Homework 2 Writeup

Instructions

- This write-up is intended to be 'light'; its function is to help us grade your work.
- Please describe any interesting or non-standard decisions you made in writing your algorithm.
- Show your results and discuss any interesting findings.
- List any extra credit implementation and its results.
- Feel free to include code snippets, images, and equations.
- Use as many pages as you need, but err on the short side.
- **Please make this document anonymous.**

Assignment Overview

This assignment's task was to build a simplified SIFT feature matching algorithm. The goal was to match features from real-life images provided to us that may have slight differences, but are focused on the same subject.
Unfortunately, I was unable to get my code, which executes perfectly in the terminal, to run in the autograder. I asked the TAs and they said that the autograder itself was the issue and that I should upload my code along with my results. All of this is down below!

$$a = b + c \tag{1}$$

Implementation Detail

I implemented this code in a fairly standard way. In my get interest points method, I used the Harris Corner detection equation to get the cornerness score of the inputted images and then set a peak local max. I based my get features method heavily off of the psuedocode that I wrote for the written homework 2. I looped through the length of the features, then used more for loops to index into the bins where I filled my histogram with the magnitude, individually, for all 8 bins. Then, I appended my histogram to my empty descriptor array and expanded that array into my features, which I returned. My match features function calculates the euclidean distance between both sets of features and then uses the ratio test to index the best matches into my matches and confidences array.

I do not think that I used any unusual structures: I tried to make my code as standard as possible and based a lot of it on TA help/class slides. // Because my autograder was not functional, here are the screenshots of my code! The result images and terminal information are in results.

1. Here is my get interest points method:

```python
    # TODO: Your implementation here! See block comments and the project webpage for instructions

    # These are placeholders - replace with the coordinates of your interest points!

    xs = np.zeros(1)
    ys = np.zeros(1)

    # STEP 1: Calculate the gradient (partial derivatives on two directions).
    g_x = filters.sobel_v(image)
    g_y = filters.sobel_h(image)

    gx = np.square(g_x)
    gy = np.square(g_y)
    xy = np.multiply(gx, gy)

    # STEP 2: Apply Gaussian filter with appropriate sigma
    gx = filters.gaussian(gx, sigma = 1)
    gy = filters.gaussian(gy, sigma = 1)
    gxy = filters.gaussian(xy, sigma=1)

    g2 = np.square(gxy)
    a = 0.05
    # STEP 3: Calculate Harris cornerness score for all pixels.
    cornerness = (np.multiply(g_x, g_y) - g2) - (a * np.square(np.add(gx, gy)))
    # STEP 4: Peak local max to eliminate clusters. (Try different parameters.)
    max_m = feature.peak_local_max(cornerness, min_distance= 1, threshold_rel=0.03)
    xs = max_m[:, 1]
    ys = max_m[:, 0]

    # BONUS: There are some ways to improve:
    # 1. Making interest point detection multi-scaled.
    # 2. Use adaptive non-maximum suppression.

    return xs, ys
```

2. Here is my get features method:

```python
# TODO: Your implementation here! See block comments and the project webpage for instructions

# STEP 1: Calculate the gradient (partial derivatives on two directions) on all pixels.
# STEP 2: Decompose the gradient vectors to magnitude and direction.
# STEP 3: For each interest point, calculate the local histogram based on related 4x4 grid cells.
#         Each cell is a square with feature_width / 4 pixels length of side.
#         For each cell, we assign these gradient vectors corresponding to these pixels to 8 bins
#         based on the direction (angle) of the gradient vectors.
# STEP 4: Now for each cell, we have a 8-dimensional vector. Appending the vectors in the 4x4 cells,
#         we have a 128-dimensional feature.
# STEP 5: Don't forget to normalize your feature.

# BONUS: There are some ways to improve:
# 1. Use a multi-scaled feature descriptor.
# 2. Borrow ideas from GLOH or other type of feature descriptors.

# This is a placeholder - replace this with your features!

features = np.zeros((len(x), 128))

gx = filters.sobel_v(image, mask=None)
gy = filters.sobel_h(image, mask=None)

mag = np.sqrt(np.add(np.square(gx), np.square(gy)))

grad_o = np.add(np.arctan2(gx, gy), np.pi)


for i in range(0, len(x)):
```

```python
for i in range(0, len(x)):
    des = np.array([])
    if (x[i] + 8 < image.shape[1]) and (y[i] + 8 < image.shape[0]):
        for outerY in range(int(y[i]) - 8, int(y[i]) + 8, 4):
            for outerX in range(int(x[i]) - 8, int(x[i]) + 8, 4):
                histogram = np.zeros((8, 1))
                for innerY in range(outerY, outerY + 4):
                    for innerX in range(outerX, outerX + 4):
                        orientation = grad_o[innerY][innerX]
                        mag_help = mag[innerY][innerX]
                        if (orientation >= 0) and (orientation <= 1/4 * np.pi):
                            histogram[0] += mag_help
                        elif (orientation > 1/4 * np.pi) and (orientation < 1/2 * np.pi):
                            histogram[1] += mag_help
                        elif (orientation > 1/2 * np.pi) and (orientation < 3/4 * np.pi):
                            histogram[2] += mag_help
                        elif (orientation > 3/4 * np.pi) and (orientation < np.pi):
                            histogram[3] += mag_help
                        elif(orientation > np.pi) and (orientation < 5/4 * np.pi):
                            histogram[4] += mag_help
                        elif(orientation > 5/4 * np.pi) and (orientation < 3/2 * np.pi):
                            histogram[5] += mag_help
                        elif(orientation > 3/2 * np.pi/2) and (orientation < 7/4 * (np.pi)):
                            histogram[6] += mag_help
                        elif (orientation > 7/4 * np.pi) and (orientation < np.pi):
                            histogram[7] += mag_help

                des = np.append(des, histogram)

        features[i, :] = np.expand_dims(des / np.linalg.norm(des), axis = 0)

return features
```

3. Here is my match features method:

```
matches = np.zeros((len(im1_features),2))
confidences = np.zeros(len(im1_features))

B = 2 * (np.dot(im1_features, np.transpose(im2_features)))

f1_sum = np.sum(np.square(im1_features), axis=1, keepdims=True)
f2_sum = np.sum(np.square(im2_features), axis=1, keepdims=True)


f2_sum = np.transpose(f2_sum)

A = np.add(f1_sum, f2_sum)

e_dist = np.sqrt(np.subtract(A, B))


d_sort_i = np.argsort(e_dist)


for i in range(len(im1_features)):

    near_n = e_dist[i][d_sort_i[i][0]]
    near_n_2 = e_dist[i][d_sort_i[i][1]]

    if near_n_2 != 0:
        ratio = 1 - near_n/near_n_2
        confidences[i] = ratio
        matches[i][0] = i
        matches[i][1] = d_sort_i[i][0]


return matches, confidences
```

Result

One commonality I noticed in my results is that my algorithm detects many points of interest, and that this could potentially skew my accuracy measurements because even though it finds a lot of matches, it does not match most of the features that it finds. However, I am quite proud of my visual results!

1. Result 1 (Figure 1, left) is the picture result of my notre dame image.
2. Result 2 (Figure 1, right) is the picture result for my e gaudi image.

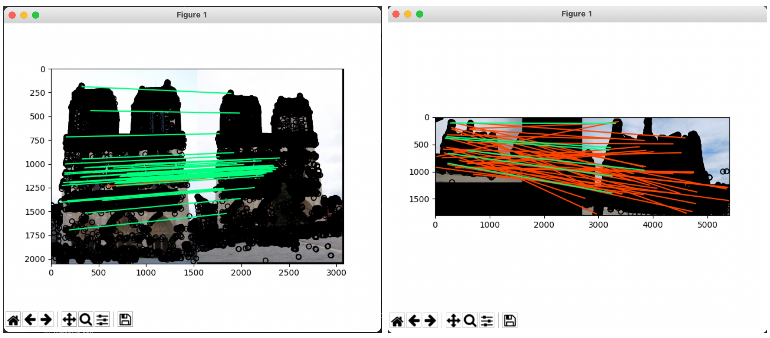

Figure 1: *Left:* Result for notre dame *Right:* result for e gaudi

1. Result 3 (Figure 2, left) is the picture result of my mt rushmore image.
2. Result 4 (Figure 2, right) is what runs in the terminal when I execute my code: AKA, all of my accuracy measurements.

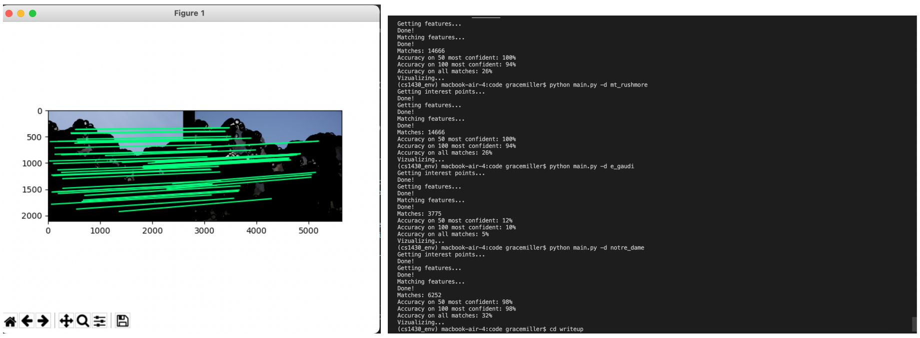

Figure 2: *Left:* Result for mt rushmore *Right:* terminal output