Comp 558
A2

Yiran Wang

Q1. To see the result, please run `make Gaussian Pyramid.m`

1024 × 1024
σ = 1

32 × 32
σ = 32

16 × 16
σ = 64

512 × 512
σ = 2

256 × 256
σ = 4

128 × 128
σ = 8

64 × 64
σ = 16

Q2 : To see the result, please run makeLaplacianPyramid.m

1024 x 1024

512 x 512

256 x 256

128 x 128

64 x 64

32 x 32

* For Q3, 4, 5, 7 please run OriginalImage Processing.m to see results.

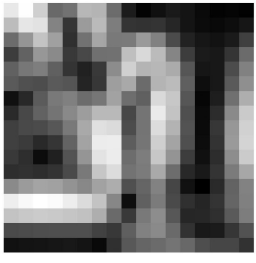
Additionally, due to some plot delay issue, this script can interpret the correct results only if you run it section by section. Otherwise, matlab may plot onto the wrong images.

Q3.

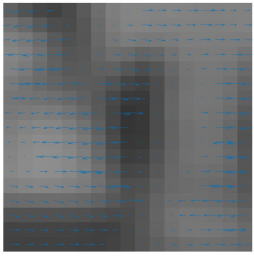
In this part, I detected all strong extrema (threshold = 25), and get this reasonable result.

Different colors represent the particular levels that keypoints are found

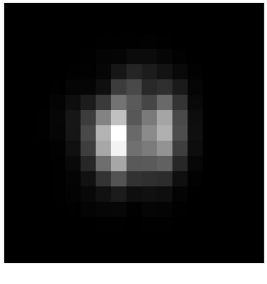
Q4.



magnitude of gradient of the neighbourhood of a selected keypoint



orientation of gradent



weighted magnitude of gradient.

The 17 × 17 gaussian kernel has sigma = 2.

In Q4, I chose to use 17×17 neighbourhood in order to put the keypoint at the center.

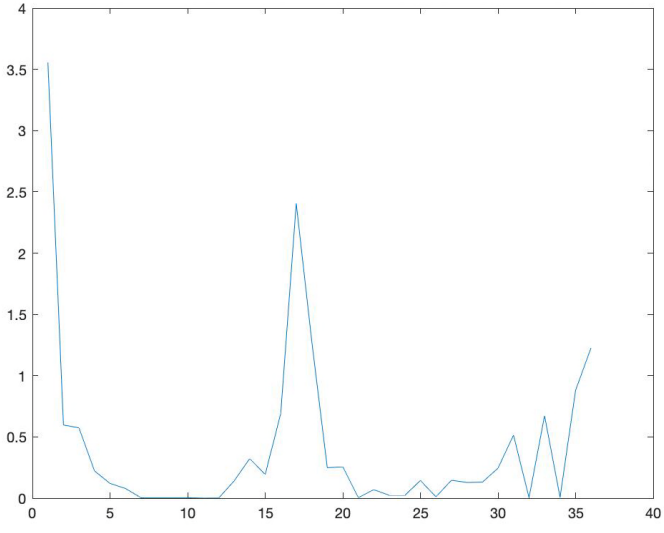
The code to implement this is in OriginalImageProcessing.m
And the helper function is Batches.m
Batches.m takes a keypoint and the gaussian image at the cbset level of this keypoint.

The gradient magnitude, orientation, weighted magnitude of neighbourhoods of all keypoints are in:

magnitude, orientation, weighted

And all of them are cell arrays.

Q5.



This is a sample histogram of the 20-th keypoint we found.

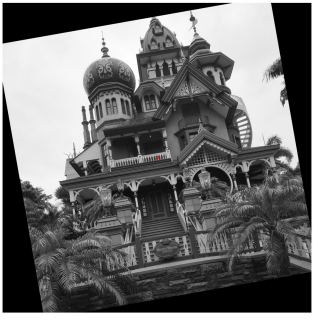
All keypoints are stored as 39-tuples. They are stored in finalKeypointTuple, which is a cell array.

Q6. Please run `rotate.m` to see results.
The helper function is `imageRotation.m`

Original

← rotate1

rotate 2 →



Rotation center : (512, 512)
Rotation : CW 10°
Scale : 1.1

Rotation center: (512, 512)
Rotation : CCW 10°
Scale : 0.9

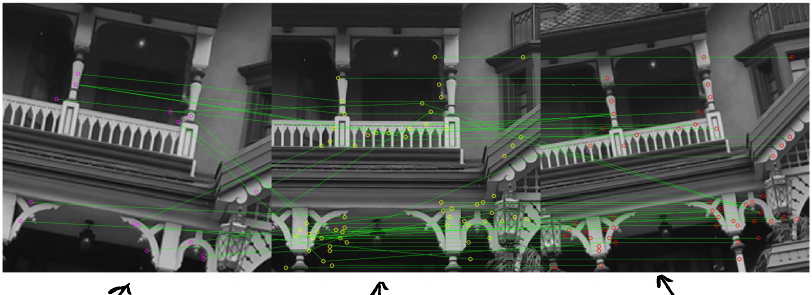
Q7. Please run OriginalImageProcessing.m to see result.
The interest region we pick is centered at (512, 512), and its width and height are 256. (square).

Here are the interest regions on 3 images.



Cropped images from: rotate1, original, rotate 2.

And then we try to math them.



rotate 1 original rotate 2.

Appoach:
For every possible pair of keypoints (p_1, p_2), where p_1 is from "original" and p_2 is from "rotate 1" or "rotate 2", we compare p_1 and p_2 by calculating Bhattacharya coefficient $B(p_1, p_2)$. And we set a thereshold 0.9, which means, if $B(p_1, p_2) \geq 0.9$, we say that (p_1, p_2) is a match.

The helper function Bhattacharya_coefficient.m is to calculate B_coefficient, and return 1 if $B(p_1, p_2)$ exceeds or is equal to threshold and return 0 otherwise.

As we can see, the matching result is not so good. There are many mismatches. One of my observation is that this matching appoach performs poorly in the aera having repeatitive patterns.
My thesis is that the keypoints in these aeras have similar features, that is why they match even though the threshold is relatively high.

They are two different aera, but they still match.

Q8. ($B(p_1, p_2)$ stands for Bhattacharya coefficient between p_1 and p_2)
Possible approach:
Say we pick a point p_1 from original image, and we want to match it with some point p_2 in "rotate 1".

We try every single point in "rotate 1", and calculate $B(p_1, p)$ for all sift point p in "rotate 1".
We rank these sift points w.r.t. $B(p_1, p)$.
Assume the first two places are p_2, p_2'.
If $B(p_1, p_2) / B(p_1, p_2') >$ some threshold, then we can say that p_2 is distinct enough and we can believe (p_1, p_2) is a valid match.
Otherwise, we say (p_1, p_2) is not valid enough, and we do not match them.

In this way, we can make sure that each pair is distinct enough, and the ratio of mismatching will reduce.