A02 Fitting and Alignment

Name: K.N.A.L.Fernando

Index no: 200164H

Git-Hub: https://github.com/leonfdo/Image_Processing_Assig2

Blob detection

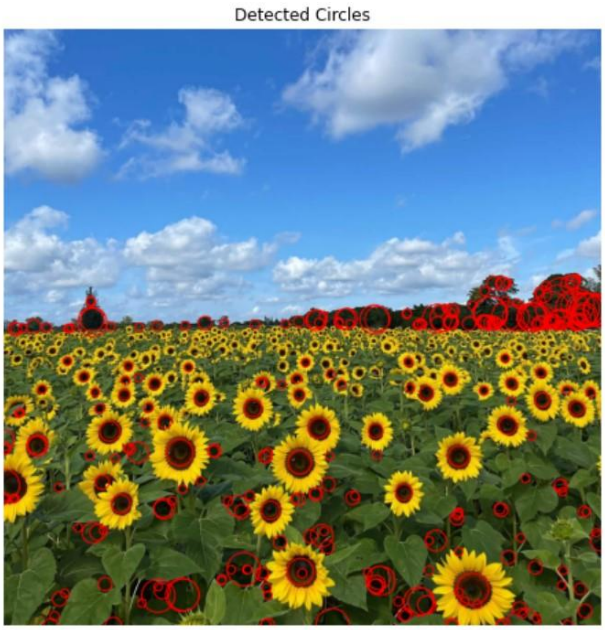
- The necessary libraries were imported, and the picture was then read and shown. Next, various Laplacian gaussian kernels of varying sigma and length were formed. The image was then convolved with each Laplacian gaussian kernel, and scale space was produced.

```
sig=4
pic=[]
for i in range(1,5):
    sigma=i*sig
    h_w =3*sigma
    X,Y=np.meshgrid(np.arange(-h_w,h_w+1,1),np.arange(-h_w,h_w+1,1))
    log = 1/(2*np.pi*sigma**2)*(X**2/(sigma**2) + Y**2/(sigma**2) - 2)*np.exp(-(X**2 + Y**2)/(2*sigma**2))
    new_img=cv.filter2D(img,-1,log)
    pic.append(new_img)
arr=np.array(pic)
```

- Using a 3*3 kernel, the maximum value of each sigma space array is then computed. The maximum value among them is then chosen, and the relevant coordinate and sigma value are calculated.

```
 def max_cordinates(arr,img):
    h,w=img.shape
    co_ord=[]
    for i in range(1,h):
        for j in range(1,w):
            slice=arr[:,i-1:i+2,j-1:j+2]
            max_com=np.amax(slice)
            if max_com>=0.01:
                z,x,y=np.unravel_index(slice.argmax(),slice.shape)
                co_ord.append((i+x-1,j+y-1,z*sig))
    return co_ord
t=max_cordinates(arr,img)
```

```
Parameters of the Largest Circle:
Center: (359, 172)
Radius: 35
Range of Sigma Values Used: 5.0 to 50.90000000000002
```

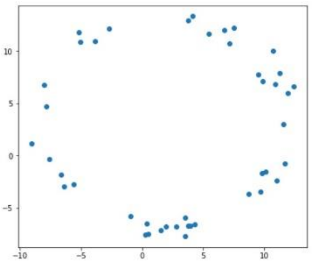


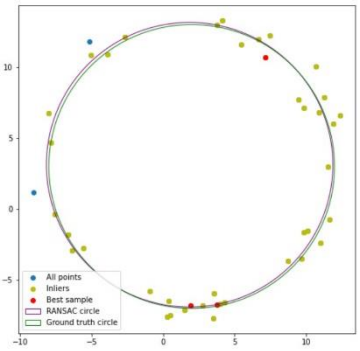
Ransac

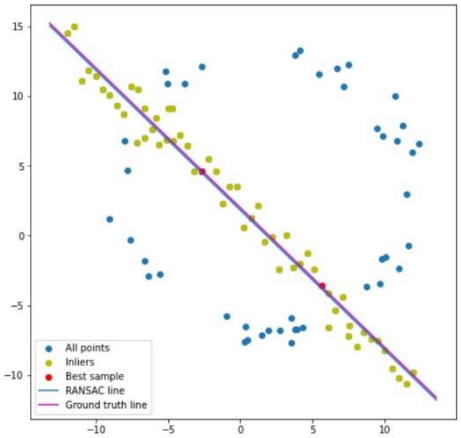
1)Estimating the line

- A line was drawn between two randomly chosen points, and the distance between the line and the other points was determined. The points that had a distance below the threshold value 2 was counted.
- This process was done 200 times and the line which has the maximum number of points below threshold is selected as the estimated line.

2) Estimating the circle



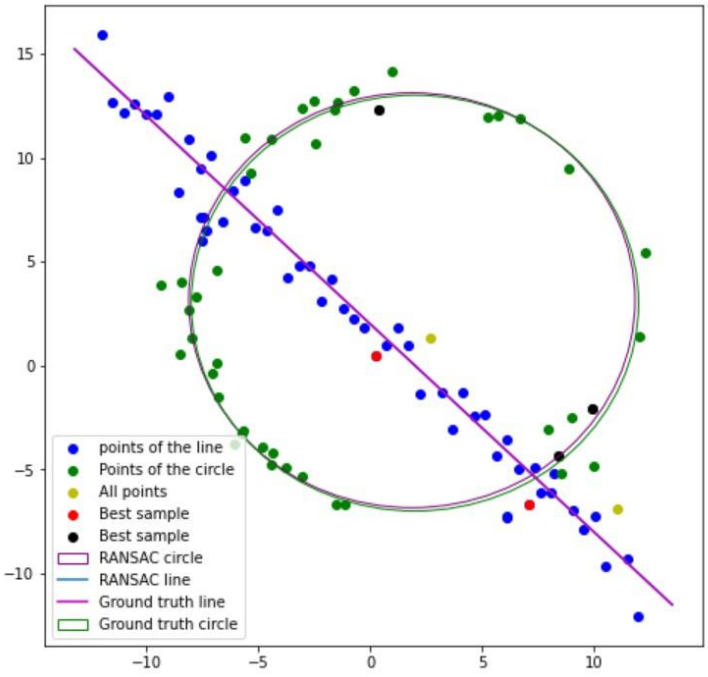




First the points related to the line was removed from the Points and then ransack model is used to estimate the circle

- To estimate the circle first we randomly select three points and draw a circle which pass through that three points then we calculate the points which has distance below the threshold value and count them
- This process is continuously for 200 times and the circle which has maximum number of points below threshold is selected as the circle.

3) The diagram is drawn



4)

We are unable to do this because, if we fit the circle first, the circle will then attempt to fit to the line's points. Since a circle can fit to a line by having a radius of infinity, the Ransac will incorrectly identify this circle as the best circle.

Homography

- First the two images were selected which is required to perform hormography
- Next, the point function is called using the cv.setMouseCallback function, and we must give the points in the first image to which we must map the second image.
- The required homography matrix can be found using the cv.findHomography function in order to transform the second image to the first image plane.
- Then the two images are warped using cv.warpPerspective fuction

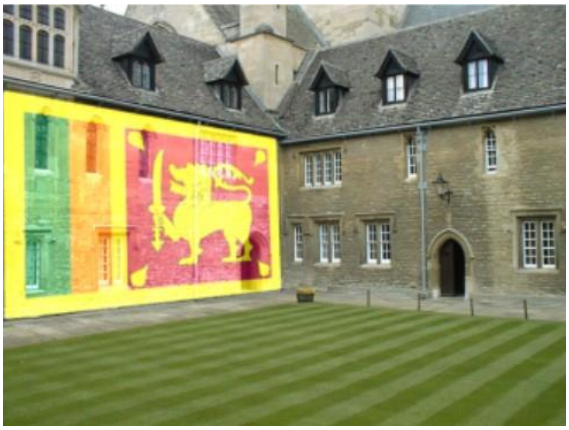


Image 1

Image 2

Final Image

Mouse Functionality

```
def points(event, x_cord, y_cord, flags, param):
    global pos, count

    if event == cv.EVENT_LBUTTONUP:
        cv.circle(img1, (x_cord, y_cord), 2, (255, 0, 0), -1)
        pos.append([x_cord, y_cord])
        if (count != 3):
            pos2.append([x_cord, y_cord])
        elif (count == 3):
            pos2.insert(2, [x_cord, y_cord])
        count += 1

img1 = cv.imread("./wall.jpeg")
img2 = cv.imread("./download.png")

cv.namedWindow("window")
cv.setMouseCallback('window', points)
```

Detecting Hormography matrix and Translation

```
point1 = np.float32([[0, 0], [w1, 0], [0, h1], [w1, h1]])
point2 = np.float32(pos)
h, mask = cv.findHomography(point1, point2, cv.RANSAC, 5.0)
height, width, channels = img1.shape
img_reg = cv.warpPerspective(img2, h, (width, height))
alpha_mask = np.ones(img_reg.shape[:2], dtype=np.float32)
alpha = 0.7
alpha_mask[alpha_mask == 1] = alpha
blended = cv.addWeighted(img1, 1.0, img_reg, alpha, 0)
```

Photo stitching

Compute and match SIFT features between the two images

- First, we need to determine the interesting points in the two images using the SIFT algorithm. And uniquely assign signatures to each interesting point.
- Then map the interesting points in each image using these signatures
- cv.SIFT_create and detectAndCompute functions are used to detect co-ordinates and the signatures corresponding to each image. Then cv.BFMatcher brute force matching is used to match each points.

Computed Hormography vs Real Hormography

Ransac algorithm is utilized here. What's going on is The hormography transformation matrix is constructed using the similar points that were matched together, and Euclidian distances are computed to assess how well the points match. After 100 iterations of this method, the best performing hormography mat

```
Homography calculated
 [[ 5.74131342e-01  4.52331198e-02  2.25448543e+02]
 [ 1.90357236e-01  1.11937193e+00 -1.26196868e+01]
 [ 3.93285701e-04 -7.98630780e-05  1.00000000e+00]]
--------------------------------------------------------------
Original Homography
    6.2544644e-01   5.7759174e-02   2.2201217e+02
   2.2240536e-01   1.1652147e+00  -2.5605611e+01
   4.9212545e-04  -3.6542424e-05   1.0000000e+00
```

this image shows the calculated hormography matrix and the reall hormography matrix

Stitch img1.ppm onto img5.ppm.