Assignment 2 Questions

Instructions

- One exercise graded for completion only (please include your results, e.g., two-to-three screenshots of your findings).
- Four graded questions.
- One ungraded 'something to think about'.
- Write code where appropriate.
- Feel free to include images or equations.
- submit as a pdf

1 Exercise

E1: Take a look at the webcam Fourier decomposition demo code include. Run it with a webcam.

```
$ python liveFFT2.py
```

This file contains five parts for you to explore and see the amplitude image, the phase image, and the effect of the reconstructed image.

- Part 0: Scanning the basis and observing the output image.
- Part 1: Reconstructions from different numbers of basis frequencies.
- Part 2: Replacing amplitude and phase with that from a different image.
- Part 3: Replacing amplitude and phase with that from a noise image.
- Part 4: Manipulating the amplitude and phase images.

Uncomment the different parts and explore the camera feed decomposition! Please include the results of your experimentation, e.g., two-to-three screenshots of what you discover. We'll be grading for completion, not correctness. *Note:* For anonymous grading, try not to put yourself in the camera frame. Show your favourite vector calculus book, wear a mask, use your cat, etc. Extra credit for creative effort.

Questions

Q1: Imagine we wished to find points in one image which matched to the same world point in another image—so-called feature point correspondence matching. We are tasked with designing an image feature point algorithm which could match world points in the following three pairs of images.

Please use the included python script `plot_corners.py` to find corners using Harris corner detection. Discuss the differences in the returned corners (if any) for each image pair and what real world phenomena or camera effects may have caused these differences. Then discuss which real world phenomena and camera effects might cause us problems when matching these features. Please provide at least one problem per pair.

RISHLibrary: One Two — *Chase:* One Two — *LaddObservatory:* One Two

A1: Your answer here.

Q2: In the Harris corner detector, what do the eigenvalues of the 'M' second moment matrix represent? Discuss both how they relate to image intensity and how we can interpret them geometrically.

A2: Your answer here.

Q3: Given a feature point location, the SIFT algorithm converts a 16×16 patch around the feature point into a 128×1 descriptor of the gradient magnitudes and orientations therein. Write pseudocode *with matrix/array indices* for these steps.

Notes: Do this for just one feature point at one scale; ignore the overall feature point orientation; ignore the Gaussian weighting; ignore all normalization post-processing; ignore image boundaries; ignore sub-pixel interpolation and just pick an arbitrary center within the 16×16 for your descriptor. Please just explain in pseudocode how to go from the 16×16 patch to the 128×1 vector. You are free to simplify the gradient computation.

A3: Your answer here.

Q3: Given a feature point location, the SIFT algorithm converts a 16×16 patch around the feature point into a 128×1 descriptor of the gradient magnitudes and orientations therein. Write pseudocode *with matrix/array indices* for these steps.

Notes: Do this for just one feature point at one scale; ignore the overall feature point orientation; ignore the Gaussian weighting; ignore all normalization post-processing; ignore image boundaries; ignore sub-pixel interpolation and just pick an arbitrary center within the 16×16 for your descriptor. Please just explain in pseudocode how to go from the 16×16 patch to the 128×1 vector.

A3: Your answer here.

```
# You can assume access to the image, x and y gradients, and their
                                        magnitudes/orientations.
image = imread('rara.jpg')
grad_x = filter(image, 'sobelX')
grad_y = filter(image, 'sobelY')
grad_mag = sqrt( grad_x .^2 + grad_y.^2 )
grad_ori = atan2( grad_y, grad_x )

# Takes in a feature point x,y location and returns a descriptor
def SIFTdescriptor(x, y)
    descriptor = zeros(128,1)


    return descriptor
```

Q4: Explain the difference between the Euclidean distance and the cosine similarity metrics between descriptors. What might their geometric interpretations reveal about when each should be used? Given a distance metric, what is a good method for feature descriptor matching and why?

A4: Your answer here.

Something to think about: In designing a feature point matching algorithm, what characteristics might we wish it to have? How might two world points change in appearance across photographs? Consider that we might allow brightness or contrast changes, or texture changes, or lighting changes, or geometric changes in appearance like rotation and translation in three dimensions or camera perspective effects. All exist between some two photographs of real-world points.

We are faced with a fundamental trade-off between feature point invariance (how much variation in appearance I allow and still say that two points are the same) and discriminative power (our ability to say that two points are different or the same at all).

How should we design for this trade-off?