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.

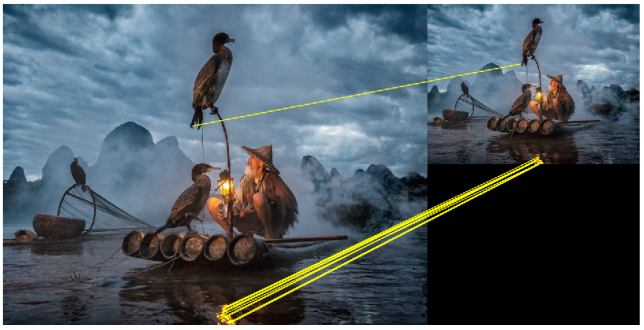

Figure 2: Running detection on itself decimated by half.

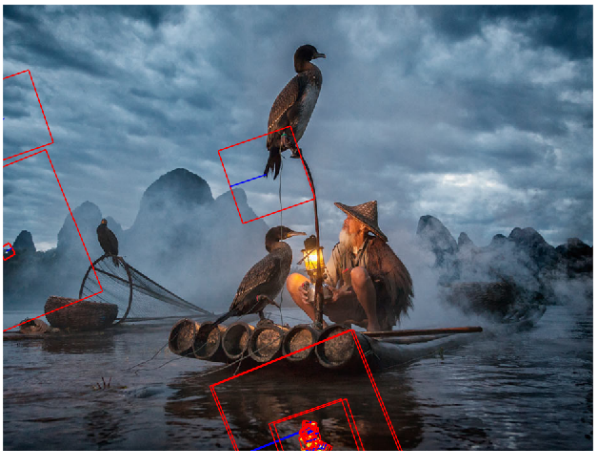

Figure 3: Feature location, scale, and orientation. Looks very promising!

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

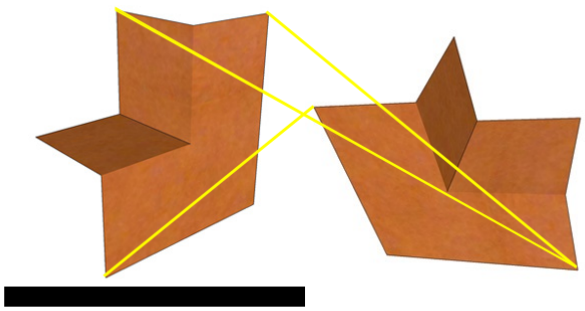

Figure 4: Image rotated 90 degrees clockwise

matchFeatures

Go to page 14 to skip code snippets.

```
1  function [matches] = matchFeatures(I1, I2, maxDistance, CT,
       baseSigma, threshold, sliceSize)
2      % given two untampered images I1, I2
3      % this function returns an arraylist consisting of
           feature pairs
4      % maxDistance - the allowable amount of difference two
           features may
5      % have to be considered matching
6      % CT - Corner threshold, how far away does it have to be
            from
7      % the corners because, normally, features close to the
           outter corners
8      % are bad.
9
10     % the 20 here is arbitrary, we dont actually care about
           drawing
11     % these features
12     f1 = harrisDetection(I1, baseSigma, 20, threshold);
13     f2 = harrisDetection(I2, baseSigma, 20, threshold);
14
15
16     % filter out corners of image%%%%%%%%%%%%%%%%%%%%%%%%%%
17     % there is probably a better way to do this.
18     indices = find(f1(:,1) < CT);
19     f1(indices,:) = [];
20     indices = find(f1(:,2) < CT);
21     f1(indices,:) = [];
22
23     indices = find(f2(:,1) < CT);
24     f2(indices,:) = [];
25     indices = find(f2(:,2) < CT);
26     f2(indices,:) = [];
27
28     maxX = size(I1, 1) - CT;
29     maxY = size(I1, 2) - CT;
30
31     indices = find(f1(:,1) > maxX);
```

```
32      f1(indices,:) = [];
33      indices = find(f1(:,2) > maxY);
34      f1(indices,:) = [];
35
36      maxX = size(I2, 1) - CT;
37      maxY = size(I2, 2) - CT;
38
39      indices = find(f2(:,1) > maxX);
40      f2(indices,:) = [];
41      indices = find(f2(:,2) > maxY);
42      f2(indices,:) = [];
43      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44
45      v1 = convertToVectors(I1, f1, sliceSize);
46      v2 = convertToVectors(I2, f2, sliceSize);
47
48      matches = java.util.ArrayList();
49
50
51      % for every v1,
52      % get its min SSD from v2
53      % if the minSSD is allowable,
54      % match them
55      for i = 0 : v1.size() - 1
56          minSSD = intmax;
57          prevMinSSD = intmax;
58          minMatch = [];
59          vec1 = v1.get(i);
60          if (v2.size() > 0) % wouldnt make sense to run with
                no values
61              for j = 0 : v2.size() - 1
62                  newSSD = SSD(vec1, v2.get(j));
63                  if minSSD > newSSD
64                      % pair them
65                      prevMinSSD = minSSD;
66                      minSSD = newSSD;
67                      minMatch = [f1(i+1,:) ; f2(j + 1,:)];
68                  end
69              end
70
71              % now that we have the minSSD
72              % if it is within our allowable distance
```

```
73              % make the match
74              if isempty(minMatch) == false && (minSSD /
                    prevMinSSD) < maxDistance
75                  matches.add(minMatch);
76                  %v2.remove(minMatchID);
77              end
78          end
79      end
80  end
```

harrisDetection

```
1   function [interestPoints] = harrisDetection(I, baseSigma,
        maxBoxSize, thresh)
2       % returns a set of points where each
3       % row is x y size theta
4       % I - The image
5       % baseSigma - steps to 10% image size from 100%
6       % maxBoxSize - the largest size of a box, gets doubled
            twice per
7       % scale
8       % thresh - increase to get less features, lower to get
            more
9
10      O = I;
11      maxY = size(I, 1);
12      maxX = size(I, 2);
13
14      % 3d matrix where an interest point on the image
            corresponds to an
15      % index in this matrix A(x, y, :) where : is a size 3
            matrix of Value,
16      % theta, size
17      interestPointsMatrix = zeros(maxY, maxX, 3);
18
19      % do harris corner detection to get points of interest
20      boxSize = maxBoxSize;
21
22      interestPointsList = H_detectFeatures(I, .005, thresh, '
            p');
```

```
23
24      interestPointsList(:,4) = boxSize;
25
26      for j = 1 : size(interestPointsList, 1)
27              x = interestPointsList(j, 1);
28              y = interestPointsList(j, 2);
29              % flag interest point at image location
30              interestPointsMatrix(y, x, :) = [1
                    interestPointsList(j, 3:4)];
31      end
32
33      % decimate image to three scales
34      for i = 1 : 2
35          %dog = DoG(I, baseSigma);
36          I = decimate(I, 2);
37          boxSize = boxSize / 2;
38
39          % H_detect returns set of [x y theta]
40          % append the size we found the point at
41          smallerPointsList = H_detectFeatures(I, .005, thresh
                , 'p');
42          smallerPointsList(:,4) = boxSize;
43
44          % add all points to our interest points list
45          % when we decimate an image, we remove every other
                row and column,
46          % so when we detect features on a decimated image,
                we know that we
47          % are taking 2*decimation steps instead of one
48          for j = 1 : size(smallerPointsList, 1)
49              x = min(smallerPointsList(j, 1) * (2^i), maxX);
50              y = min(smallerPointsList(j, 2) * (2^i), maxY);
51
52              % flag interest point at image location if not
                    already flaged
53              if (interestPointsMatrix(y, x, 1) == 0)
54
55                  % add to matrix
56                  interestPointsMatrix(y, x, :) = [1
                        smallerPointsList(3:4)];
57              end
58          end
```

```
59      end
60
61      % using difference of gaussians, filter out points that
           arent scale
62      % invarient
63
64      dog = DoG(O, baseSigma);
65
66      for i = 2 : size(dog, 3)
67          dog(:,:,1) = dog(:,:,1) .* dog(:,:,i);
68      end
69
70      interestPointsMatrix(:,:,1) = interestPointsMatrix
           (:,:,1) .* dog(:,:,1);
71  %     imshow([interestPointsMatrix(:,:,1) dog(:,:,1)]);
72
73      interestPoints = [0 0 0 0];
74
75      for y = 1 : size(interestPointsMatrix, 1)
76          for x = 1 : size(interestPointsMatrix, 2)
77              if (interestPointsMatrix(y, x, 1) ~= 0)
78                  interestPoints = [interestPoints ; x y
                       interestPointsMatrix(y, x, 2)
                       interestPointsMatrix(y, x, 3)];
79              end
80          end
81      end
82
83      interestPoints = interestPoints(2:end, :);
84  end
```

H_detectFeatures

```
1   function [P] = H_detectFeatures(I, alpha, thresh, returnVal)
2       % Returns a set of x y theta
3       % points of distinct image features
4       % This function is for a singular level
5       % for harris detection with sigma see: harrisDetection
6       % returnVal - 'img' for img, 'p' for points
7
```

```
8       % save original for display purposes
9       O = I;
10      I = rgb2gray(I);
11      I = im2double(I);
12
13
14      dKernel = [3 3];
15
16      Ix = imfilter(I, XGradient(dKernel), 'conv');
17      Iy = imfilter(I, YGradient(dKernel), 'conv');
18
19      % get Ixy
20      Ixy = Ix .* Iy;
21
22      % square them
23      Ix = Ix .* Ix;
24      Iy = Iy .* Iy;
25      Ixy = Ixy .* Ixy;
26
27      % convolve with gaussian filter
28      g = fspecial('Gaussian', [60 60]);
29
30      Ix = imfilter(Ix, g, 'conv');
31      Iy = imfilter(Iy, g, 'conv');
32      Ixy = imfilter(Ixy, g, 'conv');
33
34
35      % Harris corner detection
36      % we want points that are strong in both Ix and Iy
37      % these are corners and they are good candidates for
38      % describing an image
39      F = Ix .* Iy - (Ixy .* Ixy);
40      F = F - alpha .* ((Ix + Iy) .* (Ix + Iy));
41
42      % threshold image by user threshold
43      for point = find(F < thresh)
44          F(point) = 0;
45      end
46
47
48      % blob detect with laplacian
49      %lFilter = fspecial('laplacian', .1);
```

```
50      %L = imfilter(I, lFilter);
51      %F = F .* L;
52
53      % select local maximas
54      windowSize = 3;
55      maximum = windowSize^2;
56      maxes = ordfilt2(F, maximum, ones(windowSize, windowSize
            ));
57      thresh = (mean(maxes) + max(maxes)) / 2;
58      corners = (F == maxes) & (maxes > thresh);
59
60      % get corners, their gradients, and the size
61      IND = find(corners);
62      [Y, X] = ind2sub(size(corners), IND);
63      P = [X Y];
64      gradients = double(zeros(size(Y)));
65
66      for i = 1 : size(Y)
67          y = Y(i);
68          x = X(i);
69          gradients(i) = atan((Iy(y, x) / Ix(y, x)));
70      end
71
72
73      % if returnVal is p, return the points with orientation
            array
74      % else return a visual
75      if strcmp(returnVal, 'p')
76          P = [P gradients];
77      else
78          [p,q] = size(maxes);
79          P = zeros(p, q*2);
80          P(1:p, 1:q) = I;
81          P(1:p, q+1:end) = corners;
82      end
83  end
```

DoG

```
1   function [dogList] = DoG(I, baseSigma)
```

```
    I = rgb2gray(I);
    % I - the image
    % baseSigma - for gaussian multiplied by 2 per filter
    % returns a 3d matrix of the difference of gaussians, IxIx4 - for 4
    % DoGs

    gaussianList = java.util.ArrayList();

    % convolve image with gaussians
    % increasing sigma per index by 2
    for i = 1 : 5
        g = fspecial('gaussian', [25, 25], baseSigma);
        newG = imfilter(I, g);
        gaussianList.add(newG);
        baseSigma = baseSigma * 2;
    end


    %A = [];
    %for i = 0 : gaussianList.size() - 1
    %    A = [A gaussianList.get(i)];
    %end
    %imshow(A);

    % compute difference of gaussian array
    dogList = zeros(size(I, 1), size(I,2), 3);

    for i = 0 : gaussianList.size() - 2
        dogList(:, :, i + 1) = gaussianList.get(i) - gaussianList.get(i + 1);
    end



    %A = [];
    %for i = 0 : dogList.size() - 1
    %    A = [A dogList.get(i)];
    %end
    %imshow(A);

end
```

convertFeature

This function is called on every feature given by harrisDetection.

```
function [V] = convertFeature(I, xc, yc, theta, s)
    % takes a features location, angle and scale
    % and orders every pixel into a s^2x3 array
    % where every rows values represent the intensities of:
    % 1 -> R, 2 -> G, 3 -> B
    % Params:
    % I - the original un altered image
    % xc - feature x location
    % yc - feature y location
    % theta - feature angle
    % s - feature scale

    V = zeros(s^2, 3);
    h = s / sqrt(2);
    x = xc + h * cos(135 + theta);
    y = yc + h * sin(135 + theta);

    for i = 0 : s - 1
        x = x + cos(270 + theta);
        y = y - sin(270 + theta);
        V(i*s+1 : i*s+s, :) = addPoints(I, floor(x), floor(y
            ), theta, s);
    end
end
```

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.

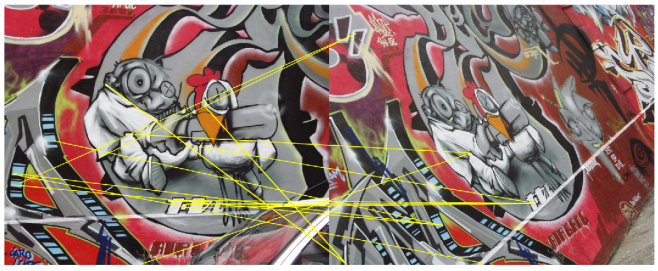

Figure 7: Results from increasing illumination change in image.

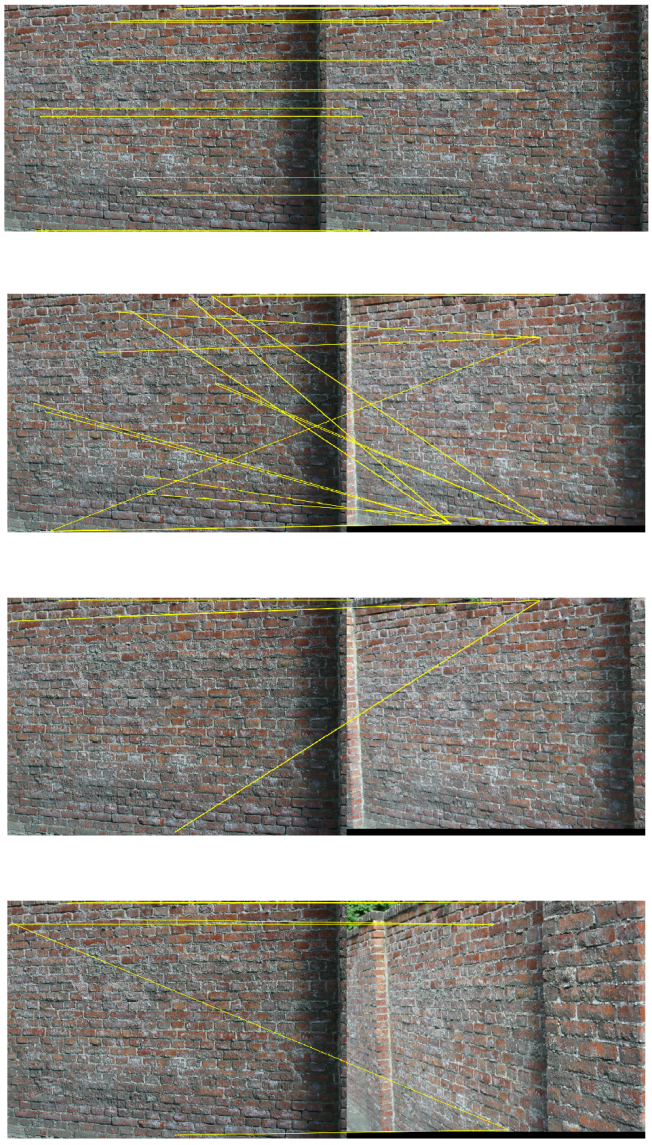

Figure 8: Results from increasing distortion change in image.

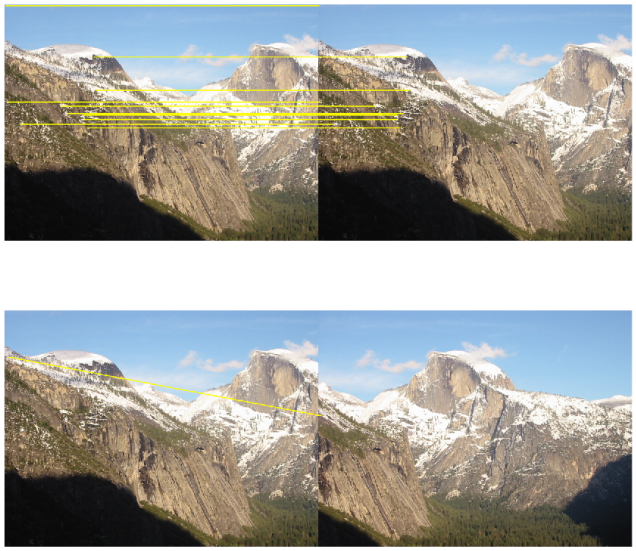

Figure 9: We found one!