Assignment 3: Corner Detection

Ana Maria Martinez Sidera
Department of Computer Science
Illinois Institute of Technology

October 22, 2017

1. Problem statement

The problem that we are going to solve is the detection of corners and features in two different images. The steps we are going to take to perform the problem are:

1. Prepare and display two images, which were taken from the same area but have different perspectives.
2. Estimate the gradient of each image and calculate the Harris corner algorithm to detect them in each of the photos.
3. Obtain a location of each corner.
4. Get a vector that link the most relevant features of the corners we have found.
5. Visualize the photos in black and white, with the corners found in a empty rectangle and with the vectors.
6. Control with a trackbar the variance of the Gaussian to perform a smooth to each image.
7. Control with a trackbar the neighbors that we use in Harris algorithm.
8. Control with a trackbar the weight of the trace in the Harris corner detector.
9. Control with a trackbar a threshold to control the number of corners that we will visualize.
10. Special keys on the keyboard should be used to modify the displayed image as follows:
    (a) 'h' - Display a short description of the program, its command line arguments, and the keys it supports.

2. Proposed solution

1. The first step we have to do for this problem is to visualize the two images concatenated side by side. Also, convert the images to gray.
    (a) We read the pictures with their names in the path.
    (b) We change the color to gray and also the channel of the images.
    (c) Concatenate them in a single one in the direction of x.

```python
imagen1 = cv2.imread(name1)
imagen2 = cv2.imread(name2)
imagen1 = grayscale(imagen1)
imagen2 = grayscale(imagen2)
imagen_final = combinar(imagen1, imagen2)

def combinar(image1, image2):
    image_combine = np.concatenate((image1, image2), axis=1)
    return image_combine
```

2. In the same image we created the four trackbars that will serve us to control the different images that we pass to the program or the solutions that we want to appear to us.

   (a) In each trackbar, we update the value of R in each pixel.

```python
cv2.imshow('CS_512_imagen6', imagen_final)
cv2.createTrackbar('Gaussian','CS_512_imagen6', 1, 255, sliderHandler_1)
cv2.createTrackbar('Neighborhood','CS_512_imagen6', 1, 10, sliderHandler_2)
cv2.createTrackbar('Weight','CS_512_imagen6', 40, 60, sliderHandler_3)
cv2.createTrackbar('Threshold','CS_512_imagen6', 50, 100, sliderHandler_4)

def sliderHandler_1(self):
    if self != 0:
        update()

def sliderHandler_2(self):
    update()

def sliderHandler_3(self):
    if self >=40 and self <= 60:
        update()

def sliderHandler_4(self):
    if self != 0:
        update()
```

3. We calculate the gradient of each image and normalize it.

```python
imagenxx = cv2.Sobel(imagen_new_1,cv2.CV_64F,1,0,ksize=5)
Ixx = cv2.normalize(imagenxx, imagen_new_1, alpha = 0, beta = 1,
    norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_64F)
imagenyy = cv2.Sobel(imagen_new_2,cv2.CV_64F,1,0,ksize=5)
Iyy = cv2.normalize(imagenyy, imagen_new_2, alpha = 0, beta = 1,
    norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_64F)
Ixx = Ixx**2
```

```
Iyy = Iyy**2
Ixy = Iyy*Ixx
```

4. We apply the Harris algorithm to detect which are possible corner in our images and save the R value in each pixel in a list.

```
height = Ixx.shape[0]
width = Ixx.shape[1]
for y in range(neighborhood, height-neighborhood):
    for x in range(neighborhood, width-neighborhood):
        windowIxx = Ixx[y-neighborhood:y+neighborhood+1,
            x-neighborhood:x+neighborhood+1]
        windowIxy = Ixy[y-neighborhood:y+neighborhood+1,
            x-neighborhood:x+neighborhood+1]
        windowIyy = Iyy[y-neighborhood:y+neighborhood+1,
        x-neighborhood:x+neighborhood+1]
        Sxx = windowIxx.sum()
        Sxy = windowIxy.sum()
        Syy = windowIyy.sum()
        M = np.matrix([[Sxx,Sxy],[Sxy,Syy]])
        det = np.linalg.det(M)
        tr = np.trace(M)
        r = det - weight*tr**2
```

5. Then we store the value of each x, y and the value r of each pixel.
   We also calculate the maximum R of each image. Thanks to this, we could perform the threshold trackbar as a percent of the total corners in the image.

```
r = det - weight*tr**2
cornerList.append([x, y, r])
if r > r_max:
    r_max = r
```

6. The next step we will do is find the characteristics of each image, concatenate the images and create a vector that unites them in the two images.
   (a) First we look in each image for the features in each pixel.
   (b) We pair between all the pixels of each image to see which features matches the best.
   (c) We choose the features with the lowest distance that are the ones that have a better similarity takint into account the threshold.
   (d) We draw the vectors that join the points.

```
def featurevectores(imagen):
    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(imagen,None)
    return kp1, des1

def features():
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    feature_x1,feature_y1 = featurevectores(img1)
    feature_x2,feature_y2 = featurevectores(img2)
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(feature_y1,feature_y2, k=2)
    good = []
    for m,n in matches:
        if m.distance < threshold*n.distance:
            good.append([m])
    print ("Features Corners...")
    imagen_final = cv2.drawMatchesKnn(img1,feature_x1,img2,feature_x2,good,
    imagen_final,flags=2)
```

7. After performing all the arithmetic operations with the images, we will draw on both images the corners that we have found previously.

   (a) We save the variables *(x,y,r)* in a list and we are going to pop them out.

   (b) If the corner exceeds the value set by the threshold we will draw it.

   (c) We have the right corner, next step is to draw the rectangle. The parameters that we have to pass to *cv2.rectangle* are: the image, the upper left corner, the lower right corner, the color of the rectangle and the border thickness.

   (d) For the second image we proceed in the same way but taking into account that we place an image1.shape [1] to the right in the direction of the x.

```
while corner1:
        x=corner1.pop()
        if x[2]> (threshold * r_max1):
            cv2.rectangle(imagen_final,(x[0]-neighborhood,x[1]+neighborhood),
                (x[0]+neighborhood,x[1]-neighborhood),(0,0,255),1)
    while corner2:
        x=corner2.pop()
        if x[2]> (threshold * r_max2):
            cv2.rectangle(imagen_final,(x[0]-neighborhood+image1.shape[1],x[1]+neighborhood),
                (x[0]+neighborhood+image1.shape[1],x[1]-neighborhood),(0,255,0),1)
```

8. Visualize the both images, corners rectangles and features vectors.

```
cv2.imshow('CS_512_imagen6',imagen_final)
```

9. Each time we change the value of each trackbar, we have to update the value of R in each pixel and the features we find.

```python
def update():
    global name1
    global name2
    threshold = cv2.getTrackbarPos('Threshold','CS_512_imagen6')/100
    image1, corner1, r_max1 = findcorners(name1)
    image2, corner2, r_max2 = findcorners(name2)
    imagen_final = features(name1, name2)
    while corner1:
        x=corner1.pop()
        if x[2]> (threshold * r_max1):
            cv2.rectangle(imagen_final,(x[0]-1,x[1]+1),
                (x[0]+1,x[1]-1),(0,0,255),1)
    while corner2:
        x=corner2.pop()
        if x[2]> (threshold * r_max2):
            cv2.rectangle(imagen_final,(x[0]-1+image1.shape[1],x[1]+1),
                (x[0]+1+image1.shape[1],x[1]-1),(0,255,0),1)
    while imagen_final.shape[0] > 1200 or imagen_final.shape[1] > 1200:
        imagen_final = cv2.resize(imagen_final,(int(imagen_final.shape[1]/2),
            int(imagen_final.shape[0]/2)))
    cv2.imshow('CS_512_imagen6',imagen_final)
```

3. Implementation details

- **Problems found during the implementation of the program:**

Problem 1 : The algorithm to detect corners is not quite good. Comparing the result of the image shown with that shown by textit .cornerHarris we obtained different results.

Problem 2 : At the time of calculating the features of each image I have not used only the corners obtained in my algorithm and I have visualized all the features obtained for a certain distance that fluctuates with the threshold trackbar.

Problem 3 : The code is too slow. Correct would have been to realize the algorithm that detects the corners in Cython.

- **Instructions for using the program:**

Step 1 : Move the Gaussian trackbar (1-255) to smooth the images and get a different result from the corners and features.

Step 2 : Move the neighbor trackbar to increase or decrease the large (1-10) which is the window in the Harris algorithm.

Step 3 : Move the threshold trackbar to increase or decrease the percentage (1-100) of corners and features that you want displayed.

Step 4 : Moves the trackbar the weight to increase or decrease between (0.04-0.06) the parameter that we pass to the algorithm of Harris.

Step 5 : Press h for information.

Step 6 : To finish press the Esc button.

4. Results and discussion

- **Discuss of the correctness of the implementation:**

  1. The program is quite slow and could be solved by putting the algorithm of Harris in a cython program that solves much faster the value R that each pixel has.

  2. The features should be done with the selected corners and not depending on the distances they have and implementing a threshold.

- **Result obtained:**

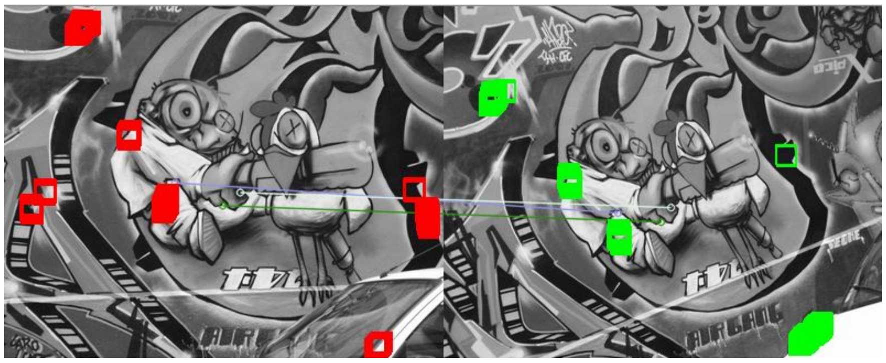

Figure 1: Neighbours 10 Threshold 90 Weight 0.04 Gaussian 1.

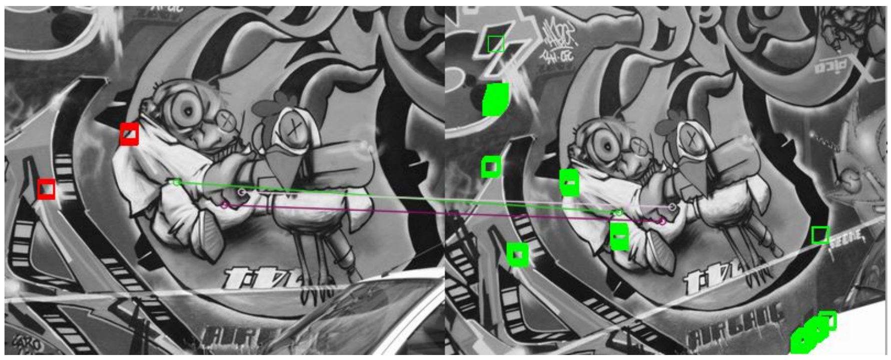

Figure 2: Neighbours 7 Threshold 90 weight 0.04 Gaussian 1 .

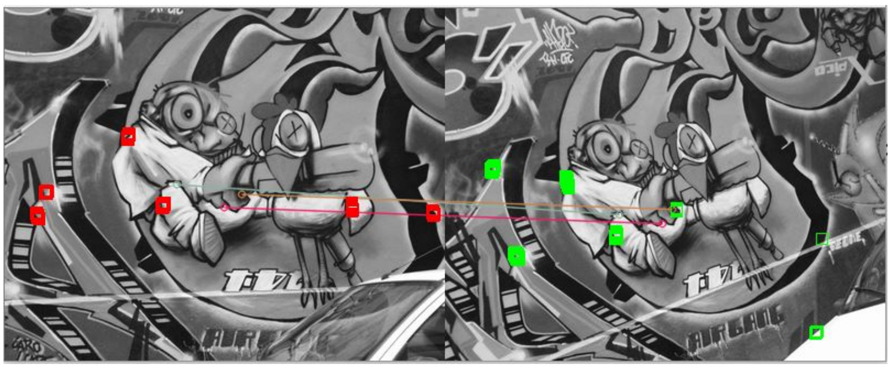
Figure 3: Neighbours 5 Threshold 90 weight 0.04 Gaussian 1 .

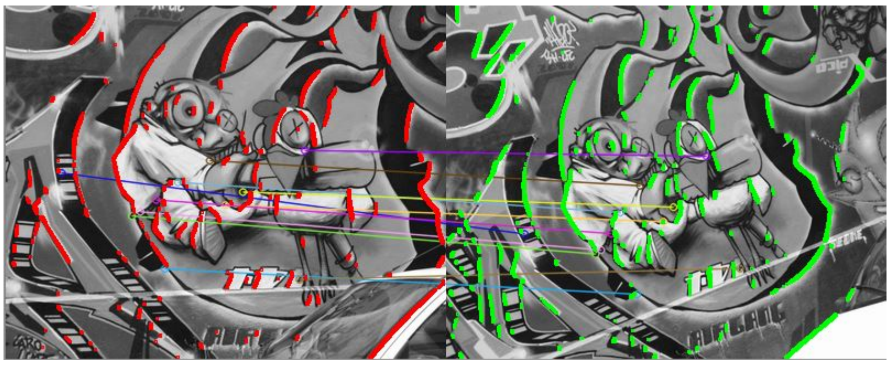
Figure 4: Neighbours 1 Threshold 75 weight 0.04 Gaussian 1.

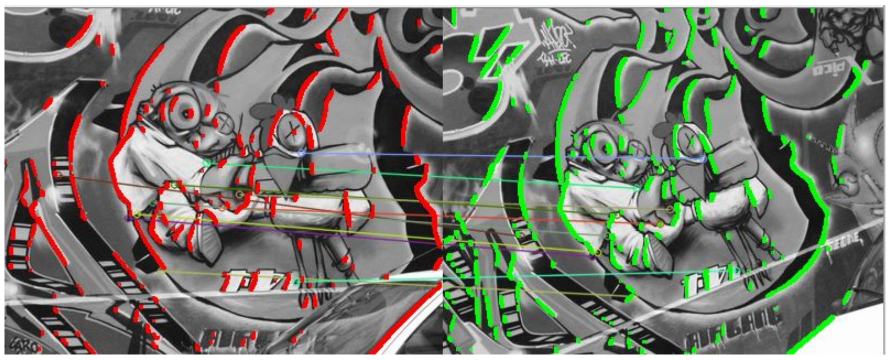
Figure 5: Neighbours 1 Threshold 75 weight 0.052 Gaussian 1 .

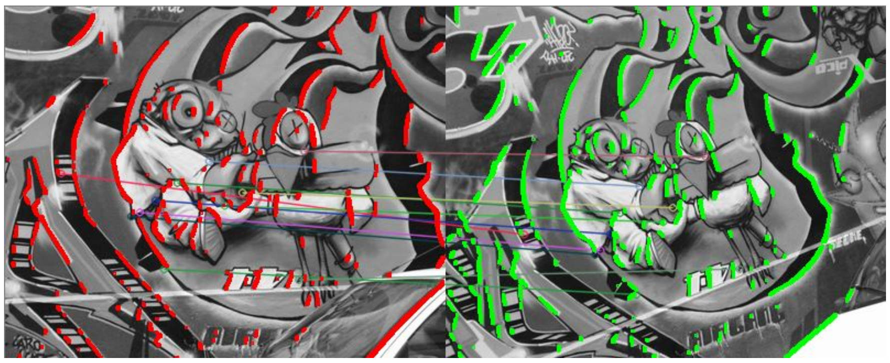
Figure 6: Neighbours 1 Threshold 75 weight 0.06 Gaussian 1.

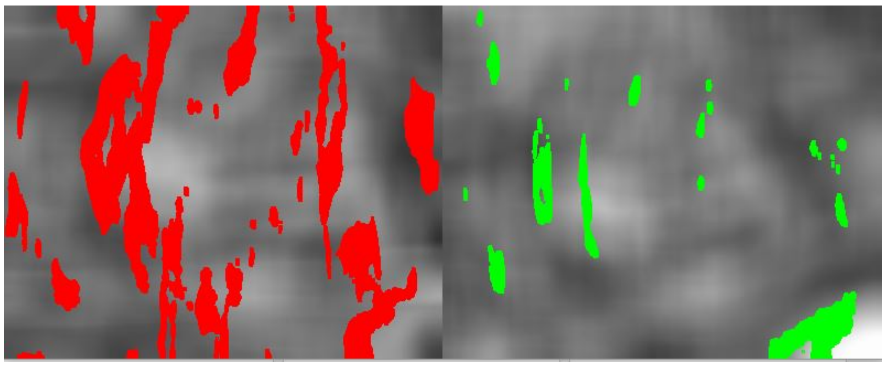
Figure 7: Neighbours 1 Threshold 70 weight 0.04 Gaussian 52

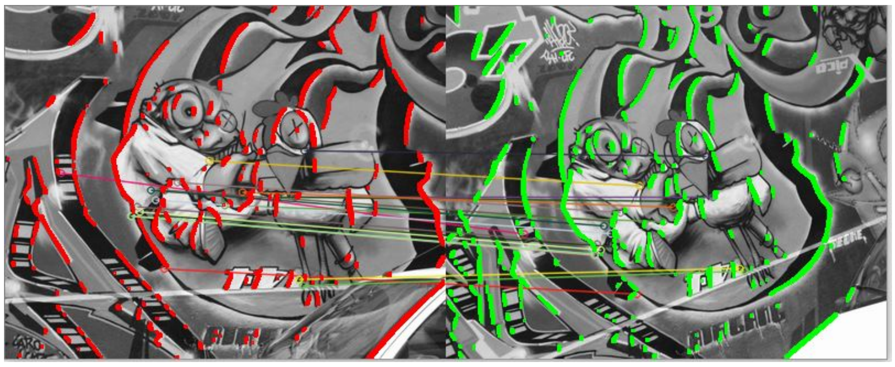
Figure 8: Neighbours 1 Threshold 70 weight 0.04 Gaussian 52.

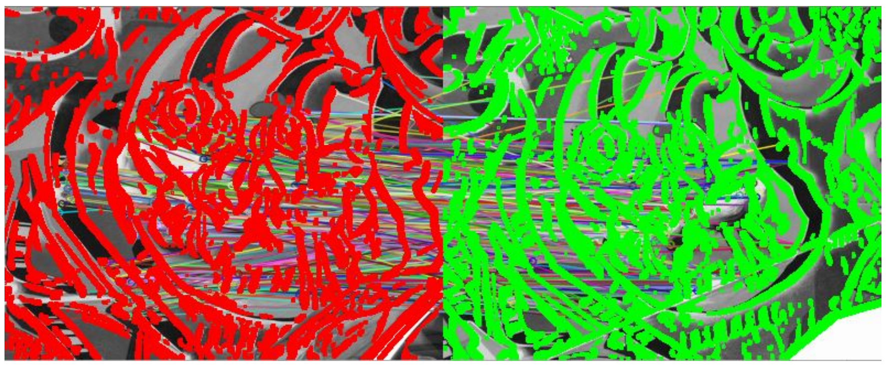
Figure 9: Neighbours 1 Threshold 70 weight 0.04 Gaussian 30.

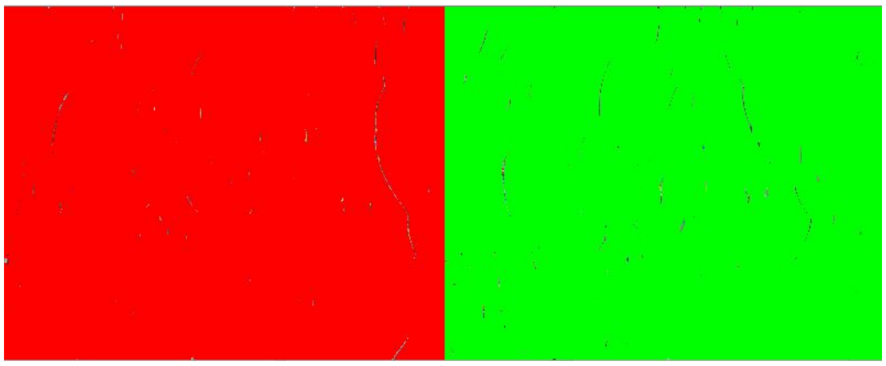
Figure 10: Neighbours 1 Threshold 70 weight 0.04 Gaussian 10.