Computer Vision

Lab Assignment - Local Features

ETH Zurich, Computer Science Department

Objective

In this lab assignment, you will implement your own feature detector and a basic matching protocol to establish pixel-wise correspondences between images. Along with the template source code, some test images are provided.
The main structure of the program is provided. The procedures will be run from `main.py`.
Do not modify the function interfaces. Do not change the already implemented functions at `functions/vis_utils.py` and `functions/extract_descriptors.py` for visualization and descriptor extraction.
You will need to have the following Python packages required for this Lab Assignment: `numpy`, `opencv-python`, and `scipy`.

1 Detection (Total: 60%)

In this section, you will implement a Harris corner detector to find interest points in an image. For this, you will start by computing the Harris response function $\mathcal{C}$ for all pixels in an image. Afterwards, a pixel (i, j) is selected as a keypoint in an image if the following two conditions are satisfied: $\mathcal{C}(i, j)$ is above a certain detection threshold T and $\mathcal{C}(i, j)$ is a local maxima in its 3×3 neighborhood. For more details, please refer to [1]. Add your code to `functions/extract_harris.py`.

1.1 Image gradients (10%)

Compute the image gradients I_x and I_y in the x and y directions respectively. For this question, you may use the `scipy.signal.convolve2d` function with the right convolutional filters.

$$I_x(i,j) = \frac{I(i,j+1) - I(i,j-1)}{2}, I_y(i,j) = \frac{I(i+1,j) - I(i-1,j)}{2} \tag{1}$$

1.2 Local auto-correlation matrix (20%)

Compute the elements of the local auto-correlation matrix defined by the following equation at each pixel position p:

$$M_p = \sum_{p' \in \mathcal{N}(p)} w_{p'} \begin{bmatrix} I_x(p')^2 & I_x(p')I_y(p') \\ I_y(p')I_x(p') & I_y(p')^2 \end{bmatrix} \ , \tag{2}$$

where $\mathcal{N}(p)$ is a neighbourhood of point p. The local weighting w is generally chosen to be Gaussian with standard deviation σ (centered in $(0, 0)$). You may use the `cv2.GaussianBlur` function.

1.3 Harris response function (10%)

The Harris response function can be defined as $\mathcal{C}(i, j) = \det(M_{i,j}) - k \operatorname{Tr}^2(M_{i,j})$ with $k \in [0.04, 0.06]$, an empirically determined constant. Compute the Harris response function for all pixels using the closed-form formulas for the determinant and trace.

1.4 Detection criteria (20%)

Take into account the two detection conditions (strength and local maximality of the response) mentioned at the beginning of section 1 to obtain the final keypoints. For the local maximum check, you may take advantage of the `scipy.ndimage.maximum_filter` function.

In the report, plot the detected keypoints using `plot_image_with_keypoints` in the `main.py` for the test images (`blocks.jpg` and `house.jpg`) when varying the threshold T, the constant k and the standard deviation σ. Choose the most appropriate values to use for the rest of the assignment. What possible issues do you notice with the detected keypoints?

2 Description & Matching (Total: 40%)

In this section, you will implement a matching protocol for image patches and test it out on the provided image pair (`I1.jpg`, `I2.jpg`). Add your code to `functions/extract_descriptors.py` for Question 2.1 and `functions/match_descriptors.py` for Questions 2.2 and 2.3 respectively.

2.1 Local descriptors (10%)

Use the provided `extract_descriptors` function to extract 9×9 patches around the detected keypoints which will be used as descriptor. Make sure to first filter out the keypoints that are too close to the image boundaries so as to avoid out-of-bounds issues.

2.2 SSD one-way nearest neighbors matching (20%)

The sum-of-squared-differences (SSD) is defined as:

$$SSD(p, q) = \sum_i (p_i - q_i)^2 \quad . \tag{3}$$

Compute the SSD between the descriptors of all features from the first image and the second image. Avoid the for-loop by using vectorized computation in Python.
Implement a one-way nearest neighbors matching protocol where each feature from the first image is associated to its closest feature from the second image.
In the report, plot the obtained matches using the provided `plot_image_pair_with_matches` function and comment on the results.

2.3 Mutual nearest neighbors / Ratio test (5% + 5%)

Implement the mutual nearest neighbors matching protocol where for each one-way match, you check that it is also valid when swapping the images. Comment on the results.
Implement the ratio test matching protocol where a one-way match is considered valid if the ratio between the first and the second nearest neighbor is lower than a given threshold (for instance, you can use 0.5). You may take advantage of the `np.partition` function. Comment on the results.

Hand in

Hand in your commented Python code and write a short report explaining the main steps of your implementation and addressing the questions from above. Make sure to include the essential parts of your code in the report!
Send the report together with your source code via the moodle submission system (do not send it over email).

References

[1] Chris Harris and Mike Stephens. "VLFeat open source library". In: *Proceedings of the Alvey Vision Conference*. 1988.