Project 2 Writeup: Image Feature Detection

Dustin Cook

Description

The purpose of this project is to find corresponding features from two images that are of the same object. This can appear challenging at first, and there are a wide range of methods, but my algorithm goes as the following.

What makes a good feature? Answer: a corner. When taking the x and y gradient of an image, a corner has a high and kind of equal rate of change in both the x and y gradient.

I chose the Harris method for corner detection. I implemented his algorithm in H_detectFeatures(I, alpha, thresh, returnVal). There is still a problem though. We need our features to be invariant to: scale, illumination, rotation, etc. In an attempt to make these features invariant to scale, I wrote another function called harrisDetection(I, baseSigma, maxBoxSize, thresh). What this function does is: it decimates the image by three scales and runs H_detectFeatures(params) on the different scales. We keep track of the scale, orientation and location of the features per scale. We then compute the difference of Gaussians given by the function: DoG(I, baseSigma). Then, for every feature we found, we make sure it exists in the difference of Gaussians. This should make the corners we found invariant to scale. As for rotation invariance, we kind of get that for free with the Harris method.

After getting the features from image 1 and image 2 - I1, I2 respectfully, we now need to match them. To do this, I wrote a function called matchFeatures(I1, I2, maxDistance, CT, baseSigma, threshold, sliceSize). There are a lot of parameters, there is a description per parameter in the file matchFeatures.m, but I will also provide examples. I1, I2 are the two images, CT is a corner threshold but it means something else. From what I saw, The corners of the image often flared as features which was normally not correct, so CT determines what distance does the feature have to be from the border of the image. baseSigma is used by DoG - difference of gaussians function. The sigma is doubled per Gaussian filter. sliceSize is the size of the image descriptor. In order to get a descriptor, this function calls convertToVectors(I, F, s) where I is the untampered image, F is a list of features found for that image, and s is the descriptor side length: the descriptor will be sxs. This function will make the descriptor invariant to scale by ordering the image values by the angle essentially taking out the descriptor and bringing it to 0 degrees so that it can be compared in the other image without rotation getting in the way.

Now that we have our features and descriptors per feature, we can brute force match them. Using SSD(v1, v2), for every descriptor for I1, we find its best and second best match for the descriptors for I2. If the ratio test passes, it is added to our set of matches. We then draw the matches with the function drawMatches(I1, I2, matches, brushSize). This function is pretty simple, it just takes the two original images and the matches computed by matchFeatures, and a desired brushSize to draw lines between where the algorithm believes a match to be. The variables can be fine tuned to your liking. I have found that every particular set of images requires its own set of params to find the features properly. If the maxDistance and threshold are too low you may find a lot of your features corresponding to the same location. It is also important to balance sliceSize with maxDistance.

Does it work?

Figure 1: Running detection on itself.

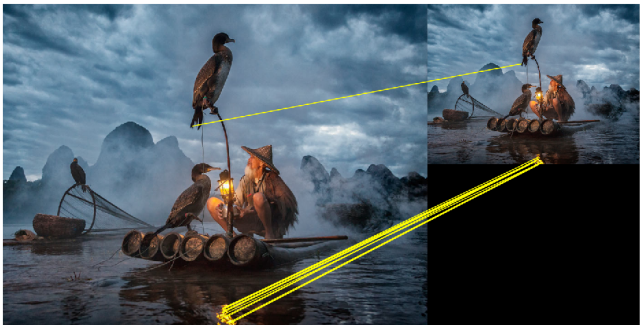

Figure 2: Running detection on itself decimated by half.

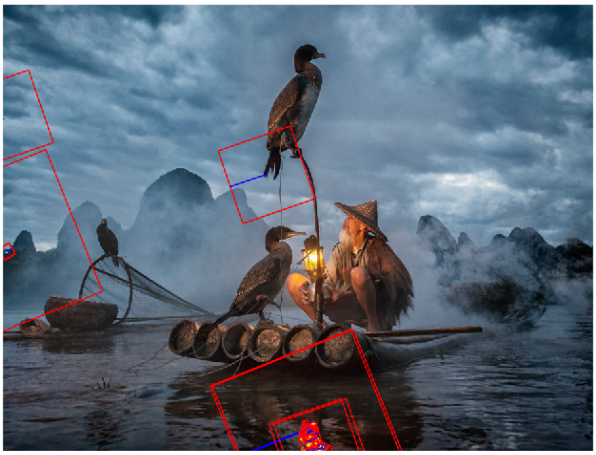

Figure 3: Feature location, scale, and orientation. Looks very promising!

Figure 1 - 3 can be achieved with the following code:

```
1        I1 = imread('river.jpg');
2        I2 = I1;
3        imshow(drawMatches(I1, I2, matchFeatures(I1, I2, 10,
             5, 3, .3, 6), 2));
4        imshow(drawMatches(I1, decimate(I2,2), matchFeatures
            (I1, decimate(I2,2), 10, 5, 3, .3, 6), 2));
5        imshow( ShowFeatures(I1, harrisDetection(I1, 3, 20,
            .3), 3) );
```

Rotation Invariant

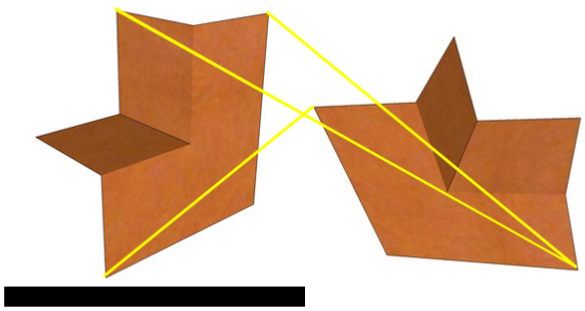

Figure 4: Image rotated 90 degrees clockwise

A Result

We are invariant to:

- Scale: Kind of (starts to false positive after x2 decimation).
- Illumination: No.
- Distortion: No.
- Rotation: Yes.

How does my algorithm work for the benchmark images?

not well..

Figure 5: Results from increasing blur in image.

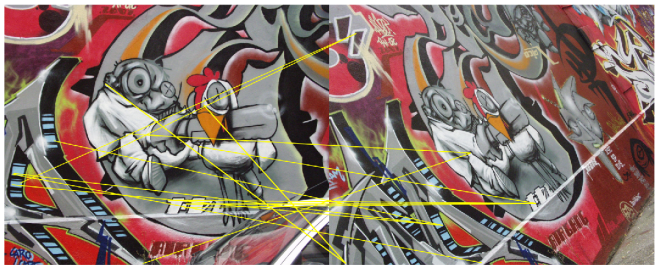

Figure 7: Results from increasing illumination change in image.

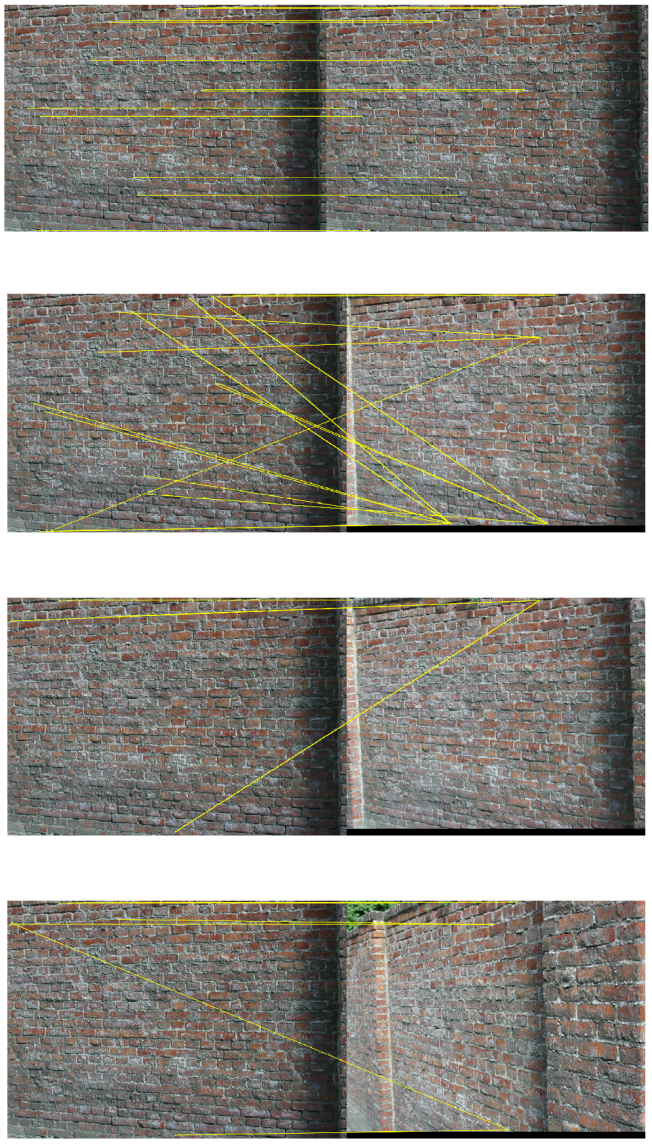

Figure 8: Results from increasing distortion change in image.

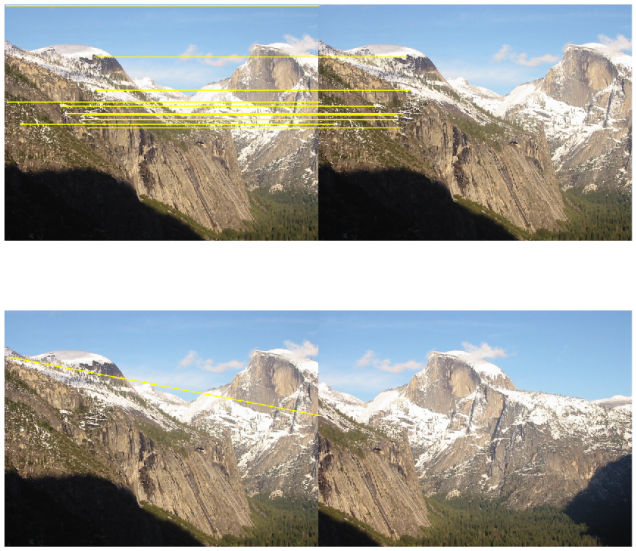

Figure 9: We found one!