Exercise 2: Interest Points and Descriptors

In this exercise you will implement an interest point detector, the Harris corner detector. You will then compute descriptors at each of the interest points. These descriptors will then be used to match similar interest points from consecutive image frames with each other.

[Download files for exercise 2]

Part 1: Implementation (10 points)

You should write code so that it can be understood by someone else (the tutors, for example). Comment sections and lines which might not be easy to understand. If you've thought about a line or section for more than a couple of minutes, it probably warrants a comment. Use descriptive variable names.

Your code does not have to run extremely fast, but it should not take too long to run. Points may be deducted for slow code, even if your solution is correct.

1. Harris corner detector (4 points)

In the file **harris_corner.py** you should write code for the Harris corner detector presented in the lecture. This function is called from the `main_harris_image.py` script.

The input parameters are a grayscale image with values in the range of [0,1] and the following parameters:

- `k` : Coefficient of the Harris formula
- `threshold` : Threshold for corners
- `sigma1` : Sigma of Gaussian filter used in the first step of the algorithm
- `sigma2` : Sigma of Gaussian filter used after the calculation of the derivatives in x and y direction (should be larger than `sigma1`)

For debugging purposes, the result of every intermediate step should be returned as a parameter. For a detailed description of the output parameters refer to the comments in the file.

The basic steps you will have to implement are:

1. Compute the horizontal and vertical derivatives of the image (I_x and I_y) by convolving the image I with derivatives of Gaussians in x- and y-direction. You can create the Gaussians as in exercise 1, using `sigma1` or you can use these commands:

   ```
   kernel = cv2.getGaussianKernel(kernel_width_dog, sigma1)  # 1x5
   gauss = np.outer(kernel, kernel.transpose())  # 5x5
   ```

   The width of the filter can be calculated like you did in exercise 1. To get a Gaussian derivative you can filter a Gaussian kernel with the small filter [-1 0 1], or use the `np.gradient` command.

2. Compute the Harris feature value for each pixel and normalize the values to have a maximum value of 1. Use another Gaussian (with `sigma2`) as the weighting kernel.

3. Apply non-maximum suppression and threshold the feature values. You can adapt your non-max suppression from the previous exercise or use a built-in function.

Complete the code in **harris_corner.py**. To test your implementation you can call the script `main_harris_image.py`.

2. Patch descriptor (1 points)

Once the corner detector is complete, we want to match corners in similar images to find correspondences. To be able to do that, you first have to implement a feature descriptor characterising each detected corner. For the basic descriptor, take the image intensity values around a corner and concatenate them to get a large feature vector (this is a one-line operation).

The `compute_descriptors` function is a helper function which passes a valid image patch to a descriptor function, ensuring that descriptors are not computed on patches which would go outside the bounds of the image.

Complete the code for the functions `patch_basic` and `compute_descriptors` in **descriptor_functions.py**.

3. Matching interest points (1 points)

Now that we have a feature vector describing each interest point, we can try to match similar

interest points across different images. The matching function has to be implemented in the file **match_descriptors.py**, which takes the feature descriptors of two images as input parameters. To test this function, you can call the script `main_descriptors_image.py`.

For each descriptor $\mathbf{x}_A$ of image A, you have to calculate the distance to each descriptor $\mathbf{x}_B$ of image B. As a distance measure, you can simply use the euclidean distance:

$$d = |\mathbf{x}_A - \mathbf{x}_B|$$

The $\mathbf{x}_B$ with the smallest distance value corresponds to the best match for $\mathbf{x}_A$. Some descriptors can be very ambiguous and cause false matches. To reduce the number of false matches, you should add the following heuristic: Consider the two "closest" matches $\mathbf{x}_{B1}$ and $\mathbf{x}_{B2}$ to $\mathbf{x}_A$ with distances d_1 and d_2. The descriptor $\mathbf{x}_{B1}$ should only be considered as the unique best match for $\mathbf{x}_A$ if $\frac{d_1}{d_2} < 0.8$. That way, many ambiguous and probably wrong matches will be discarded.

An example of a good matching result can be seen in Figure 1 below.

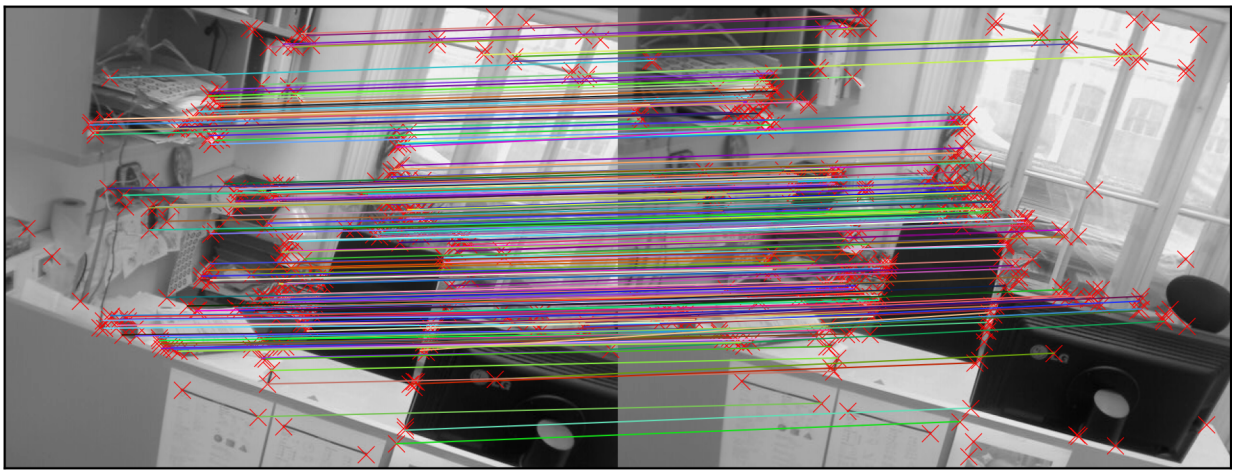

Figure 1: Good matching result

4. Better descriptors (4 points)

Extend the basic patch descriptor. These functions should not require more than 3 lines of code.

1. Normalise the descriptor in `patch_norm` (0.5p)
2. Sort the normalised descriptor in `patch_sort` (0.5p)
3. Extract pixels in a circular region of the patch, and sort them after normalising in `patch_sort_circle`. You can use the `circle_mask` function to help (0.5p)

Implement an orientation-based descriptor `block_orientations` (2.5p). This descriptor will operate on blocks of 4x4 pixels on a 16x16 image patch. Compute the gradient magnitude and orientation of each pixel as you did in the previous exercise, using the simple

filter [1 0 -1]. You do not need to apply additional blur to the patch. The final descriptor will be a concatenation of orientation histograms computed on the 16 4x4 blocks. Each block should produce a histogram with 8 bins in the range [-pi, pi]. Each bin should be weighted by the sum of the gradient magnitudes of the pixels in that bin. Normalise each block's histogram individually before concatenating it. The `np.digitize` function and/or functions for histograms should be helpful.

You can check if your block extraction is correct by operating on the following matrix:

```
np.kron(np.arange(16).reshape(4,4),np.ones((4,4))
```

Your blocks should come in row major order, i.e. be blocks of all 1s, 2s, 3s and so on (as opposed to 1,5,9... for column major order). The descriptor should also follow this ordering.

Forbidden commands

`cv2.cornerHarris()`, `cv2.cornerSubPix()`

Useful commands

`cv2.filter2d` - Filtering with linear filter kernel

`cv2.getGaussianKernel` - Return the $ksize \times 1$ matrix of Gaussian filter coefficients:

`np.gradient` - Return a set of gradients of an N-dimensional array corresponding to the derivatives with respect to each dimension

`np.linalg.norm` - Return one of eight different matrix norms, or one of an infinite number of vector norms

`np.reshape` - Return an array containing the same data with a new shape.

`np.resize` - Change shape and size of array in-place.

`np.r_` - Concatenate along first axis described here

`np.c_` - Concatenate along second axis described here

`np.argsort` - Return the indices that would sort this array

`np.arctan2` - Element-wise arc tangent of x1/x2 choosing the quadrant correctly.

`np.digitize` - Return the indices of the bins to which each value in input array belongs.

Part 2: Documentation (6 points)

Perform the following experiments with the given images and **at least one picture set you took yourself**, and answer the questions in a few sentences.

- Answer each numbered question separately, with the answer labelled according to the experiment and question number. For example, experiment 1 question 2 should be numbered 1.2. You do not need to include the question text.
- Include images to support your answers. These should be referenced in the answers. The document should include information about what parameter settings were used to generate images so that they can be reproduced..
- Your answers should attempt an explanation of why an effect occurs, not just describe what you see in the supporting images.
- Be precise. "It is better" or similarly vague statements without any additional information are not sufficient.

Points may be deducted if your report does not satisfy the above points.

Document the results using representative example images. Only change the parameters in the main_... files at the marked places.

Maximum 1500 words, arbitrary number of pages for result images (referenced in the text). Violating the text limit will lead to point deductions!

Experiment 1 (2.5 points)

Calculate the Harris Corners with different sigma values for the Gaussian filtering and test their influence on the results. Call the script `main_harris_image.py`.

1. How do different sigma values for the initial Gaussian filtering influence the output of the detector?
2. How does the second Gaussian filtering improve the output of the detector?
3. The Harris detector is not scale invariant. Give an example of an image patch which would fire the detector at one scale, but not at another.
4. How does not being scale invariant limit the effectiveness of a detector? Use a simple example to explain.
5. What is the effect of varying the k parameter?

Experiment 2 (3.5 points)

Match the patch descriptors of several similar images using different parameter settings. **Use at least one picture set you took yourself.** You should look at the effect of rotated images, and images with different illumination (you can use lightened images with suffix `cl` in the

`desk` directory), or modify your own images using the curves or levels tools in Photoshop, GIMP or any other image editor. Also try with non-sequential images, e.g. use desk00 and desk03.

1. How does changing the patch size of the affect the accuracy of the matching with the patch descriptors?
2. How and why are the descriptors affected by rotation of the image? Approximately at which point do the patch descriptors no longer produce good results? Use the `rotate_bound` function from `helper_functions.py` to make sure the detector fires approximately in the same locations.
3. What problem does normalising the patch descriptor mitigate? Is it still as effective as the basic descriptor?
4. What problem does sorting the patch mitigate? Are the results better with the circular version? Why is this?
5. How does the matching quality of the block descriptor compare to the others?
6. What are two possible benefits of using a histogram to describe parts of the patch?
7. What are two possible benefits of using blocks within the patch? Are there any disadvantages?

Assistance

Please keep to the following hierarchy whenever there are questions or problems:

1. Forum: Use the TUWEL forum to discuss your problems.
2. Email for questions: machinevision@acin.tuwien.ac.at

Upload

Please upload all Python files and your documentation (pdf-format) **as one ZIP-file** via TUWEL.