Coursework 2: Local Feature Detection and matching for object recognition

The goal of **Local Feature** detection and matching is to identify pairs of matching (corresponding) points between images. You can then use these features for a task such as finding and recognising similar objects in different images.

In this coursework, you will write code to detect local features in an image and find the best matching features in another image.

For this: Use the image *bernieSanders.jpg* as your reference image and find the best matching features in all the other images provided (on BB).

To help you visualise the results you may use any of the of python snippets code provided on BB or any OpenCV functions you consider appropriate. In the following link you will find an example of using the ORB feature detector, a popular technique in the vision community, for comparison:

https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html

Coding Rules

You may use NumPy, SciPy and OpenCV functions to implement mathematical, filtering and transformation operations. ***Do not*** use functions which implement **interest point** detection or **feature matching.**

When using the Sobel operator or gaussian filter, you should use the '*reflect*' mode, which gives a zero gradient at the edges.

Please refer to the 'Implementation and report submission details' section at the end of this document for further details.

Description

The coursework has three parts: Feature detection, feature description, and feature matching.

1. Feature detection

In this step, you will identify ***interest points*** in the image using the **Harris corner** detection method. *You will have to write this function yourself and the steps of the method can be found in the lecture slides*. Let's call this function ***HarrisPointsDetector***.

For each point in the image, consider a window of pixels around that point. Compute the Harris matrix M for (the window around) that point, defined as

$$M = \sum_p w_p \begin{pmatrix} I_{x_p}^2 & I_{x_p} I_{y_p} \\ I_{x_p} I_{y_p} & I_{y_p}^2 \end{pmatrix}$$

where the summation is over all pixels p in the window. I_{x_p} and I_{y_p} are the x and y derivatives of the image at point p. You should use the *3x3* **Sobel operator** to compute the x, y derivatives (for pixels outside the image, pad the image using *reflection*). The weights w_p should be circularly symmetric (for rotation invariance) - use a 5x5

Gaussian mask with 0.5 *sigma*. Use reflection image padding for handling gradient values outside of the image range. Note that M is a 2x2 matrix.

Then use M to compute the corner strength function, R, at every pixel.

$$R = c(M) = det(M) - 0.05\big(trace(M)\big)^2$$

We will also need the orientation in degrees at every pixel. Compute by finding the gradient direction in each pixel. The zero angle points to the right and positive angles are counter-clockwise.

Select the strongest **interest points (keypoints)** (according to R) which are local maxima in a 7x7 neighbourhood.

You have to write the code *for this function* yourselves but you may use OpenCV functions for the Sobel operator, for getting the Gaussian mask for padding the image etc. etc.

2. Feature description

Now that you have identified the ***interest points***, the next step is to create a ***descriptor*** for the features centred at each interest point detected above in part 1. This descriptor will be the representation you use to compare features in different images to see if they match. You will be using the ORB descriptor ***compute*** function (see https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html) to create a descriptor.

Also, you will compare the performance achieved by the ORB ***compute*** function that makes use of the features detected by your own ***HarrisPointDetector*** function, with the performance achieved by calling the ORB ***detect*** functions, with its own built-in settings:

- Harris;
- FAST

3. Feature matching

Now that you have detected and described your features, the next step is to write code to match them (i.e., given a feature in one image, find the best matching feature in another image).

The simplest approach is the following: compare two features and calculate a scalar *distance* between them. The best match is the feature with the smallest distance. You will write yourself two functions:

1. Sum of squared differences (SSD): This is the squared Euclidean distance between the two feature vectors.
2. The ratio test: Find the closest and second closest features by SSD distance. The ratio test distance is their ratio (i.e., SSD distance of the closest feature match divided by SSD distance of the second closest feature match).

You have to write the code for this function yourselves but you may use other python or OpenCV functions to help you do that.

You may find useful using *cv2.drawMatches()* function for visualising your results.

Visualising and Reporting your results

- ***Interest point (Keypoint) Detection***
  Compute the points of interest with their orientation in all images provided.
  How did you choose the threshold value to find the strongest interest points?
  Plot the number of detected feature points as you vary the threshold value.

- ***Feature Matching***
  Here you load two images and view the computed best matches.
  Compare all images in the benchmark set with the *bernieSanders.jpg* image.
  How well does it work? What are your observations? How (and what) parameter choices affect your results?

We are providing you with a set of benchmark images to be used to test the performance of your algorithm as a function of different types of variation (i.e. rotation, scale, illumination, blurring).

Implementation and Report submission Details

1. Implement the Harris interest point Detector function (let's call it ***HarrisPointsDetector***). Your implementation should include the computation of local maxima function. Your ***HarrisPointsDetector*** function should return an array of locations (with orientations) that can be read by the ORB ***compute***

function in OpenCV. You may find the following functions helpful:

- *scipy.ndimage.sobel*: Filters the input image with Sobel filter.
- *scipy.ndimage.gaussian_filter*: Filters the input image with a Gaussian filter.
- *scipy.ndimage.filters.maximum_filter*: Filters the input image with a maximum filter filter.
- *scipy.ndimage.filters.convolve*: Filters the input image with the selected filter.

2. You will need to implement your feature descriptor function (let's call it ***featureDescriptor)*** using the ORB ***compute*** function from OpenCV. This function should take the location and orientation information already stored in a set of key points (Harris corners) from the previous step 1 of your implementation, and compute descriptors for these key points, then store these descriptors in a *numpy array* which has the same number of rows as the computed key points and columns as the dimension of the feature.

3. Finally, you will implement a function for matching features. You will implement the **matchFeatures** function of **SSDFeatureMatcher** and **RatioFeatureMatcher**. These functions return a list of *cv2.DMatch* objects. You should set the *queryIdx* attribute to the index of the feature in the first image, the *trainIdx* attribute to the index of the feature in the second image and the *distance* attribute to the distance between the two features as defined by the particular distance metric (e.g., SSD or ratio). You may find *scipy.spatial.distance.cdist* and *numpy.argmin* helpful when implementing these functions and also *cv2.drawMatches()* for visualising your results.

4. Submit a ***report*** which includes benchmarking results (as described above), your choice of parameters, quantitative results, and your observations. Also, according to your experimental results and observations, which local features do a better job at finding Bernie (well ok, matching Bernie)? ORB local features using Harris interest points detector or ORB local features using FAST interest point detector?