

–Table of Contents

Correspondence Problem

Searching for correspondences is a fundamental task in computer vision. The goal is to find corresponding parts (patches) of a scene in two or more images. Finding the correspondences for each pixel is computationally difficult. Therefore, methods based on correspondences of local patches first detect, in each image, the so called local features (a.k.a. interest points, distinguished regions, covariant regions) i.e. **regions** detectable in other images of the same scene regardless of geometric and photometric transformations. Correspondences are then found between local features.

For each feature the exact position and a description of the feature neighborhood is saved. The description can be a vector of pixel intensities, histogram of intensities, histogram of gradients, or other invariant. The choice of the descriptor influences the discrimination and invariance of the feature. Roughly speaking, local feature descriptor is a vector, which characterises the image function in a small, well localized neighborhood of a feature point.

When establishing correspondences, for each feature in one image we look for features from the second image that have a similar description. We call these correspondences “tentative”, as some of them are usually wrong – they do not associate the same points in the scene. This happens because of noise, repeated structures in the scene, occlusions or inaccuracies of models of geometric and photometric transformation.

As a next step we need a robust algorithm, which (based on the knowledge of the image-to-image transformation model) will choose the largest subset of correspondences which are all consistent with the model. Such correspondences are called „verified“ or „inliers“ (with the model). An algorithm widely used for this purpose is called RANSAC (RANdom Sampling Consensus) and will be introduced it in the last section.

1. Feature detection

Feature detection is the first step in the process. We will implement two often used feature detectors. Detector of Hessian maxima and Harris detector.

Hessian detector

The goal of the detector is to repeatedly find the feature points which are well localized and there is enough information in their neighborhood for creating a good description. Detector of the local extrema of Hessians – determinant of Hessian matrix [http://en.wikipedia.org/wiki/Hessian_matrix] (matrix of second order derivatives)

$$\mathbf{H} = \begin{bmatrix} D_{xx}(x, y; \sigma) & D_{xy}(x, y; \sigma) \\ D_{xy}(x, y; \sigma) & D_{yy}(x, y; \sigma) \end{bmatrix},$$

finds centers of the so called blobs, local extrema of the intensity function, which are well localized in a neighborhood given scale σ . Interest points are strong local maxima with value (response) above a threshold t , which is set according to the level of noise in the image.

- Implement a function `response=hessian_response(img, sigma)`, which computes Hessian for each pixel of the given image `img` according to formula: $\det(\mathbf{H}) = D_{xx}D_{yy} - D_{xy}^2$. For computation of the derivatives D_{xx}, D_{yy}, D_{xy} use the derivative of Gaussian with variance σ^2 .

For the localisation of Hessian extrema we will need a function to find out if a point is a local extremum. This is done by comparing the pixel value with pixels in its neighborhood.

- Implement a function `maximg=nonmaxsup2d(response, thresh)` for non-maxima suppression. From input image `response` (matrix of size $H \times W$, where H stands for image height and W for image width) computes a new matrix

maximg of the same size, in which all local maxima with response value greater than *thresh* will have value 1 and non-maxima points or local maxima with response below *thresh* will have value 0. Function checks for local maxima in 8-neighbourhood of the point.

- Connect these two functions into a detector `[x,y]=hessian(img,sigma,thresh)`, which will find Hessian maxima in the image *img* with scale of gradient *sigma*. Return only points (x, y) which were thresholded to 1 - you can use function `find` `[y,x]=find(maximg)` for finding these points in the matrix. Return vectors *x* and *y* as row vectors of the same size, where *x*(1), *y*(1) are points with coordinates of the first point, *x*(2), *y*(2) coordinates of the second, etc.. Pay attention to indices, which are one-based in MATLAB, while we want to have coordinates (0,0) for the top-left pixel of our image.

Harris detector

Harris detector detects points, in which the gradient changes in two orthogonal directions. For this reason it is sometimes called a corner detector. It uses properties of the so called autocorrelation matrix of gradients $\mathbf{C}(x, y; \sigma_d; \sigma_i)$.

$$\mathbf{C}(x, y; \sigma_d, \sigma_i) = G(x, y; \sigma_i) * \begin{bmatrix} D_x^2(x, y; \sigma_d) & D_x D_y(x, y; \sigma_d) \\ D_x D_y(x, y; \sigma_d) & D_y^2(x, y; \sigma_d) \end{bmatrix}$$

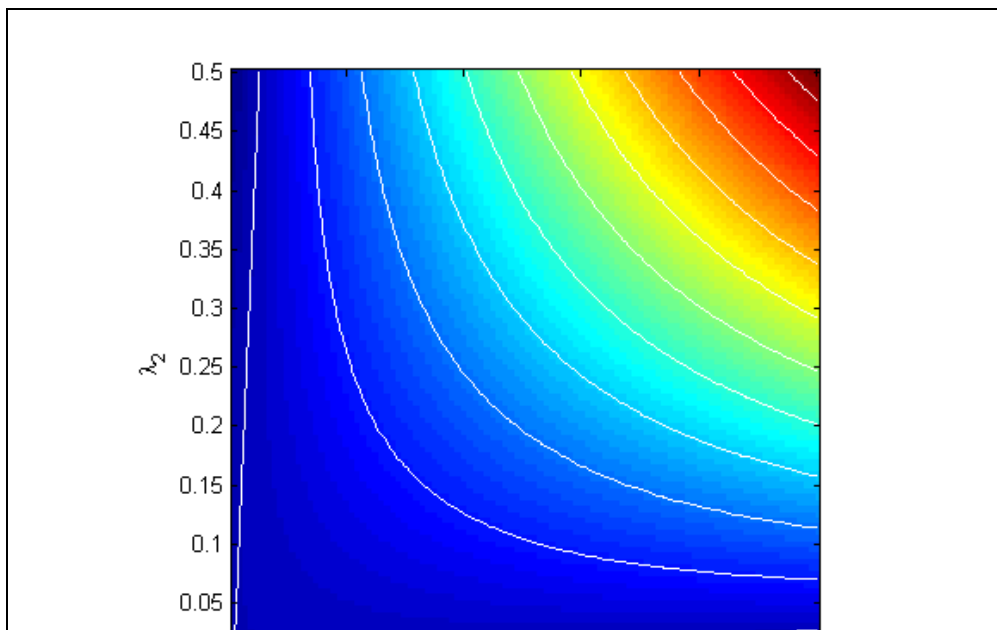
where * is the convolution operator. Using the windowing function $G(x, y; \sigma_i)$ matrices of outer products of gradients in the σ_i neighborhood are accumulated. The matrix of outer product has an eigenvector corresponding to the higher eigenvalue in direction of gradient. By accumulating the outer product using convolution and finding the eigenvalues of auto-correlation matrix we can find two dominant orientations of gradient and its size in a point neighborhood. Harris and Stephens [http://en.wikipedia.org/wiki/Corner_detection#The_Harris_26_Stephens_2F_Plessey_corner_detection_algorithm] realized this and designed an elegant function.

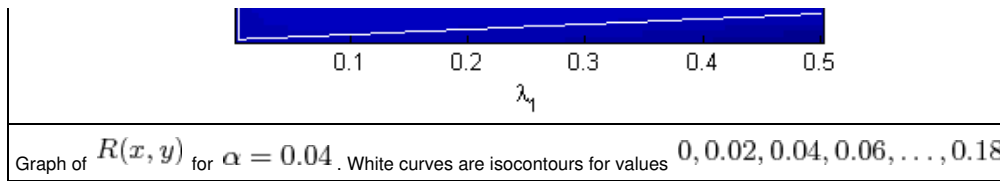
$$R(x, y) = \det(\mathbf{C}) - \alpha \text{trace}^2(\mathbf{C}).$$

which avoids the computation of eigenvalues λ_1, λ_2 of matrix \mathbf{C} thanks to the equivalency

$$\begin{aligned} \det(\mathbf{C}) &= \lambda_1 \lambda_2 \\ \text{trace}(\mathbf{C}) &= \lambda_1 + \lambda_2 \end{aligned}$$

to find properties of their ratio.





The image shows the function $R(x, y)$ for $\alpha = 0.04$, where white lines show iso-contours with values 0, 0.02, 0.04, 0.06, ..., 0.18. When we want to have values greater than a threshold, we want the smaller of the two eigenvalues to be greater than the threshold. We also want to have the value of the greater eigenvalue greater if the smaller eigenvalue is smaller. In the case, when the both eigenvalues are similar, they can be smaller.

- **Write** a function `response=harris_response(img, σ_d , σ_i)`, which computes the response of Harris function for each pixel of input image `img`. For computation, use the derivative of Gaussian D_x, D_y with standard deviation σ_d and a Gaussian as window function with standard deviation σ_i .
- In the same way as with the detector of Hessian maxima, write a function `[x,y]=harris(img, σ_d , σ_i , thresh)`, which detects the maxima of Harris function which are greater than `thresh`, using standard deviation σ_d^2 for derivatives and σ_i^2 for the window function.

n

Scale-space - automatic scale estimation

The basic version of Harris or Hessian detector needs one parameter (standard deviation), the scale on which it estimates gradients in the image and detects “blobs”. It can be shown that the scale can be estimated automatically for each image. For this purpose, it is necessary to build the “scale-space” – a three dimensional space, in which two dimensions are x, y -coordinates in the image and the third dimension is the scale. The image is filtered to acquire its new versions corresponding to increasing scale. Suppressing the details simulates the situation when we are looking at the scene from greater distance.

With increasing suppression of details the variance of intensities is decreasing. Therefore selection of characteristic scale is necessary to have comparable gradients between two scales. The term “normalized derivative” (considering the “distance” between pixels) was introduced for this purpose to obtain a “scaleless” gradient.

$$\frac{\partial f}{\partial \xi} = \frac{\partial f}{\partial (x/\sigma)} = \sigma \frac{\partial f}{\partial x}, \quad N_x(x, y; \sigma) = \sigma D_x(x, y; \sigma), \quad N_{xx}(x, y; \sigma) = \sigma^2 D_{xx}(x, y; \sigma),$$

The normalized Hessian matrix is thus:

$$\mathbf{H}_{norm}(x, y, \sigma_i) = \sigma^2 \begin{bmatrix} D_{xx}(x, y; \sigma) & D_{xy}(x, y; \sigma) \\ D_{xy}(x, y; \sigma) & D_{yy}(x, y; \sigma) \end{bmatrix}.$$

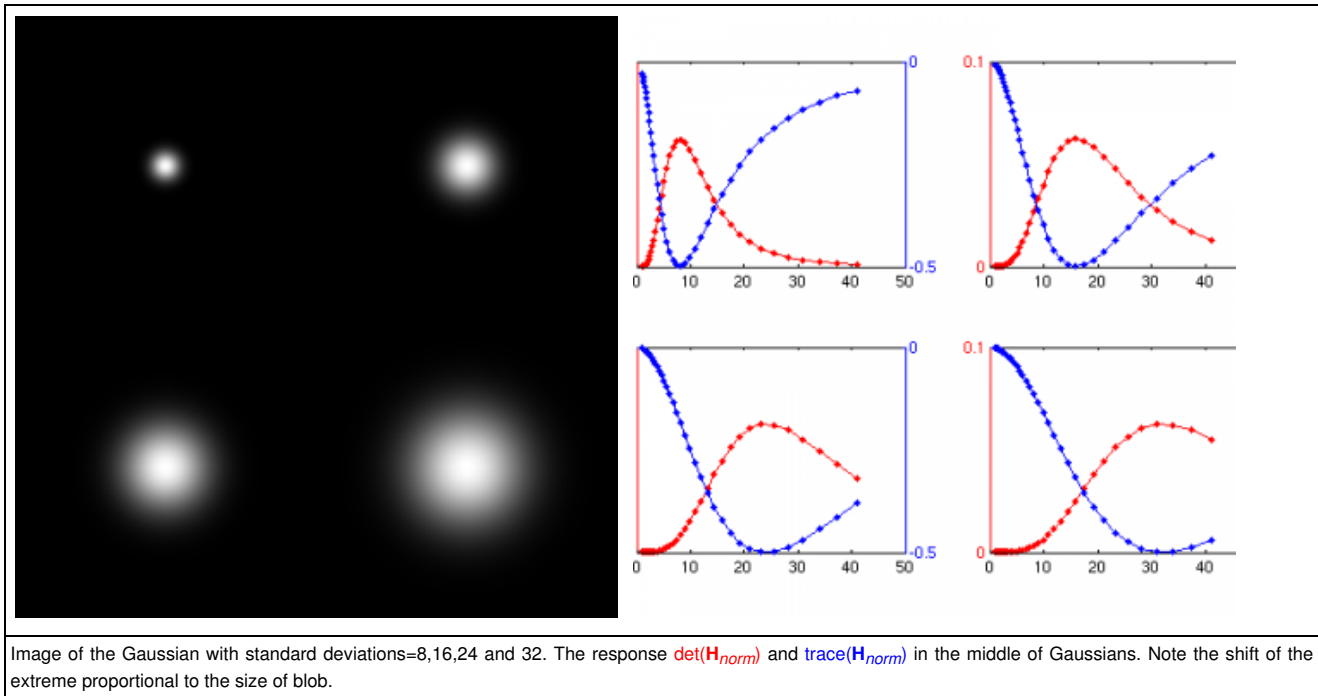
Lindeberg in his work showed that for such normalized derivatives it is possible to compute the response of differential Laplacian operators

$$\text{trace}(\mathbf{H}_{norm}(x, y, \sigma_i)) = N_{xx}(x, y; \sigma_i) + N_{yy}(x, y; \sigma_i)$$

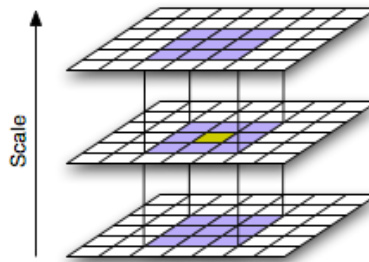
and Hessians

$$\det(\mathbf{H}_{norm}(x, y, \sigma_i)) = N_{xx}(x, y; \sigma_i)N_{yy}(x, y; \sigma_i) - N_{xy}^2(x, y; \sigma_i)$$

and use it for automatic learning of characteristic scale of ideal blob. After applying these operators, the local maxima of the image will give us x, y -coordinates and a scale σ of the blobs.



- Write a function `[ss,sigma]=scalespace(img,levels,step)`, which returns a 3D scale-space (matrix $H \times W \times L$, where H is height, W is width and L is number of *levels*), where `ss(:, :, 1)` will be input image *img*, for which we will assume $\sigma_1 = 1$ and `ss(:, :, i)` will be its filtered version $S(x, y, \sigma_{i+1})$, where $\sigma_{i+1} = \text{step} \cdot \sigma_i$ for $i \in \{1, \dots, \text{levels}\}$. Vector *sigma* will contain values σ_i .
- Write a function `maximg=nonmaxsup3d(response, threshold)` for non-maxima suppression in a 3D matrix *response* (i.e. take to consideration all 26 points in the neighborhood). The function returns a matrix of the same size as input matrix *response*, in which there will be 1 in places of local maxima which are greater than *thresh* and 0 elsewhere.



- Write a function `[hes,sigma]=sshessian_response(img)`, which will use function `scalespace` and compute Hessian response for normalized derivation of Gaussian. Choose the parameters *step* and *levels* of function `scalespace` wisely (e.g. *step*=1.1, *levels*=40). For computation of the response use estimation of the second derivatives which comes from applying of differential filters to the result of function `scalespace`. Normalize the derivatives with a standard deviation for each order of derivation (as described earlier). The result will be a 3D matrix *hes* with elements corresponding to normalized response of Hessian in scale-space.
- Join functions `nonmaxsup3d` and `sshessian_response` to detector of Hessian maxima with automatic scale estimation `[x,y,s]=sshessian(img, thresh)`. Return only points with response greater than *thresh*. The result will be vectors of the same length for position *x*, *y* and scale *s* of thresholded Hessian maxima in the space-space. Top left point of the image has coordinates (0,0). You can use the following code for localization of points:


```
% get the matrix of maxima
maximg=nonmaxsup3d(response, thresh);
% find positions
[y x u]=ind2sub(size(maximg), find(maximg));
% change coordinates system to zero-based
x=x-1; y=y-1;
% change u to scale s
...
```

Test your functions on this image:



For visualization of detected points use the function showpts.m:

```
% read an image
img=imread('sunflowers.png');
% finds Hessian maxima in scale-space, use threshold 0.02
[x,y,s]=sshessian(im2double(rgb2gray(img)), 0.02);
% show result
imshow(img);
p.linewidth=2; p.color='red';
showpts([x;y;s], p);
```

You should obtain an output similar to this:



Affine covariance, Baumberg iteration

(optional study material for interested students)

The detection of similarity-covariant points, as maxima of Hessian in scale space, can be extended to affine-covariant points to an unknown rotation. It is based on the so called Baumberg iteration, the idea is to observe the distribution of gradients in the vicinity of the detected point. Assuming that there is a transformation, which “flattens” the intensity in the point's neighborhood in such a way that the distribution of gradients would be isotropic (with gradients distributed equally in all directions), the transformation can be found using the second moment matrix of gradients, i.e. the autocorrelation matrix of gradients that we already know.

$$\mathbf{C}(x, y; \sigma_d, \sigma_i) = G(x, y; \sigma_i) * \begin{bmatrix} D_x^2(x, y; \sigma_d) & D_x D_y(x, y; \sigma_d) \\ D_x D_y(x, y; \sigma_d) & D_y^2(x, y; \sigma_d) \end{bmatrix}$$

As we have said earlier, this matrix reflects the local distribution of gradients, we can show that when we find a transformation $\mu = \mathbf{C}^{-1/2}$ that translates the coordinate system given by eigenvectors and eigenvalues of matrix \mathbf{C} to a canonical one, the distribution of gradients in the transformed matrix μ will be “more isotropic”. However, since we used a circular window function it is possible that for some gradients around, the weight will be calculated wrongly, or not at all. Therefore, it is necessary to repeat this procedure until the eigenvalues of matrix \mathbf{C}_i from the i -th iteration of the point's neighborhood will be close to identity (a multiplication of identity). We'll find out by comparing the ratio of eigenvalue of matrix \mathbf{C}_i . The resulting array of local affine deformation is obtained as the total deformation of the neighborhood needed to “become” isotropic:

$$N = \prod_i \mu_i$$

This transformation leads (after the addition of translation and scaling) from the image coordinates to the canonical coordinate system. In practice, we are mostly interested in the inverse transformation $\mathbf{A} = \mathbf{N}^{-1}$.

- Write a function `[x, y, A11, A12, A21, A22] = affinehessian(img, Threshold)`, which extends the detector of Hessian maxima from the previous step, and for each maximum in the scale space it calculates one step of the Baumberg iteration. The function returns the position of a point and the 2x2 submatrix of matrix \mathbf{A} :

$$A = \sigma_i \text{inv} \left(C^{-1/2} / \sqrt{\det(C^{-1/2})} \right)$$

Maximal Stable Extremal Region (MSER)

The detector of Maximal Stable Extremal Regions is based a different idea. Detection is based on growing regions in a binary thresholded image while increasing the threshold for image intensity. While increasing intensity, new regions appear in the image, they are merging together and at the end there is one region with area of the whole image. During the region growing, the region statistics (area and the border length) are monitored. The detector finds such intensity ranges where the ratio of the area and border length changes the least. The MSER detector is implemented as a MEX module. It is used as follows:

```
p.min_margin = 10;
p.min_size   = 30;
mser = extrema(img, p, [1 2]);
```

MSERs are affine co-variant regions. It is possible to visualize their covariance matrix as an ellipse with the longer axis corresponding to the maximal variance of elements. We can use following code for conversion into elliptical regions:

```
regs=[mser{1}{2,:} mser{2}{2,:}];
x=[regs.cx]; y=[regs.cy];
a11=sqrt([regs.sxx]); a12=zeros(size(a11));
a21=[regs.sxy]./a11; a22=sqrt([regs.syy] - a21.*a21);
imshow(img); showpts([x;y;a11;a12;a21;a22]);
```

For visualisation download `showpts.m`.

The Dataset

To try your new detectors, take pictures of three objects from five different viewpoints with increasing complexity. Two pictures should contain a simple rotation, with and without scale change. The other pictures should be taken from different viewpoints. Name the images with 1024px on the longer side as obj(0-2)_view(0-4). Pack them to a .zip archive and upload them to upload system (task 02_data). If necessary (due to long runtime of your code) you can shrink the images to half.

What should you upload?

You are supposed to upload functions `hessian_response.m`, `hessian.m`, `harris_response.m`, `harris.m`, `nonmaxsup2d.m`, `scalespace.m`, `nonmaxsup3d.m`, `sshessian_response.m` a `sshessian.m` together with all used non-standard functions you have created. Do not forget the dataset.

Testing

We use a script and the MATLAB function 'publish' to test your code. Download `detect_test.zip` and unpack it to a directory which is in MATLAB paths (or put it into the directory with your code) and execute. Compare your results with ours [http://cmp.felk.cvut.cz/~perdom1/mpv/02_detect/test.html].

References

1. Overview of the most common used methods for feature point detection. [[http://en.wikipedia.org/wiki/Feature_detection_\(computer_vision\)](http://en.wikipedia.org/wiki/Feature_detection_(computer_vision))]
2. Foundations of scale-space. [<http://www.bmia.bmt.tue.nl/education/courses/FEV/book/pdf/02%20Foundations%20of%20scale-space.pdf>]
3. Scale-space, the basic method for feature point detection independent on the scale. [<http://en.wikipedia.org/wiki/Scale-space>]
4. Affine shape adaptation. [http://en.wikipedia.org/wiki/Affine_shape_adaptation]
5. Overview of the affine detectors. [<http://www.robots.ox.ac.uk/~vgg/research/affine/>]

2. Computing local invariant description











To compute a local description invariant to geometric and photometric transformations, the neighborhood of the feature points needs to be normalized to undo the effects of transformations. After elimination of these deformations, the invariant description can be computed.

As we already know, the task of the detector is to repeatedly find the shape of the neighborhood of feature points in such a way that the detected points are co-variant with a desired class of transformations (geometric or photometric). For instance, the basic version of Harris or Hessian detector detects points which are co-variant with translation (if you shift the image, points are shifted too). Hessian in the scale-space or DoG detector adds information about the scale of the region and therefore points are co-variant with similarity transformation up to an unknown rotation (detected points have no orientation assigned). Affine co-variant detectors like MSER, Hessian-Affine or Harris-Affine are co-variant up to an affine transformation. The following text describes how to handle geometric information from the point neighbourhood and how to use it for normalization. Depending on the amount of information that a detector gives about a features, the descriptions can be invariant to translation, similarity, affine or perspective transformation.

Geometric normalization

Geometric normalization is a process of geometric transformation of feature neighborhood into a canonical coordinate system. Information about geometric transformation will be stored in the form of "frame" – a projection of the canonical coordinate system into the neighborhood of a feature point (or region) in the image. The frame will be represented by a 3x3 matrix A , the same as in

the [first lab](#). The transformation matrix A is used to obtain a “patch” – a small feature neighborhood in the canonical coordinate system. All other measurements on this square patch of original image are now invariant to desired geometric transformation.

Original image	Translation	Translation and rotation	Similarity	Affine transformation
				
				

Example of geometric normalization, with different classes of transformation (author of images: Štěpán Obdržálek).

For the construction of a transformation A , we use the geometric information about position and shape of point neighborhood:

1. For rotation- and translation-covariant transformation, we have coordinates x, y and angle α . The scale s (size of the frame) is fixed (manually) for all points.

$$A = \underbrace{\begin{bmatrix} s & 0 & x \\ 0 & s & y \\ 0 & 0 & 1 \end{bmatrix}}_{A_{norm}} \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. For similarity-covariant point, we have coordinates x, y scale σ and angle α .

$$A = \underbrace{\begin{bmatrix} \sigma & 0 & x \\ 0 & \sigma & y \\ 0 & 0 & 1 \end{bmatrix}}_{A_{norm}} \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. For affine-covariant point, we can use three points to which the points from the canonical coordinate system are projected (as in the [first lab](#)). We can also use a known position, partial affine transformation (a 2×2 submatrix) and the residual rotation (angle α) to construct the matrix A :

$$A = \underbrace{\begin{bmatrix} a_{11} & a_{12} & x \\ a_{21} & a_{22} & y \\ 0 & 0 & 1 \end{bmatrix}}_{A_{norm}} \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Many detectors (all from the last lab) detect points (frames), which are similarity- or affine- covariant up to an unknown rotation (angle α). For instance, the position and scale give us a similarity co-variant point up to unknown rotation. Similarly, the matrix of second moments and the centre of gravity give us five constraints for an affine transformation and only the rotation remains

unknown. To obtain a fully similarity- or affine-covariant point, we need to define orientation α from a partially geometrically normalized point neighborhood.

The normalization including orientation estimation has to be done in two steps. In the first step, the patch invariant to translation, scale and partial affine transformation is created. On this normalized patch, the dominant gradient is estimated. Both transformation are then used together in the transformation matrix A .

To estimate the orientations of the dominant gradients on the partially normalized regions (using A_{norm}), the gradient magnitudes and orientations are estimated for each region and a histogram of these gradients is computed. While we assume a random rotation of the region, it is necessary to compute gradients only in circular neighborhood. This can be done by weighting with a window-function (e.g. Gaussian) or with a condition like $(x - x_{center})^2 + (y - y_{center})^2 < (ps/2)^2$, where ps patch edge length. Otherwise the content of the corners would influence the orientation estimate undesirably. The orientations of gradients are weighted by their magnitudes. This means that a "stronger" gradient brings more to the corresponding bin of the histogram. For improved robustness, we can use linear interpolation to vote into the closest neighboring bins. At the end, the histogram is filtered with a 1D Gaussian and the maximum is found. For a more precise localization it is possible to fit a parabola into the maximum neighbourhood and to find the orientation with a precision over $360/(\text{number of bins})$ degrees.

- Write a function `angle=dom_orientation(img)` for dominant orientation estimation in a normalized patch. The input `img` is a partially normalized patch of the image ($ps \times ps$ matrix of doubles), the output is the `angle` measured from the x -axis of the image in clock-wise direction (angle for vector (x,y) can be computed with function `atan2(y,x)`).

Now we are able to write the functions for geometric normalization of feature point neighborhoods with dominant orientations:

- for a known position, write a function `pts=transnorm(img,x,y,s,opt)`, where `img` is the input image, x,y are row vectors of point coordinates and s is a scalar with the fixed point scale (the same value for all feature points).
- for a known position and scale, write a function `pts=simnorm(img,x,y,s,opt)`, where `img` is the input image, x,y are row vectors of point coordinates and s is a row vector with the scales of all points (see the output of the `sshesian` function).
- for a known position and partial affine transformation, write a function `pts=affnorm(img,x,y,a11,a12,a21,a22,opt)`, where `img` is the input image, x,y are row vectors of points coordinates and $a11,a12,a21,a22$ are row vectors with the partial affine transformation elements.

Input parameter `opt` is a structure containing normalization parameters:

```
-----
opt.ps - size of the patch (edge length)
opt.ext - size of the neighborhood in the canonical coordinate system
-----
```

The output `pts` is a one-dimensional array of structures, containing for each point: coordinates - X,y , elements of the 2×2 partial affine transformation submatrix - `a11,a12,a21,a22`, and the normalized image patch ($ps \times ps$ matrix of type double) - `patch`. Use the hierarchy of these functions at implementation - a simpler function can be used by more complex ones. The resulting patch and the patch for dominant orientation estimation are computed using function `affinetr`.

Here's a pseudocode for normalization including orientation estimation:

```
-----
% ... for each point ...
A = create_matrix_A_without_rotation(x(id),y(id),...);
% orientation estimation
tmp = affinetr(img, A, opt.ps, opt.ext);
angle = dom_orientation(tmp);
% final A, dominant orientation to angle zero
R = rotation_2x2(-angle); A(1:2,1:2) = A(1:2,1:2)*R;
% create output
pts(id).x=x(id); pts(id).y=y(id);
pts(id).a11=A(1,1);
...
pts(id).patch=affinetr(img, A, opt.ps, opt.ext);
-----
```

For the sake of clarity only the orientation of the strongest gradient is used for normalization in this task.

Photometric normalization

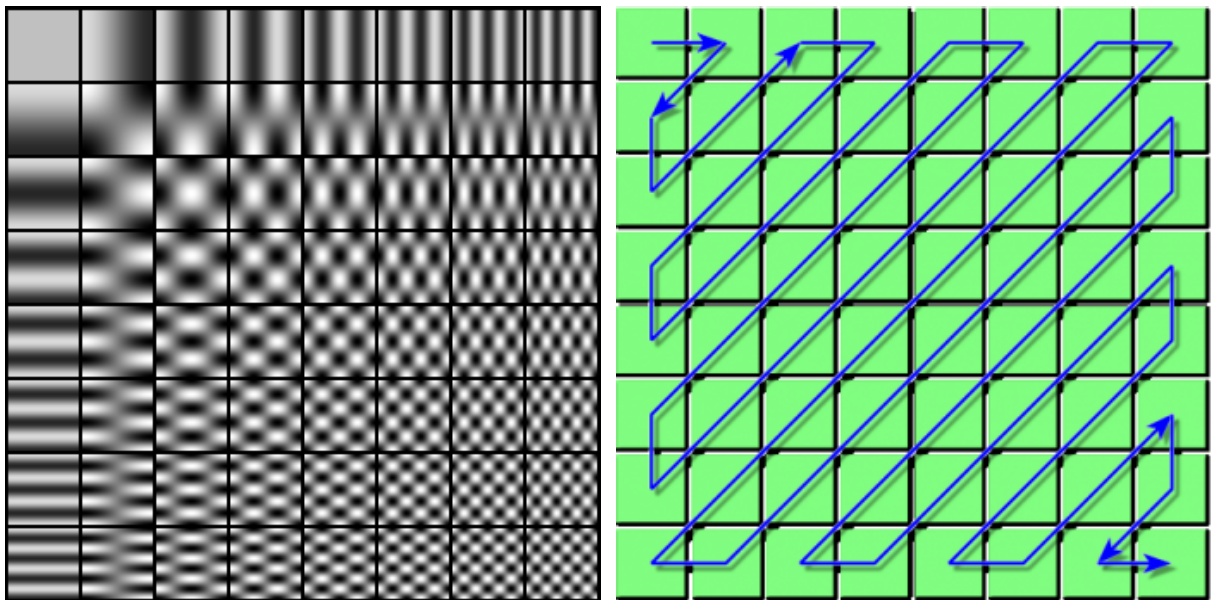
The goal of photometric normalization is to suppress the scene changes caused by illumination changes. The easiest way of normalization is to expand the intensity channel into whole intensity range. We can also expand the intensity channel such that (after transformation) the mean intensity is at half of the intensity range and the standard deviation of intensities σ corresponds to its range (or better $2 \times \sigma$). In case we want to use all three color channels, we can normalize each channel separately.

- Write a function `ptsn=photonorm(pts)`, which for each point in the array of structures `pts` photometrically normalizes the patch of the image (field `patch`) such that the mean intensity value will be 0.5 and the standard deviation will be 0.2. Values outside the range $<0,1>$ will be set to 0 or 1 respectively. Function returns an array of structures `ptsn` similar to the input array of structures `pts`, with modified patches in the fields `patch` and with the original mean values and standard deviations in fields `mean` and `std`.

Computing the description

On the geometrically and photometrically invariant image patch from the last step, any low dimensional description can be computed. It is clear that the process of normalization is not perfect and therefore it is desirable to compute a description which is not very sensitive to the residual inaccuracies. The easiest description of the patch is the patch itself. Its disadvantages are high dimensionality and sensitivity to small translations and intensity changes. Therefore we will try better descriptions too.

1. A description using the two-dimensional discrete cosine transformation (DCT). Coefficients of this integral transformation are computed by the dot product of image the patch with the so called DCT base functions. See image:



The image patch is decomposed into individual “frequencies” in a way similar to the JPEG compression. Similarly to image compression, we chose a subset of these coefficients containing information about lower frequencies. Discarding higher frequencies brings robustness to high frequency noise and to small errors in geometric normalization. It can be seen from the image of base functions that the choice of DCT coefficients will be done in so called zig-zag order, where the higher frequencies are added after the lower ones. [http://en.wikipedia.org/wiki/JPEG#Entropy_coding]

- Write a function `dct=dctdesc(img,num_coeffs)` computing the DCT transformation for input image patch `img` and returns `num_coeffs` coefficients in zig-zag order in variable `dct`. Normalize the values such that all possible DCT coefficients belong to $<0,1>$. You can make use of the MATLAB function `dct2` [<http://www.mathworks.com/help/images/ref/dct2.html>].

2. **(optional task)** RGB histogram, RG histogram and the histogram of gradients are descriptions using global image patch properties. Color channels of the image (e.g. R, G, B) are quantized into bins after pre-processing and their characteristics are then accumulated. The resulting description is a linear vector of values accumulated into bins. The so called RG chromacity [http://en.wikipedia.org/wiki/Rg_chromaticity] color space has two components

$$R = \frac{R}{R+G+B} \quad G = \frac{G}{R+G+B},$$

it is a ratio of red and green component in the image. The advantage of image patch histograms is their insensitivity to geometric normalization errors, the disadvantage is a lower distinctness.

- Write a function `dx dy=ghistodesc(img,num_bins)` computing the histogram of gradients, `num_bins` is the number of bins per channel, so the resulting `dx dy` description is a column vector with `num_bins2` elements of the histogram (size of the gradient in `dx` - row, `dy` - column) joined by columns (matlab operator `(:)`). Normalize the elements into range `<0,1>` (0 - no occurrence, 1 - occurs in all points of the patch).

What should you upload?

You are supposed to upload functions `transnorm.m`, `simnorm.m`, `affnorm.m`, `photonorm.m`, `dom_orientation.m`, `dctdesc.m` together with all used non-standard functions you have created. Do not forget the dataset.

Testing

To test your code, copy `desc_test.zip` and unpack it to directory in MATLAB paths (or put it into the directory with your code) and execute. Compare your results with ours [http://cmp.felk.cvut.cz/~perdom1/mpv/03_desc/test.html].

References

- David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110. [<http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf>]
- A performance evaluation of local descriptors [http://lear.inrialpes.fr/pubs/2005/MS05/mikolajczyk_pami05.pdf].

3. Tentative correspondences and RANSAC

Preliminaries

Before we start, we'll need a function `pts=detect_and_describe(img,detpar,descpar)` connecting the previously implemented functions together, such that it detects, geometrically and photometrically normalizes and describes feature points in the input image `img`. Detector parameters will be passed in structure `detpar`.

- for Hessian detector::

```
detpar.type='hessian';
detpar.sigma % sigma for derivatives
detpar.threshold
```

- for Harris detector:

```
detpar.type='harris';
detpar.sigmad % sigma for derivatives
detpar.sigmai % sigma for integration
```

```
detpar.threshold
```

- for Hessian detector in scale-space

```
detpar.type='sshessian';
detpar.threshold
```

- for MSER detector

```
detpar.type='mser';
detpar.min_margin % requested stability
detpar.min_size % area of the smallest region (in pixels)
detpar.max_area % area of the biggest region (in relation to image area, in range <0, 1>)
```

Parameters for normalization and description will be passed in structure *descpar*

```
descpar.type % type of the description ('dct' nebo 'ghisto' nebo 'sift')
descpar.ps % size of patch
descpar.ext % size of feature point neighborhood in canonical coordinate system
```

extra for DCT descriptor:

```
descpar.num_coeffs % number of coefficient in zig-zag order
```

extra for ghisto:

```
descpar.num_bins % number of bins in one ax of histogram
```

The structure *pts* on the output is in the format of function *affnorm* (array x,y,a11,a12,a21,a22,patch) and *photonorm* (mean,std) from last labs, for each point adds array *pts(i).desc* with the description as a column vector. You can find function *detect_and_describe* here.

Tentative correspondences

After detection and description, which run on each image of the scene separately, we need to find correspondences between the images. The first phase is looking for the tentative correspondences (sometimes called “matches”). A tentative correspondence is a pair of features with descriptions similar in some metric. From now on, we will refer to our images of the scene as the left one and the right one. The easiest way to establish tentative correspondences is to find all distances between the descriptions of features detected in the left and the right image. We call this table a “distance matrix”. We will compute the distance in Euclidean space. Let's try several methods for correspondence selection:

1. Tentative correspondences of **mutually nearest** descriptions are created, when the nearest description in the left image for description A from right image is description B and at the same time, description A is the closest description in the right image to description B from the left image.
2. For **stable pairing** in each step the algorithm finds globally closest descriptions A and B. It creates tentative correspondence and descriptions A and B are excluded from finding in next steps (for instance with setting infinity value to right place in distance matrix). The iterations continues while there are any active points left (values smaller than infinity).
3. A different procedure is to find the two closest descriptions in the right image for each description from the left image and then test, if the ratio of **first/second closest** description is smaller than a threshold (use e.g. 0.8). If so, the descriptions correspond and it is guaranteed, that there are no confusing descriptions which would be too close in the description space. This method can be used for well discriminative description - for instance SIFT.

The output of finding tentative correspondences are pairs of indices of features from the left and the right image with the closest descriptions.

- write a function `corrs=match(pts1, pts2, par)`, which finds correspondences between points `pts1` and `pts2` with method specified in the `par` structure. Description of each point i is saved in array `pts1(i).desc` resp. `pts2(i).desc`. The method is specified in the field `par.method` as a string. For each method, ignore tentative correspondences with distance greater than threshold `par.threshold`.

```
par.method = 'mutual'; % for mutual nearest method
            'stable'; % for stable pairing
            'sclosest'; % for "first/second closest" method
par.threshold % threshold, pairs with distance greater than this will be ignored
```

Output will be a 2xN matrix containing the indices of tentative correspondences, where the first row corresponds to indices in `pts1` and second row to indices in `pts2`.

RANSAC and the geometric model of the scene

The tentative correspondences from the previous step usually contain many non-corresponding points, so called outliers, i.e. tentative correspondences with similar descriptions that do not correspond to the same physical object in the scene. The last phase of finding correspondences aims to get rid of the outliers. We will assume images of the same scene and search a model of the undergoing transformation. The result will be a model of the transformation and a set of the so called inliers, tentative correspondences that are consistent with the model.

The model of a planar scene

The relation between the planes in the scene is discussed in the course Digital Image [https://cw.fel.cvut.cz/wiki/courses/ae4m33dzo/start]. In short:

The transformation between two images of a plane observed from two views (left and right image) using the perspective camera model is a linear transformation (called **homography**) of 3-dimensional homogeneous vectors (of corresponding points in the homogenous coordinates) represented by a regular 3×3 matrix \mathbf{H} :

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For each pair of corresponding points $(x_i, y_i)^\top \leftrightarrow (x'_i, y'_i)^\top$ it holds:

$$\begin{aligned} x'(h_{31}x + h_{32}y + h_{33}) - (h_{11}x + h_{12}y + h_{13}) &= 0 \\ y'(h_{31}x + h_{32}y + h_{33}) - (h_{21}x + h_{22}y + h_{23}) &= 0 \end{aligned}$$

Let us build a system of linear constraints for each pair:

$$\underbrace{\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x'_1x_1 & x'_1y_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & y'_1x_1 & y'_1y_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x'_2x_2 & x'_2y_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & y'_2x_2 & y'_2y_2 & y'_2 \\ \vdots & & & \vdots & & & & \vdots & \\ -x_n & -y_n & -1 & 0 & 0 & 0 & x'_nx_n & x'_ny_n & x'_n \\ 0 & 0 & 0 & -x_n & -y_n & -1 & y'_nx_n & y'_ny_n & y'_n \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

This homogeneous system of equations has 9 degrees of freedom and we get one non-trivial solution as a right nullspace of the matrix \mathbf{C} . For a unique solution we need at least 8 linearly independent rows of matrix \mathbf{C} , i.e. 8 constraints from 4 corresponding points in a general position (no triplet of points can lie on a line!).

- implement a function `H=u2h(u)` that gets a matrix of 4 corresponding pairs of points in homogeneous coordinates and computes the homography \mathbf{H} . In case that some of the triplets will lie close to a line, it returns an empty matrix []. The input u is a 6×4 matrix as shown below, where x_i, y_i are point coordinates in the left image and x'_i, y'_i in the right image. :

$$u = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \\ x'_1 & x'_2 & x'_3 & x'_4 \\ y'_1 & y'_2 & y'_3 & y'_4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- write a function `dist=hdist(H,u)`, which gets a $6 \times N$ matrix u of N pairs of points in homogeneous coordinates and returns a vector `dist` ($1 \times N$) of squared distances of the points $\mathbf{H}(x_i, y_i, 1)^\top$ and $(x'_i, y'_i, 1)^\top$ in **Euclidean** space.

RANSAC

To find a correct model of the scene – a projective transformation between two planes, we need at least 4 inlier point pairs (not in a line). One way of picking such points in a robust way from all the pairs of acquired tentative correspondences is the RANSAC algorithm of M.Fischler and R.C.Bolles.

In our task, the RANSAC algorithm will have this form:

1. Uniformly sample 4 corresponding pairs. You can use the provided function `sample.m`.
2. Compute the homography \mathbf{H} out of the sampled point pairs.
3. Compute the support of the found model. If the transformation \mathbf{H} was computed from an all-inlier sample, all inlier points from the left image (x_i, y_i) shall be projected by \mathbf{H} to the corresponding points (x'_i, y'_i) in the other image. The support of the model is quantified by the number of points consistent with the model, i.e. the number of correspondences with distance (function `hdist`) of transformed point from the left image and the corresponding point in the right image smaller than a predefined threshold (e.g. 1-3 pixels).
4. If the support of the model is the biggest so far, we keep this so-far-the-best homography \mathbf{H}^* and the corresponding set of inliers, i.e. points that supported this model.
5. We iterate points 1-4 and keep track of so-far-the-best \mathbf{H}^* transformation.

Set the number of iterations to 1000 in the beginning and debug the RANSAC algorithm on a simple pair of images. A more sophisticated stopping criterion is used in practice. It is based on the probability estimate of finding an all-inlier sample (of size 4 in our case) under the assumptions of iteratively estimated fraction of inliers. The stopping criterium is implemented in function `nsamples.m`. It computes total number of iterations required for a given confidence `conf`:

```
num_samples = nsamples(number_of_inliers, number_tentative_correspondences, sample_size, conf);
```

where `number_of_inliers` is iteratively estimated at each observation of a new so-far-the-best model, i.e. for each new so-far-the-best model we recompute the number of steps required by `nsamples`.

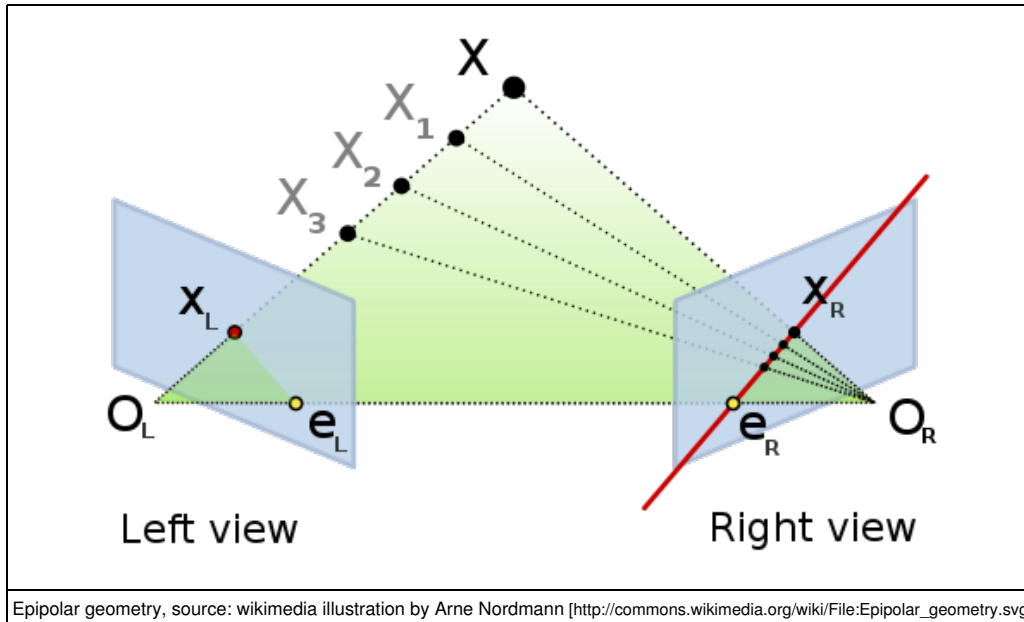
- use the provided functions to implement a function `[Hbest,inl]=ransac_h(u,threshold,confidence)` that robustly estimates the homography \mathbf{H}^* and a set of inliers `inl`. Input u is a $6 \times N$ matrix of corresponding points in homogeneous coordinates (similar to function `hdist`). The parameter `threshold` is the maximum distance of reprojected points (computed via `hdist`) that will be

marked as inliers (pairs consistent with the model \mathbf{H}^*). The parameter *confidence* is a value from the range (0,1), the requested confidence of getting a correct sample (a good value for testing is for example 0.95).

The output is the best model \mathbf{H}^* and an array *inl* (1xN), where 0 represents an outlier and 1 represents an inlier.

Position of two perspective cameras, epipolar geometry

To verify the tentative correspondences for other than planar scenes, we will need more general relation between two points from two cameras. The geometric relation of a pair of uncalibrated cameras is called the epipolar geometry. The epipolar geometry is discussed in the course of Geometry of Computer Vision and Graphics [https://cw.fel.cvut.cz/wiki/courses/a4m33gvg/start], for an initial information you can read the corresponding wikipedia page [http://en.wikipedia.org/wiki/Epipolar_geometry] or listen to a dedicated song :) [http://danielwedge.com/fmatrix/]. For our problem, it is important that the epipolar geometry is a geometric relation of corresponding points x_L and x_R , images of a physical point X in the scene observed by a pair of perspective cameras:



This relation can be written as:

$$x_L^T \mathbf{F} x_R = 0$$

The epipolar geometry is represented by a matrix \mathbf{F} , called the fundamental matrix. To find a fundamental matrix we need at least 7 corresponding points. Our RANSAC implementation from the previous section has to be modified in two ways: We'll need a function `u2f7.m` that will replace the estimation of homography by estimation of the fundamental matrix \mathbf{F} and a function `fds.m` that will replace the function `hdist` that estimates the distance of the reprojected points, here the distance of the corresponding point from an epipolar line (image of the ray through the point in the other camera). These functions have the same parameters as `u2h` and `hdist`, so you can just replace matrix \mathbf{H} by matrix \mathbf{F} . Function `u2f7` returns from one (result is a 3x3 matrix) up to three solutions (result is a 3x3xnumber_of_solutions matrix), thus it is required to modify the verification of the model to pick the best of returned solutions.

- use the provided functions to implement a function `[Fbest,inl]=ransac_f(u,threshold,confidence)` that estimates the fundamental matrix \mathbf{F}^* and a set of inliers *inl*. Input *u* is a 6xN matrix of corresponding points in homogeneous coordinates. Parameter *threshold* defines the maximal distance of points for function `fds` to be labeled as inliers, consistent with the epipolar geometry given by \mathbf{F}^* . The parameter *confidence* is value from the range (0,1), requested probability of getting a correct sample (for testing use the value 0.95).

The result will be the best model found \mathbf{F}^* and an array *inl* (1xN), where 0 represents an outlier and 1 represents an inlier.

What should you upload?

Implement functions `match.m`, `u2h.m`, `hdist.m`, `ransac_h.m`, `ransac_f.m` and submit them in task **04_corr** together with all non-standard functions you have created.

Choose a combination of detector/description for each object in your dataset. Run the detectors and generate descriptions for all images using function `detect_and_describe`. Save the results (structure `pts`) into file `obj%d_view%d.mat` together with your configuration of the detector (`detpar`), normalization and description (`descpar`). Objects and views numerate from 0 (e.g. for object 1 image 3 save the results: structures `pts`, `detpar` and `descpar` into file `obj1_view3.mat`, for object 0 image 0 in file `obj0_view0.mat`, etc.).

Find tentative correspondences for each object and all pairs of views to the object. The result will be a matrix `TC` of cells 5×5, where cells will contain arrays `corrs` from function `match`. We will assume, that the process is symmetric ($2 \rightarrow 1$ is same as $1 \rightarrow 2$) and generate only upper triangle (index of first image as row and second image as column) without diagonal. Leave other cells empty. Save matrix `TC` together with configuration `par` of the method `match` into file `obj%d_tc.mat`.

At the end run both RANSACs on generated tentative correspondences for each object and save the results into matrices `H`, `F`, `inlH` and `inlF` of cells 5×5. Save the cell matrices `H`, `F`, `inlH` and `inlF` into file `obj%d_results.mat` together with parameters `thresh_h`, `thresh_f`, `conf_h`, `conf_f` of function `ransac_h` and `ransac_f`.

Pack the all results into an archive together with your images from task **02_data named `(username)_obj(0-2)_view(0-4).jpg`; and submit into task **04_results**. To save some space you can remove the field `patch` from structure `pts` using function `rmfield`.**

Testing

To test your code, download `corr_test.zip` and unpack it into directory in MATLAB paths (or put it into directory with your code) and execute. Compare your results with ours [http://cmp.felk.cvut.cz/~perdom1/mpv/04_corr/test.html].

[courses/ae4m33mpv/labs/2_correspondence_problem/start.txt](http://cw.felk.cvut.cz/wiki/courses/ae4m33mpv/labs/2_correspondence_problem/start.txt) · Last modified: 2015/07/02 18:23 by matas