Recitation 02

AUTHOR
Christian Osendorfer

Written Questions

- How does finding corresponding points between image pairs contribute to 3D depth inference?
- Differentiate between sparse correspondence and dense correspondence in image matching.
- What image characteristics define a "corner" as an easy-to-match feature?
- Describe the characteristics of an image patch that make it "easy to match" using the concept of the Sum of Squared Differences (SSD) error.
- In the mathematical formulation of cornerness, what does the expression E(u,v) represent, and what role do u and v play?
- How do eigenvalues and eigenvectors relate to the identification of corners in an image?
- Explain how the determinant and trace of matrix A can be used to efficiently compute cornerness without directly calculating eigenvalues.
- Why is scale invariance a challenge in corner detection, and what strategies can be employed to address this issue?
- What is the purpose of coarse-to-fine search in scale selection for interest point detection?
- Briefly outline the overall pipeline for finding correspondences between two images using interest points.
- Why are feature descriptors important in computer vision?
- What is the main limitation of using raw pixel values as feature descriptors?
- How do image gradients help in creating more robust feature descriptors?
- Explain the concept of a binary descriptor and its advantages.
- What is the primary advantage of using a color histogram as a feature descriptor?
- What is the key difference between a color histogram and a spatial histogram?
- How does orientation normalization contribute to achieving rotation invariance?
- What is the significance of designing feature descriptors that are invariant to photometric transformations?
- Why is it challenging to design feature descriptors that are robust to geometric transformations?
- Describe a scenario where a combination of different feature descriptor types might be beneficial.

Implement in Python

You'll find some code snippets in the accompanying file `lab01.zip`. The code and the following text are taken from Frank Dellaert's Computer Vision course at Georgia Tech. Before starting to write code, take a brief look at the files in the zipped archive and set your local python environment up accordingly.

This project is intended to familiarize you with Python, PyTorch, and image filtering. The goal of this assignment is to write an image filtering function and use it to create hybrid images using a simplified version of the SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns. Hybrid images are static images that change in interpretation as a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances.

The archive also contains a set of 5 pairs of aligned images which can be merged reasonably well into hybrid images.

Potentially useful Python functions:

- `np.pad()`: Does many kinds of image padding.
- `np.clip()`: "Clips" out any values in an array outside of a specified range
- `np.sum()` and `np.multiply()`: Make it efficient to do the convolution (dot product) between the filter and windows of the image.

Numpy

Gaussian Kernels. Gaussian filters are used for blurring images. You will be implementing `create_Gaussian_kernel()` that creates a 2D Gaussian kernel according to a free parameter, `cutoff frequency`, which controls how much low frequency to leave in the image. This is an important step for later in the project when you create hybrid images!

The multivariate Gaussian function is defined as:

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\det(\Sigma)^{1/2}}$$

where n is equal to the dimension of x, μ is the mean, and Σ is the covariance matrix. Alternatively, you can create a 2D Gaussian by taking the outer product of two vectors. Each such vector should have values populated from evaluating the 1D Gaussian PDF at each coordinate.

The 1D Gaussian is defined as:

$$p(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

In your implementation, you will use the value of the cutoff frequency to define the size, mean, and variance of the Gaussian kernel. Specifically, the kernel G should be size (k,k) where $k = 1 + 4\times$ `cutoff_frequency`, have mean $\mu = \lfloor \frac{k}{2} \rfloor$, standard deviation $\sigma =$ `cutoff_frequency`, and values that sum to 1 (i.e. $\sum_{ij} \alpha G_{ij} = 1$ where α is some constant scale factor).

Image Filtering. Image filtering (or convolution) is a fundamental image processing tool. You will be writing your own function to implement image filtering from scratch. More specifically, you will implement `my_imfilter()` which imitates the `filter2D()` function in the OpenCV library. As specified in `part1.py`, your filtering algorithm must: (1) support grayscale and color images, (2) support arbitrarily-shaped filters, as long as both dimensions are odd (e.g. 7x9 filters, but not 4x5 filters), (3) pad the input image with zeros or reflected image content, and (4) return a filtered image which is the same resolution as the input image. There is an iPython notebook, `proj1_test_filtering.ipynb`, along with some tests (which are called in `proj1.ipynb`) to help you debug your image filtering algorithm. Note that there is a time limit of 5 minutes for a single call to `my_imfilter()`, so try to optimize your implementation if it goes over.

Hybrid Images. A hybrid image is the sum of a low-pass filtered version of one image and a high-pass filtered version of another image. As mentioned in the previous paragraphs, a cutoff frequency controls how much high frequency to leave in one image and how much low frequency to leave in the other image. In `cutoff_frequencies.txt`, we provide a default value of 7 for each pair of images (the value on line `i` corresponds to the cutoff frequency value for the i-th image pair). You should replace these values with the ones you find work best for each image pair. In the paper it is suggested to use two cutoff frequencies (one tuned for each image) and you are free to try that as well. In the starter code, the cutoff frequency is

controlled by changing the standard deviation of the Gaussian filter used in constructing the hybrid images. You will first implement `create_hybrid_image()` according to the starter code in `part1.py`. Your function will call `my_imfilter()` using the kernel generated from `create_Gaussian_kernel()` to create low and high frequency images and then combine them into a hybrid image.

PyTorch

You will implement creating hybrid images again, but this time using PyTorch (so have your environment ready!).

Dataloader. The `HybridImageDataset` class in `datasets.py` will create tuples using pairs of images with a corresponding cutoff frequency (which you should have found from experimenting in the previous paragraphs). The images will be loaded from the folder `data/` and the cutoff frequencies from `cutoff_frequencies.txt`. Refer to this tutorial for additional information on data loading & processing.

Model. Next, you will implement the `HybridImageModel` class in `models.py`. Instead of using your implementation of `my_imfilter()` to get the low and high frequencies from a pair of images, `low_pass()` should use the 2d convolution operator from `torch.nn.functional` to apply a low pass filter to a given image. You will have to implement `get_kernel()` which calls your `create_Gaussian_kernel()` function from `part1.py` for each pair of images using the `cutoff_frequencies` as specified in `cutoff_frequencies.txt` and reshapes it to the appropriate dimensions for PyTorch. Then, similar to `create_hybrid_image()` from Part 1, `forward()` will call `get_kernel()` and `low_pass()` to create the low and high frequency images and combine them into a hybrid image. You can refer to this tutorial for additional information on defining neural networks using PyTorch, even when you don't know yet what a neural network is!

Lastly, you will compare the runtimes of your hybrid image implementations from Parts 1 and 2.