EECS 504 Foundations of Computer Vision: PS2

Term: Winter 2026
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 2/24 23:59 Eastern Time

Starter code: The starter code can be found HERE. We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Submission Process:

1. To Gradescope: a `pdf` file as your write-up, including your answers to all the questions as well as the result images from the code. *Please do not handwrite.* The write-up must be electronic. You can use Word, Google Docs, LaTeX(try overleaf!), or any similar application.
2. To Canvas: The .ipynb file containing all the visualizations should be submitted to Canvas. Before submitting, please make sure to rename the file to EECS504_PS2_<your_uniquename>_<your_umid>.ipynb

Grading and Evaluation: The credit for each problem in this set is given in parentheses at the stated question (sub-question fraction of points is also given at the sub-questions). Partial credit will be given for both paper and python questions.

Constraints: This assignment may be discussed with other students in the course but must be written independently. Programming assignments should be written in python. Over-the-shoulder coding is strictly prohibited. **Web/Google-searching for background material is permitted. However, everything you need to solve these problems is presented in the course notes and background materials, which have been provided already.**

Goals: Deepen understanding of image operations, invariance, local feature point case study, and learning image bases.

Problem 1 (20): Affine Photometric Invariance of Harris Operator

We discussed how one practical goal toward photometric invariance is to seek an operator that is invariant to an affine intensity transformation (spatial range map of size one pixel):

$$\mathbf{I}'(x,y) = a\mathbf{I}(x,y) + b \quad \forall \text{ pixels } (x,y) \in \Lambda \tag{1}$$

where $a, b \in \mathbb{R}$.

For the corner detector operator we discussed in class, $\min(\text{eigs}(S(\mathbf{I}(W))))$, where $S(\cdot)$ is the structure tensor operating on some window W into image $\mathbf{I}$:

$$S(\mathbf{I}(W)) \doteq \sum_{x,y \in W} \begin{bmatrix} \mathbf{I}_x(x,y)^2 & \mathbf{I}_x(x,y)\mathbf{I}_y(x,y) \\ \mathbf{I}_x(x,y)\mathbf{I}_y(x,y) & \mathbf{I}_y(x,y)^2 \end{bmatrix} \tag{2}$$

Derive and discuss how the $\min(\text{eigs}(S(\mathbf{I}(W))))$ corner detector operator response on $\mathbf{I}$ is related to that of $\mathbf{I}'$. (Derive the response of $\mathbf{I}'$ as a function of the response of $\mathbf{I}$ and (a,b).) Is it invariant to the affine transformation? Discuss.

Problem 2 (30): Local Image Features and Image Stitching

We motivated and derived an initial local image feature point detector based on an eigen decomposition of the structure tensor. We also demonstrated an application example of image stitching. In this question, you will explore these topics further.

1. (10) Implement the local structure tensor construction and eigendecomposition as described in class. This method is called Harris, but implement the simplified corner response measure based on the minimum eigenvalue of the structure tensor, as discussed in class. Implement the function `harris` which should return a matrix where every pixel contains the corner strength. This is a "corner response image". Include the result image(s) in your report.
2. (5) We have implemented the function `corner_detections` which uses Non-Maximal suppression and thresholding to find corners from your corner response images. We wrote some code that runs this function on the `rings.jpg` image. Describe why there are so many responses yet there are no "corners" in the image, and why, even though, there are many responses, no full ring has a high corner response over the entire ring.
3. (5) Figure out a way that will improve this result to get corners only on the right boundaries of the `rings.jpg` image at most. You can modify your rings image and the corresponding harris response in any way you want. You CANNOT modify our `corner_detections` function in any way. Instead, focus on removing the false positives so that you only have detected corners on the ring boundaries. Include the result image(s) in your report.

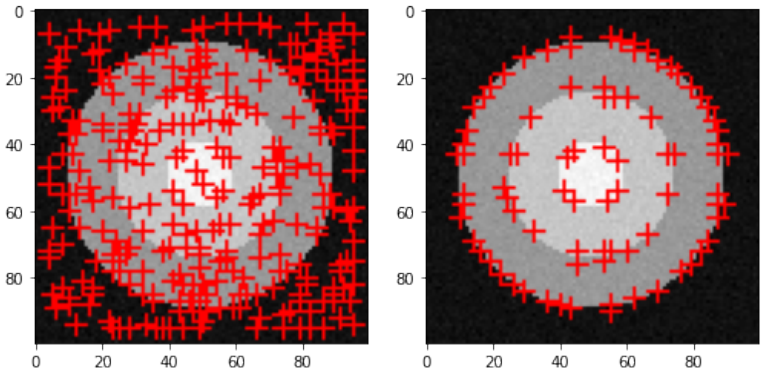

Figure 1: (Left) Corner detections before improvements. (Right) Corner detections after improvements

Figure 2: (Left) Left side of red house. (Right) Right side of red house

4. (10) **Image Stitching**. This question will show you an end-to-end use of these computer vision tools by automatically stitching the two images of a red house. You don't need to implement anything for this part. To make this possible a lot of existing code has been provided for you

   The code for doing this has the three basic pieces: reduction (extraction of features with ORB features), matching (finding the best `K` correspondences across the images), and estimation (computation of the homography that aligns the two image). You can walk through the code to get a better sense of these steps.

   We will do the stitching with different number of correspondences (4, 20 and 30). To fit the homography, at least 4 correspondences are needed, which is why we use 4. But, 4 seems to yield an inaccurate stitching. 20 is much better. However, when we go to 30, it fails completely.

   Explain (1) why 4 correspondences is worse than 20; (2) why 30 correspondences, which we may expect to be better, completely fails.

Problem 3 (25): DoG Blob Detection and Scale Selection

We discussed scale-space ideas and the relationship between the DoG and blob detection.

The three images that are included are a synthetic image of a circle, a field of sunflowers, and a *brainbow* image of neurons in a Drosophila brain. These are depicted below, left-to-right. The fly-brain is very interesting and a relevant problem for which detecting neuron nuclei is an important problem. This image has been provided by Dr. Dawen Cai in the Department of Cell and Developmental Biology `http://www.cai-lab.org`.

1. (10) Step 1 will be to implement the function `DoGScaleSpace` that builds the Difference of Gaussian scale space representation of an image. Describe the scale space and anything you find interesting about it (less than five sentences). Include the result image(s) in your report.
2. (5) Step 2 will be to implement the function `find_local_extrema`. This function looks through $3 \times 3 \times 3$ windows in space and scale to find local extrema (when the center point is larger or smaller than every other point in the window). We

Figure 3: (Left) Synthetic circle, (Center) Sunflowers, (Right) Drosophila brainbow

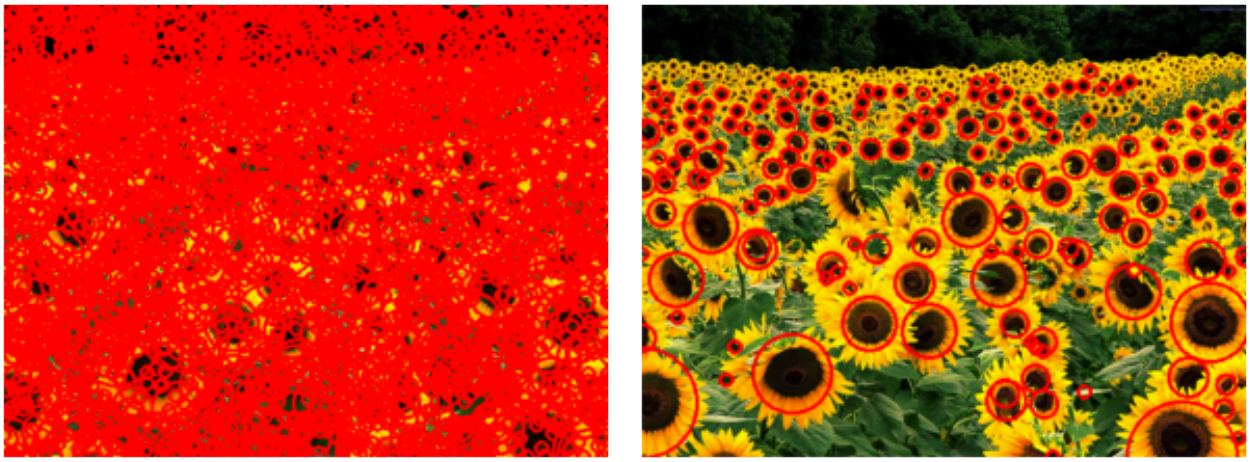

Figure 4: (Left) Blob detections before improvements. (Right) Blob detections after improvements

provide code to visualize the extrema for the scale spaces of the three images. There should be a lot of false blob detections. Answer briefly, why are so many extrema detected? Include the result image(s) in your report.

3. (5) Step 3 is to filter these detected local extrema to reduce the detected set down to a more usable size. You will have to implement the function `filterBlobs`. You will have to filter out blobs that have a DoG response magnitude smaller than a certain threshold (DoGtau), whose value has been provided for fairness. Optionally, you can improve the function by filtering out regions that do not resemble blob regions. If you carefully inspect the extrema points from the question above, you will find that extrema in the 2D DoG scale space are present in many places other than blobs. Include the result image(s) in your report.
4. (5) Consider the blob visualization on the drosophila image. Include the result blob image here. Discuss the detections you find; the false positives; the missing regions. Consider the structure of the image. Describe an algorithm that could not only find the blobs (cell nuclei) but also the long stringy-things connected to them (cell processes, like axons). Can the scale-selection ideas be generalized to lines?

Problem 4 (25): MNIST number KNN classification

Recall the eigenface example discussed in the class. Now apply the same technique to classify numbers of the MNIST dataset. MNIST is a large database of handwritten digits. Every image contains a digit and has a size of 28×28. You can learn more about how K nearest neighborgh classification (KNN) and how to implement it using online resources. We recommend this article: KNN towards data science

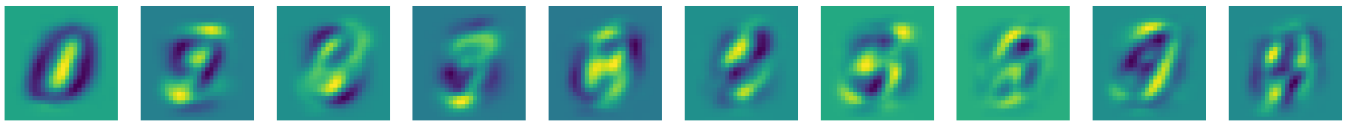

Figure 5: Visualization of top 10 eigenvectors for the covariance matrix of the training set

1. (5) You will have to implement the function `zero_mean` which returns the zero-mean version of the training and test set. Remember to normalize the test set with the mean from the training set.
2. (5) Next, you will implement the function `compute_basis` which will return the new basis for you train and test sets. Some of the vectors of this basis can be seen in Figure 5.
3. (5) Visualize your basis images. Do they match the ones in the assignment pdf? Provide three observations about the basis set that you find interesting.
4. (5) You will also implement the function `change_basis` which will apply the change of basis to your train and test sets.
5. (5) Finally, you are required to implement a K nearest neighbor classifier to recognize the numbers. Fill in the function `knn_predict` to do this. You should get a test accuracy greater than 80%.