Hough Transformation For Circle Detection Of Different Radii

Noor Ul Ain (BCSF20M503)

Rida Fatima (BCSF20M509)

Zunaira Sajjad (BCSF20M514)

Rida Shabbir (BCSF20M541)

27-Feb-2023

Introduction:

We are going to perform circle detection using hough transformation. We are given an image of a Moon and we have to locate the circle in that image which is the Moon’s boundry itself. The code implementation we did is efficient enough to detect circles in any image. The coming sections include our code pictures and the explanation of how we did our project step by step and at last we showed the graphical results of our implementation by using python’s turtle module.

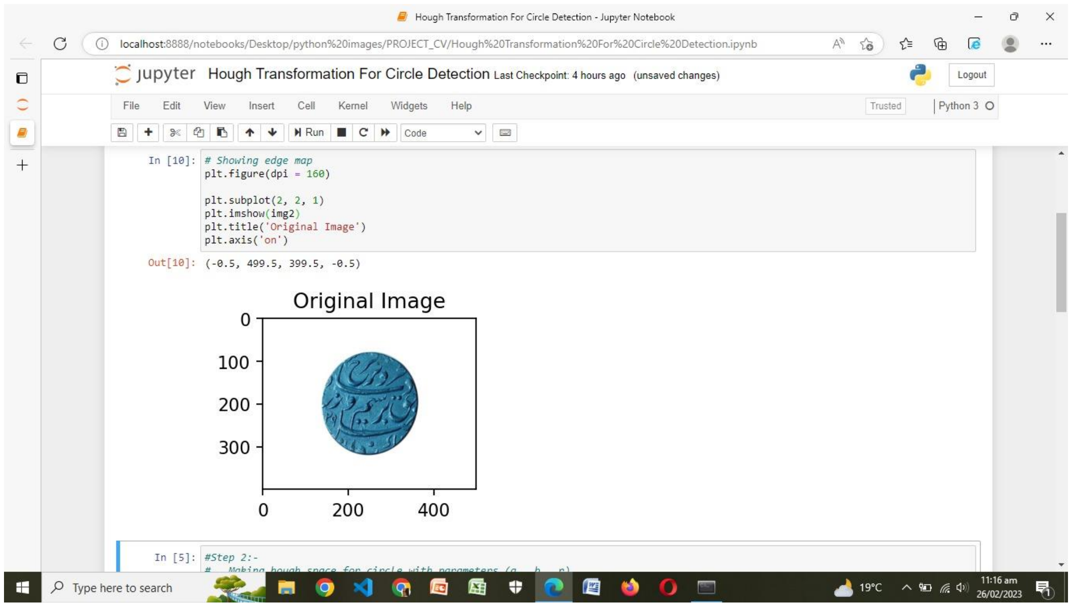
Literature Review:

We didn’t use any Books or references, our whole implementation is based on the pseudocode that was discussed in class, but we did use some built in functions from python libraries Numpy, Matplotlib, and turtle.

Implementation Details:

The implementation consist of following steps:

1. First we read our image


Hough Transformation For Circle Detection - Jupyter Notebook
localhost:8888/notebooks/Desktop/python%20images/PROJECT_CV/Hough%20Transformation%20For%20Circle%20Detection.ipynb
jupyter Hough Transformation For Circle Detection Last Checkpoint: 4 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Original Image
0
100
200
300
0
200
400
Type here to search
19°C
11:16 am
26/02/2023

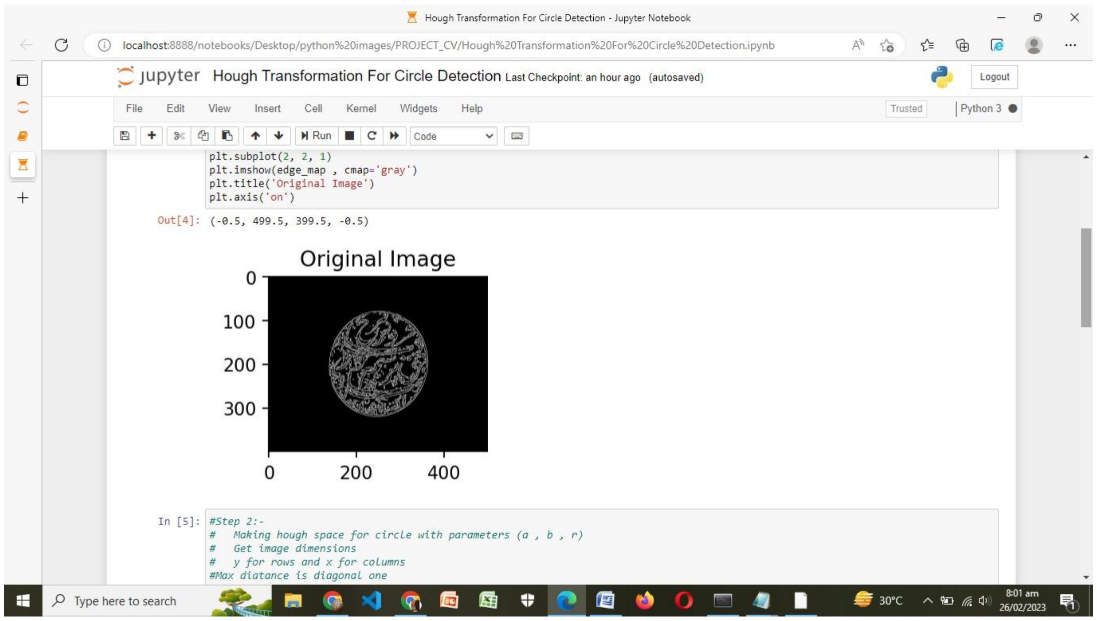
```
In [10]: # Showing edge map
         plt.figure(dpi = 160)

         plt.subplot(2, 2, 1)
         plt.imshow(img2)
         plt.title('Original Image')
         plt.axis('on')

Out[10]: (-0.5, 499.5, 399.5, -0.5)
```

```
In [5]: #Step 2:-
```

2. Then we obtained the edge map of our image to remove the constant backgroud and to get the edges we want to connect to make a circle as we know that’s how hough transform works, it connect the disjoint edges and make a shape out of them
3. Now the picture is ready, the next challenge is to select the parameters of our Accumulator array to collect the voting for our desired circle
4. As we know the equation of circle is:

$$x^2 + y^2 = r^2$$

As we can see from the equation the main descriptors of a circle are: 1. it’s the coordinates of its center and 2. its radius. So we will select these as parameters for our Accumulator array


Hough Transformation For Circle Detection - Jupyter Notebook
localhost:8888/notebooks/Desktop/python%20images/PROJECT_CV/Hough%20Transformation%20For%20Circle%20Detection.ipynb
jupyter Hough Transformation For Circle Detection Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Original Image
0
100
200
300
0
200
400
Type here to search
30°C
8:01 am
26/02/2023

```
plt.subplot(2, 2, 1)
plt.imshow(edge_map , cmap='gray')
plt.title('Original Image')
plt.axis('on')
```

```
Out[4]: (-0.5, 499.5, 399.5, -0.5)
```

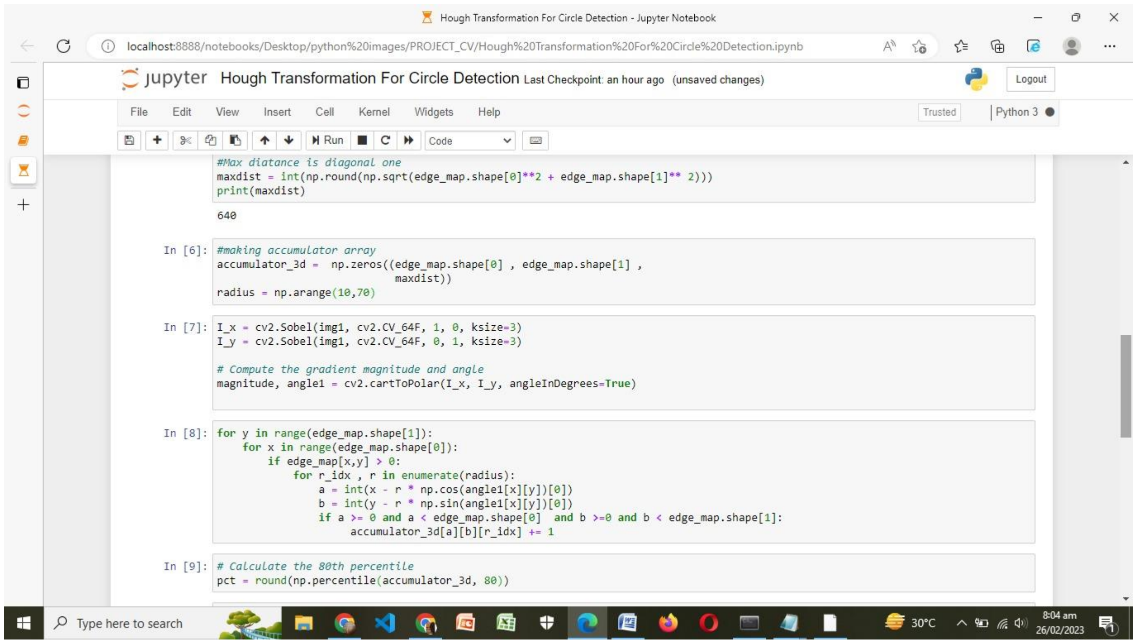
```
In [5]: #Step 2:-
        #   Making hough space for circle with parameters (a , b , r)
        #   Get image dimensions
        #   y for rows and x for columns
        #Max diatance is diagonal one
```

5. Firstly we will get dimentions of our circle to calculate the maximum distance which is the diagonal length of our image that would give us the radius of our circle
6. Before collecting votes we will initialize our array with zeros
7. And for this particular implementation we will take range of radius r from 0 to 70 as taking full range was too complex computationally but our code works for maximum distance/radius as well
8. As we know we have to calculate theta as well to find the dimention, the theta range is fixed (0-180), but that would be an extra loop and computationally heavy to add a loop for theta as well
   So what we will do is we will calculate gradient, and as we know gradient gives you the maximum directional change in a certain direction, so we will calculate gradient to improve effiiciency of our code
9. We will first calculate the gradients in x and y direction and then we will calculate the gradient magnitude and direction using the following formulas

$$\text{Magnitude} = (Ix^2 + Iy^2)^{½}$$

$$\text{Direction} = \tan^{-1} (Iy/Ix)$$

10. Now as We are clear with our dimentions and the parameters we want to use, We will move forward with collecting the voting for our circle
    We will use circle Dimentions in our outer loops and here we will add one more additional check
    We will apply an if condition to run the radius loop for only values greater than 0, As we know our edge map returned a gray scale image containing Disjoint edges so we will calculate the Radii for only those points that are edges or are actually helpful in detecting those circle. So we know we have marked our non edge values 0 so we will not consider it while calculating our results
    This will make sure that our code works efficiently
11. Next for each loop iteration we will calculate the a,b which corresponds to the center of the circle along with the radius, against each index we will have voting and the index with maximum voting will be helping us in drawing our final results
12. And last but not least we got our accumulator array but to make our results more fine we will perform one additional step here “Thresholding”. We calculated our threshold percentile and we found that to be 80 percent so we are now thresholding our results at 80



```
#Max diatance is diagonal one
maxdist = int(np.round(np.sqrt(edge_map.shape[0]**2 + edge_map.shape[1]** 2)))
print(maxdist)

640
```

```
In [6]: #making accumulator array
        accumulator_3d =  np.zeros((edge_map.shape[0] , edge_map.shape[1] ,
                                    maxdist))
        radius = np.arange(10,70)
```

```
In [7]: I_x = cv2.Sobel(img1, cv2.CV_64F, 1, 0, ksize=3)
        I_y = cv2.Sobel(img1, cv2.CV_64F, 0, 1, ksize=3)

        # Compute the gradient magnitude and angle
        magnitude, angle1 = cv2.cartToPolar(I_x, I_y, angleInDegrees=True)
```

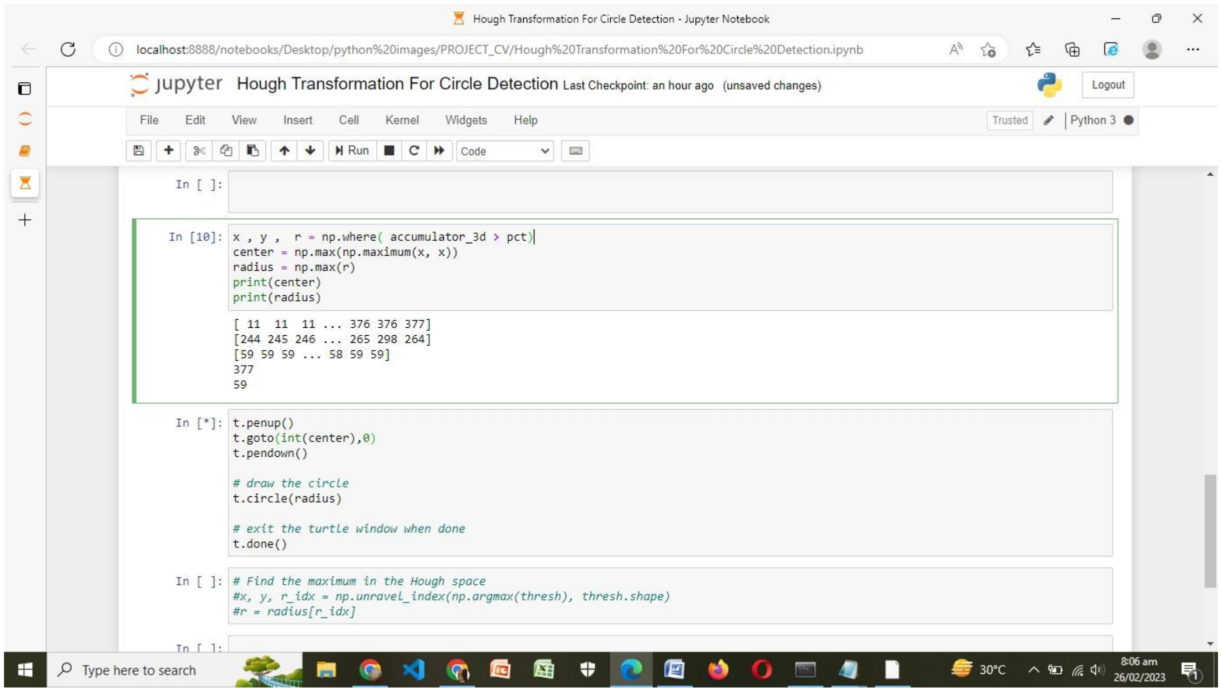
```
In [8]: for y in range(edge_map.shape[1]):
            for x in range(edge_map.shape[0]):
                if edge_map[x,y] > 0:
                    for r_idx , r in enumerate(radius):
                        a = int(x - r * np.cos(angle1[x][y])[0])
                        b = int(y - r * np.sin(angle1[x][y])[0])
                        if a >= 0 and a < edge_map.shape[0]  and b >=0 and b < edge_map.shape[1]:
                            accumulator_3d[a][b][r_idx] += 1
```

```
In [9]: # Calculate the 80th percentile
        pct = round(np.percentile(accumulator_3d, 80))
```

Results:

13. Now as we can see our accumulator array is all set, we will get the maximum center and radius from that array

w



```
In [ ]:
```

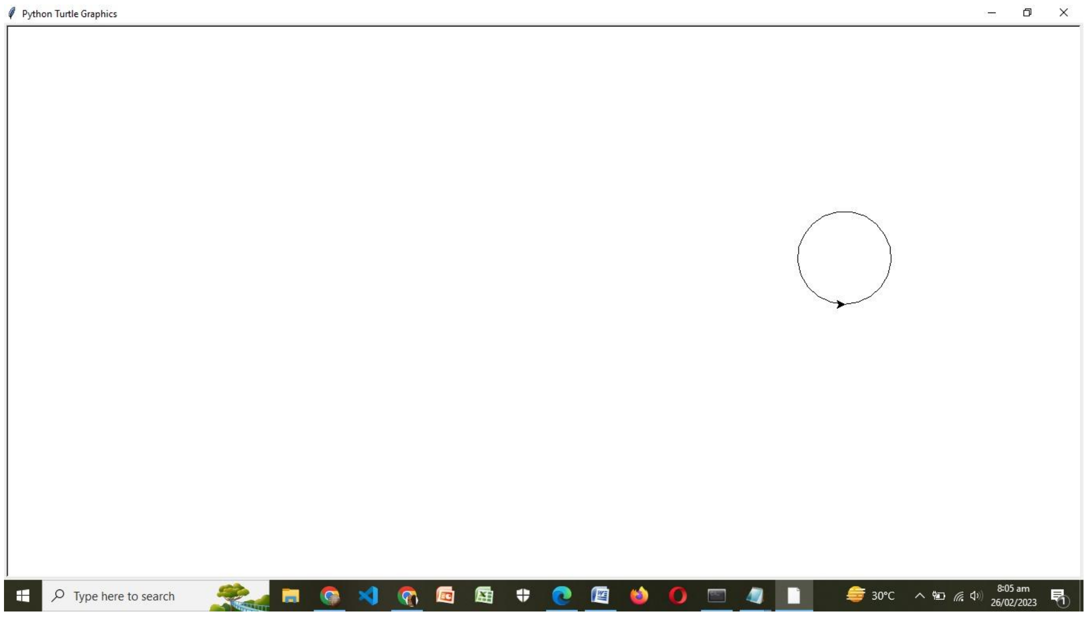
```
In [10]: x , y ,  r = np.where( accumulator_3d > pct)
         center = np.max(np.maximum(x, x))
         radius = np.max(r)
         print(center)
         print(radius)

         [ 11  11  11 ... 376 376 377]
         [244 245 246 ... 265 298 264]
         [59 59 59 ... 58 59 59]
         377
         59
```

```
In [*]: t.penup()
        t.goto(int(center),0)
        t.pendown()

        # draw the circle
        t.circle(radius)

        # exit the turtle window when done
        t.done()
```

```
In [ ]: # Find the maximum in the Hough space
        #x, y, r_idx = np.unravel_index(np.argmax(thresh), thresh.shape)
        #r = radius[r_idx]
```

14. Now final step we will use the turtle module to draw our circle, we will provide it with our calculated values of center and radius and it will do it’s magic and we will get our results

    Here is the Final Look of output from turtle



Conclusion:

We detected a circle by using it’s parameters as the dimentions of our accumulator array, and what’s accumulator array, it is simply collectiong the voting for our circle, and when enough points vote for the same circle and radius we know that it can be our potentional radius and center of circle and then we threshold our results to get the best center and radius and discard the weak ones.

< --- >