65
Towel	93.82	88.56	78.98	92.43	91.19

another set of experiments done with *ladybug* image sequences. As Ladybug3[3] is a spherical multiple view motion camera, some experiments were done on sequences from a single camera, the others were run on panoramatic images.

Two data sets were used: *vrakoviste* taken on a scrapyard and *dvorana* from the school yard. Single camera shots were 1232×1616 pixels in size, panoramatic images were 5400×2700 pixels large. We used slightly modified `demo_track` with 30 ferns of size 11, patches were 40×40 pixels in size. Harris threshold was set to 10000 in most cases (non-MEX Harris corner detector was used), but it was decreased in few cases to get more classes on dark objects. Learning on 441 samples with rotation 0 and 15° both clockwise and counterclockwise, shear (ϕ in (1)) 0, 15° and 165° and scaling from 0.7 to 1.3 with step 0.1 both horizontally and vertically took about 2–3 minutes. The rotation range was extended by additional values for sequences with larger rotation changes. Translation of Harris corners (`par.surrounding`) was set to 0 (no translation) and the post-scaling coefficients (`par.post_resize`) were `[1 0.5 0.75 1.25 1.5 1.75 2]`. Default `par_cl` values were used except for RANSAC maximum iterations thresholds, which were set to `ransac_max_iter = 1000` and `ransac_max_iter_fine = 20000`, `ransac_inls_bad` was set to 5. The variable `tracking_dist` was set to 100.

Figure 7 shows results of experiments on images from a single camera on the *vrakoviste* data set. Experiments *a* and *b* show a car with high contrasts

[3]`http://www.ptgrey.com/products/ladybug3/index.asp`

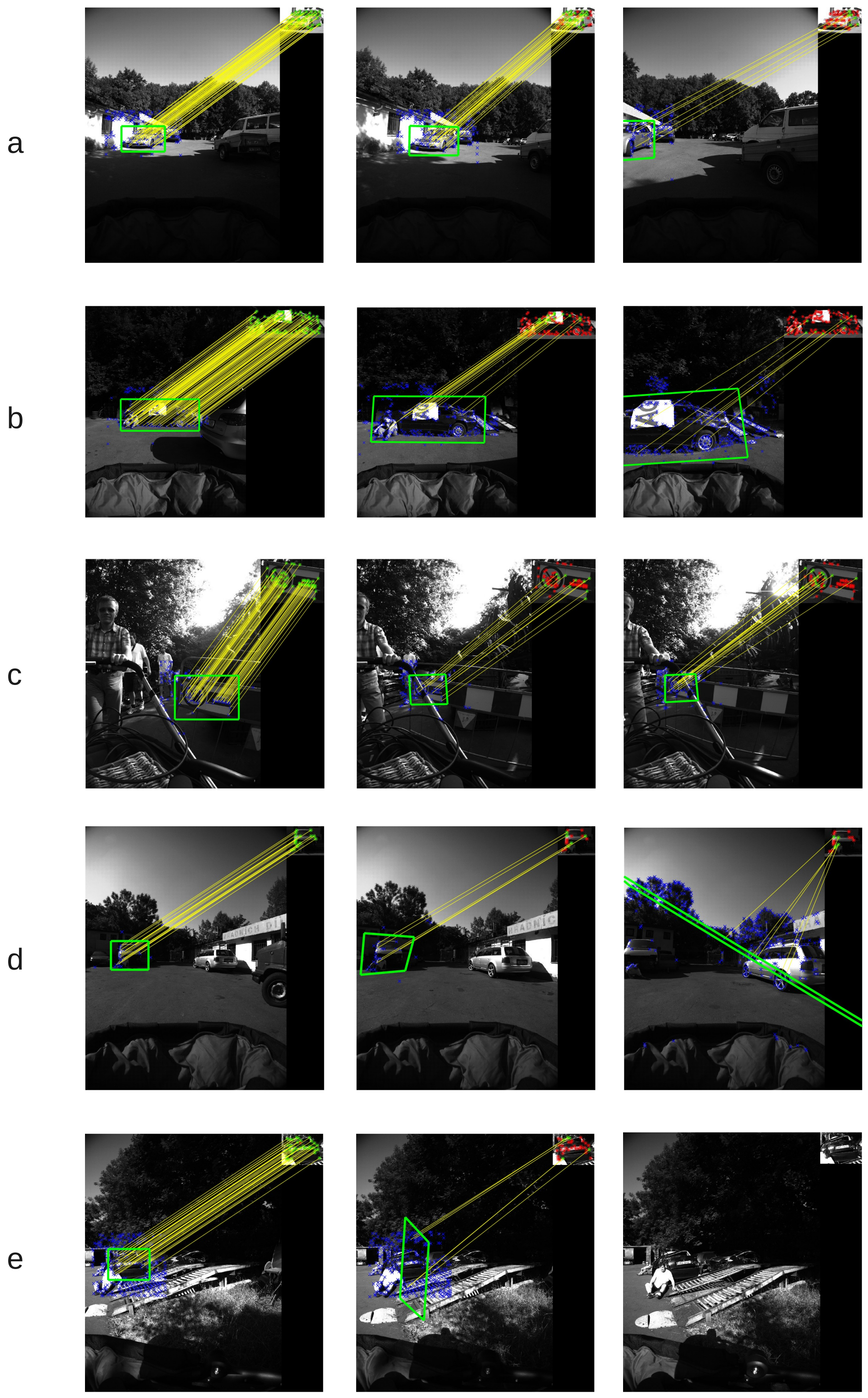

Figure 7: Ladybug experiments – vrakoviste data set, single camera.

and many classes, which gave very good results. Even when half of the car is out of the scene, it can still be recognized. In image set *c*, we show that ferns are very efficient for sign recognition, as the text and pictograms on the sign provide many unambiguous classes. The sign is recognized well even when partially covered. On the other hand, dark objects with too few Harris corners (*d*), particularly in combination with considerably changing learned surrounding (*e*), returned worse results, we were able to track them only for a few frames of the sequences.

Results of single camera experiments from the *dvorana* data set are captured on Figure 8. A kneeling person (*a*) is tracked well while slightly changing the point of view until the person stands up, then it differs too much from the learned model and the track is lost. Experiment *b* demonstrates, how background change affects the classification. The background is initially dark left of the person and bright right of the person, but as the viewpoint changes, the situation reverses. Because of that, only the patches in the middle maintain correct classification until the rotation exceeds learned rotation range, then the tracking finally fails. Tracking a window (*c*), which is specific by many Harris corners and high contrasts, outperforms tracking any other objects from this data set. The window was correctly found in all images of the tested part of the sequence, although less than half of the window was visible in the last frames (as can be seen on the third image of Figure 8 – *c*). Dark object and rather low light conditions resulted in very fast tracking failure (*d*). Considerable decreasing the Harris threshold is the only way to get at least satisfactory results.

Figures 9 and 10 show results of experiments on panoramatic images. The results were generally worse than on single camera images, as the panoramatic picture distortion represents transformations that our classifier is not learned for. As we found out, 64-bit environment and large amount of memory is required for very high resolution images (because of the Harris corner detector requirements). Finally, we tried to track a window on one of the buildings over the whole sequence (Figure 10 – *c*). An object was found on 219 of 401 images, on most of them the window was found correctly. The classifier was repeatedly able to recover from track loss (successful recovery is captured on the second and third image on Figure 10 – *c*). Once, the neighbouring window was found, but it changed to the correct one after few frames. Classification of one image took from 300 ms to 2 seconds, but when the tracking mode was temporarily left (because of track loss), it took about 30 seconds to classify a single image.

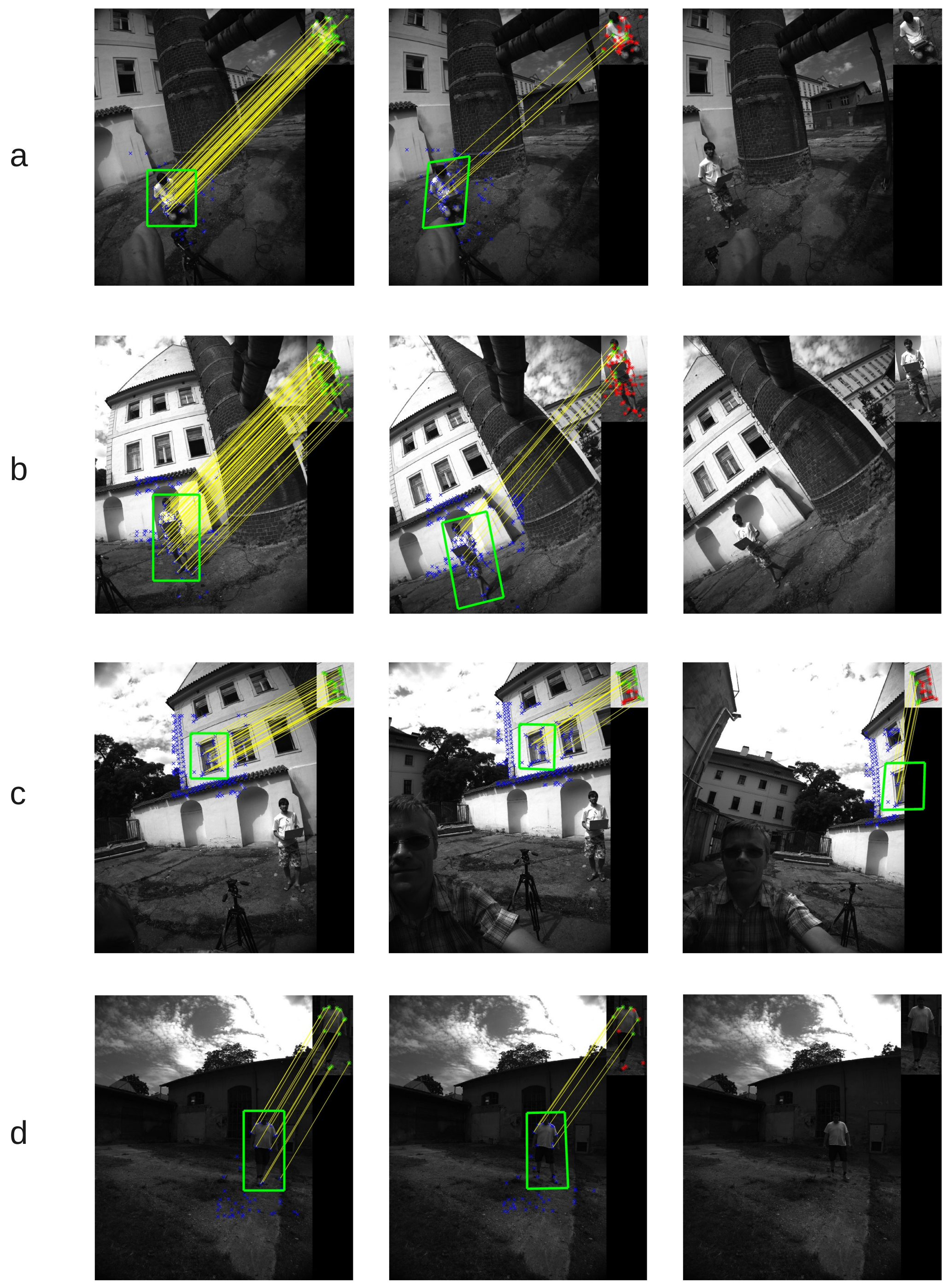

Figure 8: Ladybug experiments – dvorana data set, single camera.

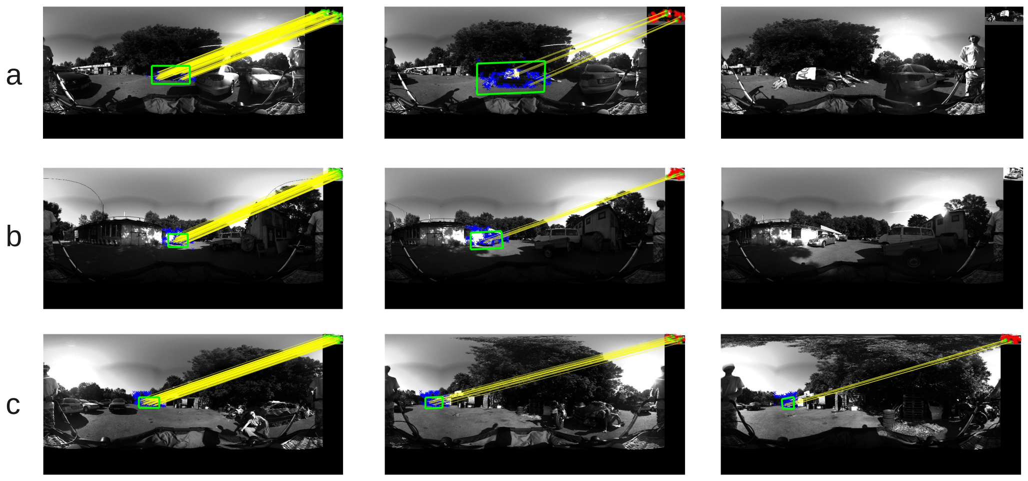

Figure 9: Ladybug experiments – vrakoviste data set, panoramatic view.

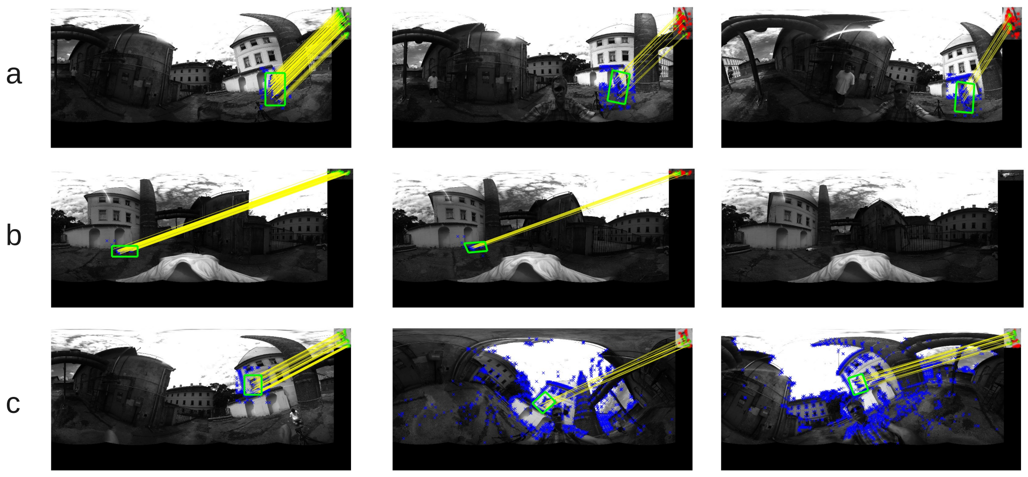

Figure 10: Ladybug experiments – dvorana data set, panoramatic view.

4.2 Matching

Using the generated images from section 4.1.1, we performed a few experiments also in the matching mode. Parameters for learning remained the same. Recognition rate, i.e., how many classes from the learned model were correctly found in the query image, was measured for certain transformations.

We also tested our modifications – finding correspondences for model classes instead of query image keypoints and translation of keypoints in the learning stage – which were not described in [4].

The classification in matching mode was performed with default `f_find` parameters. A class was evaluated as correctly classified if any keypoint in the query image was correctly matched with the given class (admitting inaccuracy of 5 pixels both horizontally and vertically). There are generally more keypoints in the query image than in the learned object (which is usually smaller than the query image). The most probable class is assigned to each keypoint, hence one class may be paired with more keypoints. If `par_cl.corr_from_model` is set to `true`, the best keypoint in the query image is found for each class, therefore at most one keypoint can be paired with one class. This yields worse results than default finding the best class for each keypoint in the query image, unless the query image contains less keypoints than number of classes. Our experiment confirms this, we got better results using the default approach for all transformations except for resizing to 33 %.

Comparison of recognition rate for different scales is on Figure 11, the effect of rotation is shown on Figure 12. We used three scenarios to determine the effect of our modifications. The experiments in scenario *def* were run with `par_cl.corr_from_model = false` and `par.surrounding = 0`. We changed `par_cl.corr_from_model` to `true` in scenario *cfm* to find out, how direction of correspondence finding affects the classification. Scenario *trk* differs from default settings in scenario *def* by setting `par.surrounding = 1`, as we tested the effect of learning with translation of keypoints in the model.

The results of classification when finding the best keypoint for each class (scenario *cfm*) were much worse than with the default settings (scenario *def*) except for resizing to 33 % because of the previously mentioned reasons. Learning with translation of keypoints (scenario *trk*), which makes the learning stage approx. 5× longer, brought slightly better results on average. However, there is no general recommendation whether to use learning with translation or not, as the classifier was in some cases less successful than in scenario *def*.

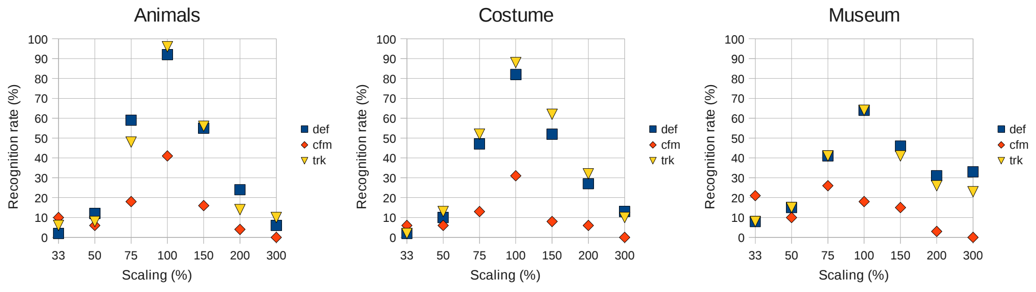

Figure 11: Transformed images experiment – results for scaling. Scenarios: *def* – default settings, *cfm* – finding correspondences for keypoints in the model image, *trk* – translation of keypoints in the learning stage.

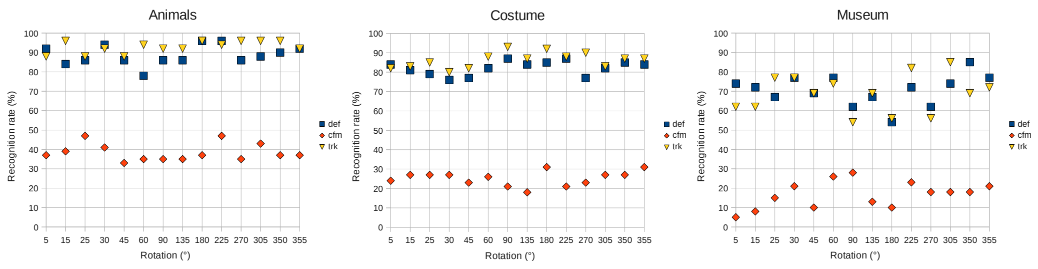

Figure 12: Transformed images experiment – results for rotation. Scenarios: *def* – default settings, *cfm* – finding correspondences for keypoints in the model image, *trk* – translation of keypoints in the learning stage.

5 Conclusion

We have successfully implemented a classifier using random ferns for image matching and object tracking. Some modifications and improvements were introduced and evaluated in experiments. Post-scaling and the refine mode showed up to have appreciable positive effect on the recognition performance, while the effect of selecting pairs with the highest probability values for RANSAC and finding correspondences for model classes instead of query image keypoints is disputable. Learning with translation (moving patches by one pixel in all eight directions) should increase the recognition rate in the matching mode in most cases.

Tracking mode improves results in combination with disabled refine mode, which could be effective in applications requiring fast classification. If the refine mode is on (RANSAC maximum iterations threshold is high enough), we didn't achieve better results using the tracking mode instead of the finding mode.

Performance of the classifier is dependent on the Harris corner detector. It performs well on objects with high contrasts, because enough classes (Harris corners) are found and the patches are very different, while dark objects with nearly uniform surface yield unsatisfactory results. Both learning and classification parameters have to be adjusted for each application, there is no configuration working well in all situations.

Further improvements like stable keypoint selection in the learning stage or adaptive learning would make the classifier more powerful.

References

[1] A semi-naive bayesian classifier for fast patch classification. `http://cvlab.epfl.ch/alumni/oezuysal/ferns.html`.

[2] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[3] C. Harris and M. Stephen. A combined corner and edge detection. In M. M. Matthews, editor, *Proceedings of the 4th ALVEY vision conference*, pages 147–151, University of Manchester, England, September 1988. Online copies available on the web.

[4] Mustafa Özuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *IEEE Trans. Pattern*

Anal. Mach. Intell., 32(3):448–461, 2010. `http://cvlab.epfl.ch/publications/publications/2010/OzuysalCLF10.pdf`.