Report 3

Image-matching

Team members:

Name	Sec	BN
Esraa Ali	1	12
Rawan Abdelhamed	1	33
Yehia said	2	52
Omar Mostafa	2	5
Mostafa Mahmoud	2	37

Methods

This project was mainly implemented using the following python packages:

1. Numpy : for computations
2. Matplotlib : for visualization and plotting
3. Streamlit : for build our app
4. CV2 : For reading and handling uploaded image
5. **OS :** Python OS module provides the facility to establish theinteraction between the user and the operating system
6. PIL : for dealing with images

Our project can be divided into 4 main points as follows:

- **Extract the unique features in all images using Harris operator and λ**:

The code is consisting of helping functions and the main function (Harris function), the idea of implementation is:

First: calculate the derivatives of the image (imx,imy).

Second: calculate the partial derivative for each derivative.

Third: calculating the lambda by using the determinant and the trace of the matrix.

Fourth: getting the corners using a threshold.

The time of executing this function is mainly dependent on the size of the image, it increases as the size increases.

- **Generate feature descriptors using scale invariant features(SIFT):**

First "The functions of pysift.py":

1. Generate Base Image:
   Generate base image from input image by upsampling by 2 in both directions and blurring.

   Return gaussian blurred image.
2. Compute Number Of Octaves:
   Compute number of octaves in image pyramid as function of base image shape (OpenCV default).
3. Generate Gaussian Images:
   Generate scale-space pyramid of Gaussian images.

   Return an array of gaussian images.
4. Generate DoG Images:
   Generate Difference-of-Gaussians image pyramid.
5. Find Scale Space Extrema:
   Find pixel positions of all scale-space extrema in the image pyramid.

   Return key points.
6. Localize Extremum Via Quadratic Fit:
   Iteratively refine pixel positions of scale-space extrema via quadratic fit around each extremum's neighbors. Make sure the new pixel_cube will lie entirely within the image.
7. Compute Gradient At Center Pixel:
   Approximate gradient at center pixel [1, 1, 1] of 3x3x3 array using central difference formula of order O(h^2), where h is the step size.
8. Compute Hessian At Center Pixel:
   Approximate Hessian at center pixel [1, 1, 1] of 3x3x3 array using central difference formula of order O(h^2), where h is the step size.
9. Compute Key points With Orientations:
   Compute orientations for each key point.

   Return key points with orientations.
10. Compare Key points:
    Return True if keypoint1 is less than keypoint2.
11. Remove Duplicate Key points:
    Sort key points and remove duplicate key points.

    Return key points and unique key points.

12. Convert Key points To Input Image Size:
    Convert key point, size, and octave to input image size.

    Return converted key points.

13. Unpack Octave:
    Compute octave, layer, and scale from a key point.

    Return octave, layer, scale.

14. Generate Descriptors:
    Generate descriptors for each key point.

    Return array descriptors.

15. Compute Key points And Descriptors:
    Compute SIFT key points and descriptors for an input image using all the functions we have created.

    Return key points and descriptors.

Second "Function of run_pysift":

16. pysift_image:
    Used to test all the function and compute SIFT key points and descriptors, initialize and use FLANN, Lowe's ratio test, estimate homography between template and scene, draw detected template in scene image, and draw SIFT key point matches.

    Return the SIFT applied image.

    Computational time of the SIFT:

    The computation time depends mainly on two factors:

    •The image dimensions width.

    •The second factor is the number of key points.

    •Complexity O(MN+I) as MxN for the size of the image, and N for the number points.

- **Match the image set features using Normalized Cross Correlation:**

First read both the base image and the target using *cv2.imread* ,then get for both the width and height using *.shape*

Then, Create a matrix for the base image and target and a matrix for NCC value with dimension of (height of base image – height of target image) and the same for width

Then , calculate value using filter-kind operation from top-left to bottom-right by calculate the NCC values and find the most match area.

the execution time of this code will depend primarily on the size of the input images, and it can be estimated as

O((height of base image – height of target image)*(width of base image – width of target image)

- **Match the image set features using sum of squared differences (SSD):**

The standard sum of squared differences (SSD) similarity metric is used and the position of the target is found as that giving the lowest dissimilarity score.

SSD	$s = \sum_{(u,v)\in\mathbf{I}} (\mathbf{I}_1[u,v] - \mathbf{I}_2[u,v])^2$	ssd()

Given a source image I and a template T, we compare the template image against the source image by sliding it one pixel at a time (left to right, top to bottom) and computing a similarity (or alternately difference) between template and the image patch at each location. The similarity scores is computed using a suitable function $\mathrm{g(T,I,i,j)\mapsto R(i,j)}$, where $\mathrm{R(i,j)}$ is the similarity score for between the template and the image patch at location $\mathrm{(i,j)}$. Location corresponding to highest (or alternately lowest) value in the result matrix R represent the "match location."

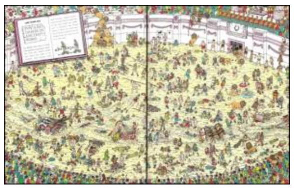
A common trick is to treat both the template and patches as vectors in a high-dimensional space. Template-patch-matching problem is then reduced to finding the nearest vector (in this high-dimensional space).

- **Picking image patches**
- **Techniques for comparing patches**

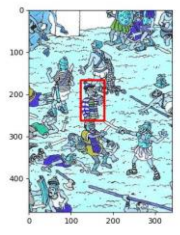
We now discuss methods for comparing image patches (and templates). The following discussion assumes that patches (and templates) have the same dimensions (*observation 1 above*). We also use the following notation:

  - Template: T
  - Image: I
  - Image patch centered at $\mathrm{(i,j)}$: $\mathrm{I(i+k,j+l)}$
  - Response: R

- **Sum of Squared Differences (SSD)**



$\longrightarrow$


0
100
200
300
400
0
100
200
300