Australian National University

ENGN6258

Computer Vision

CLAB 3 - Image Features Report

Author:
Siddharth Pethe

Student Number:
U5973641

23rd April 2017

Contents

1 Task 1 : Harris Corners

The task involved finding Corner features within the image using the Harris Cornering parameter R cornerness value.

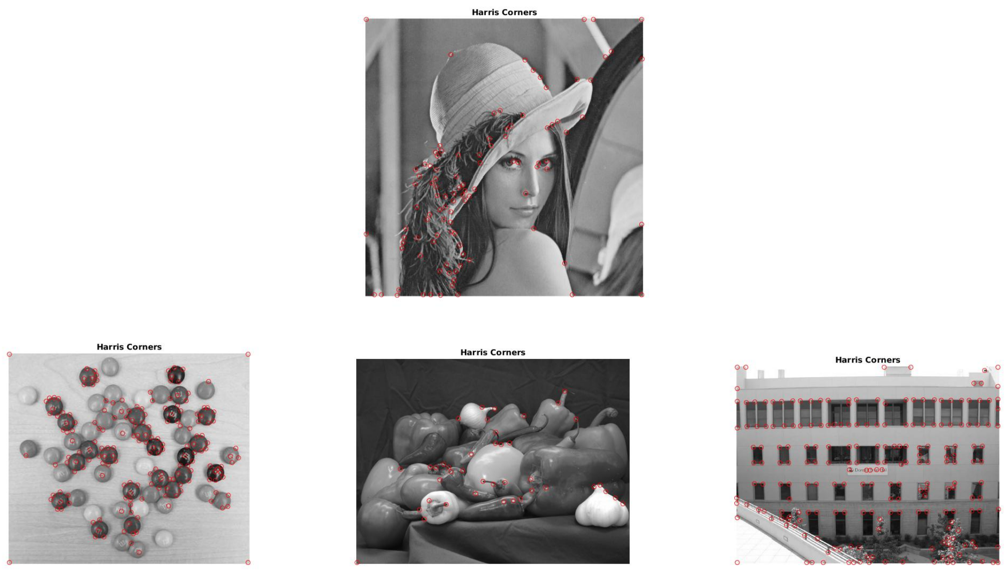

Figure 1: Harris Corners using cornerness R after thresholding and non-maximal supression using 11x11 mask.

The code was straight forward. I calculated cornerness value R for each point in the bw-image using determinant and 0.1trace. Refer Appendix A for code.

We were already provided the second moment matrix by using gaussian mask to mimic small movement.

1.1 Thresholding:

If the value of R was lesser than 0 or very small, then the feature was an edge or a flat region. The points having value of R lesser than 0.01 were set to 0 because these were not corners.

1.2 Non-maximal Suppression:

Using an 11x11 mask we found 5 values before and 5 values after the current value. If the current value was maximum among all these, then the it was retained; else it was set to 0.
This way we got a matrix which contained R values of only those points that were

max in a 11x11 window and above threshold. These points were then superimposed on the original image to show corners by using the index or all values in R matrix with non-zero values.

2 Task 2: Image Matching using SIFT descriptors

This task involved the use of provided function to find SIFT keypoints for an image and match two forms of the same image by comparing the SIFT descriptors of the two images. Refer Appendix B for code.

2.1 Inputs:

First we took an input image and rotated it by a given angle. Then we rescaled it by a certain factor. This gave us a new image have the same features as the first image, but located at a different place in the image. Also, they were of a different size as well due to rescaling.

2.2 SIFT keypoints:

Then we gave these images to the SIFT function to find keypoints. The SIFT function, took the images and returned 128-point descriptors for each feature in the images along with the location of the descriptor.

- Now, I used these descriptors and found the euclidean distance between a descriptor in input image and all descriptors in the output image.
- I then found sorted these distances in ascending order using sort.
- Taking the ratio of the distances between the first smallest distance (best match) and second smallest distance (second-best match), I checked if the feature was indeed a unique match.

- For all unique matches, I saved the location of feature in input image and location in rotated and rescaled image in a 10x2 matrix.
- Thus we got first 10 SIFT feature matches.

Once the matches were obtained, I simply appended the two images side by side and displayed a line between the matches.

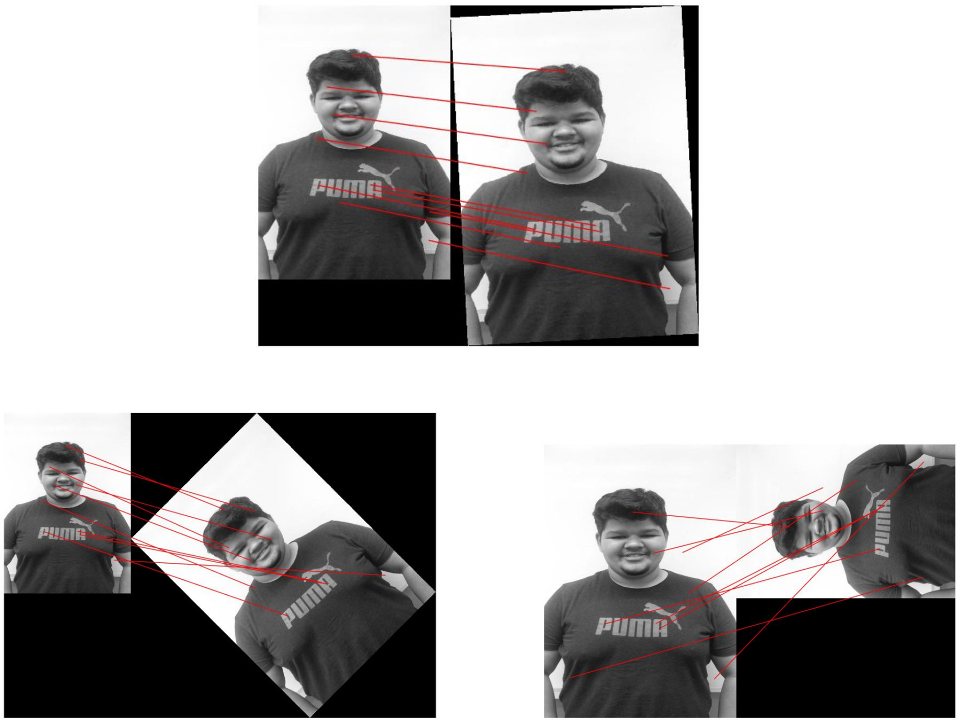

Figure 2: SIFT keypoints and their respective best matches

3 Task 3: Colour Segmentation using K-means Clustering

This task involved use of K-means clustering method by specifying the number of clusters and then finding cluster means.

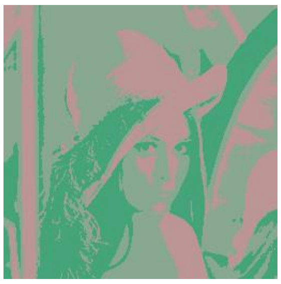

Figure 3: Colour Segmentation using K-means after 5 iterations only

3.1 Creating 5 dimensions for each pixel and normalizing:

The first task here was to input a color image. Then we convert that image to L-A-B form.
Using the im2feature() function, we created a data structure containing the 5 dimensional information for each pixel. The output contained as many rows as there were pixels in image and 5 columns.
But we had to reduce the values of x and y parameters to between 0 and 1, just as the rest of the elements. Not doing so would mean that X-Y played a bigger role in segmentation and means compared to the other values i.e. l*,a*,b*. By diving each value by max(x) and max(y), I was able to achieve this.

3.2 K-Means Clustering

We passed the data structure to mykmeans() function. We kept iterating until the difference between clustering in 2 successive iterations was very small. At first we assigned random clustering for each pixel.

- In each iteration, we first calculate the cluster mean for each dimension based on mean of all pixels belonging to that cluster.

- Next, we proceed pixel-by-pixel, finding the euclidean distance of that pixel to each cluster center. Whichever distance is minimum, we assign the pixel to that cluster. Refer Appendix C for code.
- Then back to the next iteration, we find mean again compare for exit criteria and continue if not close enough.

3.3 Displaying image with clustering:

I could not understand the display clusters method so I wrote my own. I took each pixel and found its cluster membership. Then I simply assigned the mean value of that cluster to that pixel in the new colour segmented image.

```
im_clustered=zeros(size(lab));
pixel=0;
for row=1:size(lab,2)
    for col=1:size(lab,1)
        pixel=pixel+1;
        switch data_clusters(pixel)
            case 1
                im_clustered(col,row,1)= cluster_stats(1,2);
                im_clustered(col,row,2)= cluster_stats(1,3);
                im_clustered(col,row,3)= cluster_stats(1,4);
            case 2
                im_clustered(col,row,1)= cluster_stats(2,2);
                im_clustered(col,row,2)= cluster_stats(2,3);
                im_clustered(col,row,3)= cluster_stats(2,4);
            case 3
                im_clustered(col,row,1)= cluster_stats(3,2);
                im_clustered(col,row,2)= cluster_stats(3,3);
                im_clustered(col,row,3)= cluster_stats(3,4);
            otherwise
                im_clustered(col,row)=0;
        end

    end
end
```

Appendix A Task 1:

Listing 1: R-cornerness value

```
R=zeros(size(Ix2,1),size(Ix2,2));
R1=zeros(size(Ix2,1),size(Ix2,2));
for row=1:size(Ix2,1)
    for col=1:size(Ix2,2)
        H=[Ix2(row,col),Ixy(row,col);Ixy(row,col),Iy2(row,col)];
        dH=det(H);trH=trace(H);
        R1(row,col)=dH-(0.01*(trH^2));
    end
end
```

Listing 2: Thresholding

```

for row=1:size(R,1)
    for col=1:size(R,2)
         if  ((R1(row,col))<thresh)
             R1(row,col)=0;
         end
    end
end
```

Listing 3: Non-maximal Suppression for sze X sze mask.

```
for row=1:size(R1,1)
    for col=1:size(R1,2)
         max=0;
         for rowt=1:sze
             for colt=1:sze
                if(row-fix(sze/2)+rowt<=0||col-fix(sze/2)+colt<=0||row-
                    continue;
                end

                if max<R1(row-fix(sze/2)+rowt,col-fix(sze/2)+colt)
                    max=R1(row-fix(sze/2)+rowt,col-fix(sze/2)+colt);
                end
             end
         end
         if R1(row,col)==max
             R(row,col)=R1(row,col);
```

```
        end

    end
end
```

Appendix B Task 2

Listing 4: finding matches using SIFT

```

[im_in, des, loc]=sift('im_in.jpg');
[im_rz_in, des_rz, loc_rz]=sift('im_rz.jpg');

dist=pdist2(des,des_rz,'euclidean');
ratio=0.8;
links=0;
match=zeros(10,2);
for i=1:size(des,1)
    [dist_sort, I]=sort(dist(i,:));
    if (dist_sort(1)/dist_sort(2))<=ratio
        links=links+1;
        match(links,1)=i;
        match(links,2)=I(1);
    end
    if links>=10
        break;
    end

end
```

Appendix C Task 3

Here I have only inserted the code for a part of mykmeans() function because the rest was already given. The unique part of the main function is already shown in the main report.

Listing 5: Finding distance to each cluster center and changing membership for pixel

```
dist_center=zeros(size(data,1),1);
dist_c=zeros(nc,1);
       for it=1:size(data,1)
       for ite=1:nc
           dist_c(ite)=pdist2(double(data(it,:)),double(cluster_stats(
       end
       [dist_center(it), data_clusters(it)]=min(dist_c);

   end
```