Alberto Fung

ID# 1163571

HW07

COSC6373

SP2019

**HW07 Hough Transform**

**Task 1: Using Built-in OpenCV Hough Transform**

The using the built-in openCV function of Hough Transform, detect circles in a given image.

**Approach**

The approach for detecting circles using the built-in openCV Hough Transform is as follows;

1) Read an image

2) Convert the image into grayscale

3) Apply an edge detection (the HoughTransform’s buildt-in edge detection was used in this implementation)

4) Apply Hough Transformation for circles with a threshold

5) For each circle in the input image, superimpose a circle and the center on the image output

**Input: Test Images**

The input images were taken using a smartphone, some coins, and a white sheet of paper as a background.

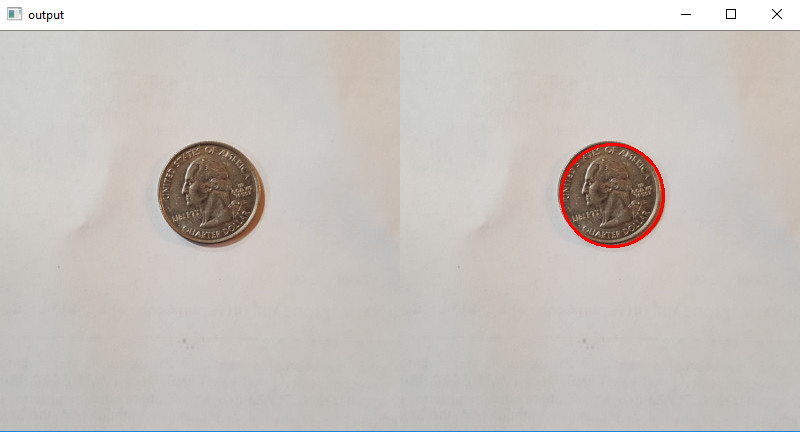
The 1st image was a single coin taken on a white background. The 2nd image was several different size coins taken with a white background.





**Output**

The following are outputs based on the input images.

****

****

**Code**

imgfile = 'images/test1\_400.jpg'

image = cv2.imread(imgfile)

output = image.copy()

imgBlur = cv2.medianBlur(image, 5)

gray = cv2.cvtColor(imgBlur,cv2.COLOR\_BGR2GRAY)

imagegray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# implementation based on OpenCV HoughCircles algorithm

circles = cv2.HoughCircles(gray, cv2.HOUGH\_GRADIENT,1,50, param1 = 100, param2 = 30)

if circles is not None:

circles = np.round(circles[0, :]).astype("int")

for (x, y, r) in circles:

cv2.circle(output, (x,y), r, (0, 0,255), 2)

cv2.imshow("output", np.hstack([image, output]))

cv2.waitKey()

cv2.destroyAllWindows()

**Analysis**

The built-in function does a good job locating the circles in a given image. One of the disadvantages of using the built-in function is the required turning of the parameters in the cv2.HoughCircles function. You have to tune the parameters for each images otherwise it would detect either too many circles due to noise or detect no circles at all.

**Task 2: Implement Your Own Hough Transform to detect a circle**

Given an image, locate all circles using your own customized implementation of the Hough Transform

**Approach**

The approach for detecting circles using the built-in openCV Hough Transform is as follows;

Using the equation of a circle:

**r^2 = (x – x0)^2 + (y-y0)^2**

1) Read an image

2) Convert the image into grayscale

3) Apply an edge detection

4) Create an accumulator array based on maximum image height, width and r

5) For each edge point, iterate over all a values and b values and calculate the r at every point

6) Increment the accumulator array at the locations for each point

7) Find all points in the accumulator array that is above a certain voting threshold

8) Given the points, superimpose a circle on top of the input image

**Input: Test Images**

The input images were taken using a smartphone, some coins, and a white sheet of paper as a background.

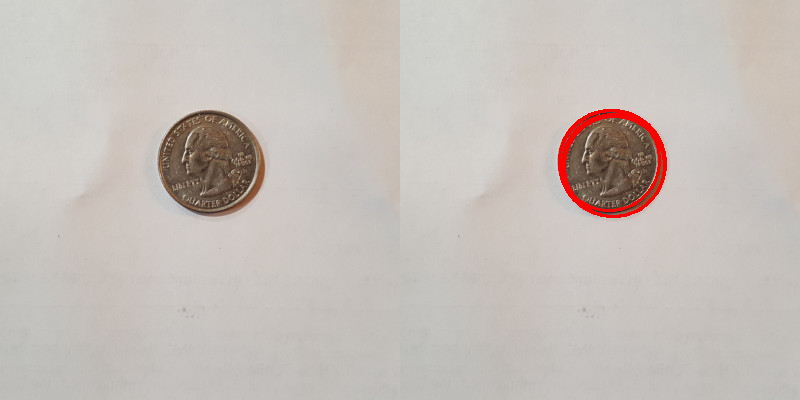
The 1st image was a single coin taken on a white background. The 2nd image was several different size coins taken with a white background.





**Output**

The following are outputs based on the input images.

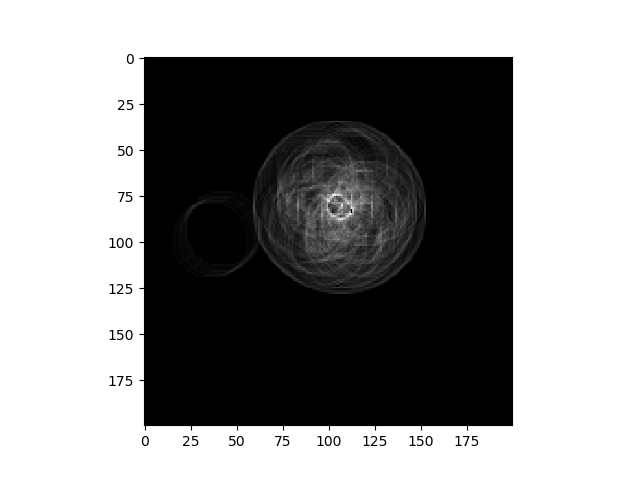
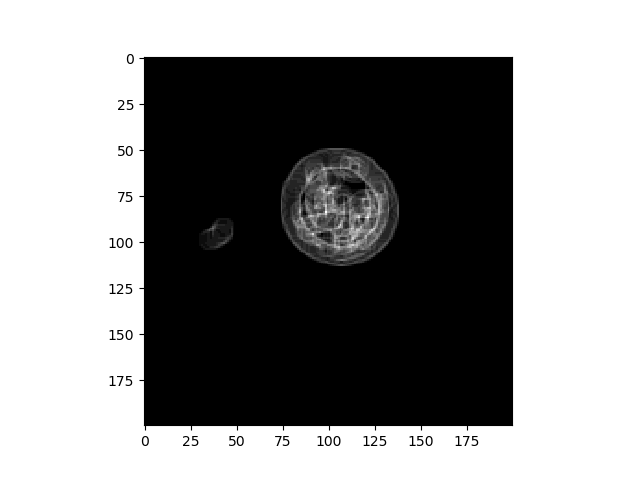
****

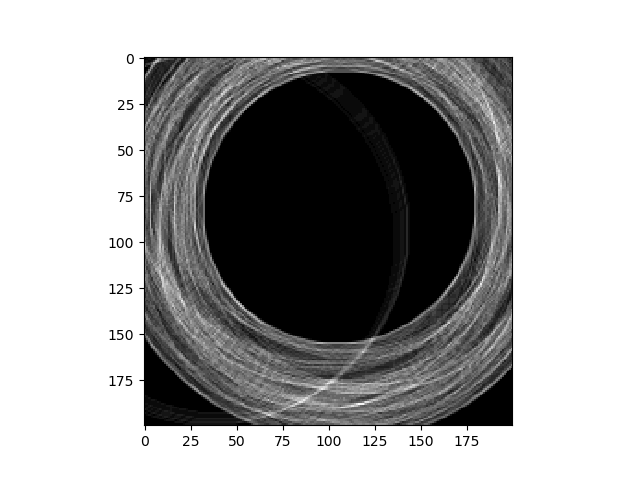
****

**Accumulator Array Visualization**

The following is a visualization of the accumulator array for both of the input images. The accumulator array for these visualization were created were limited to 200 pixels as being the largest for either dimension of the image.

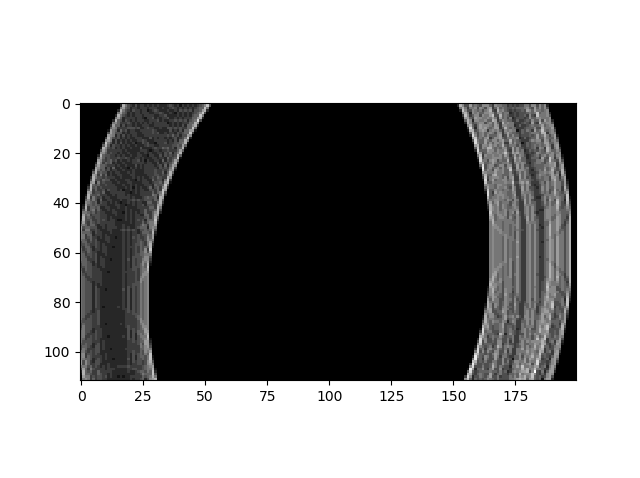
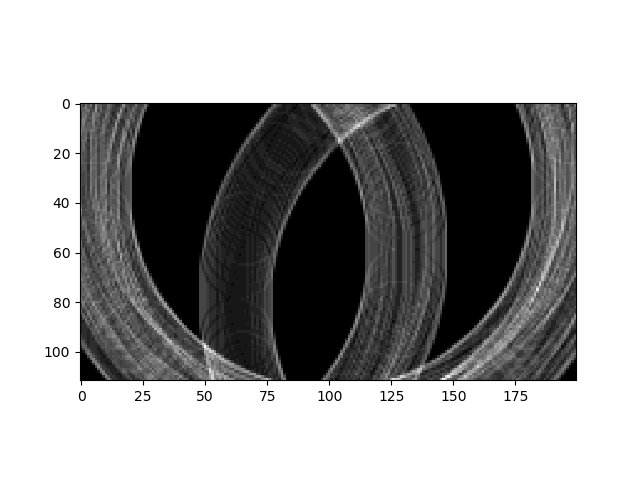
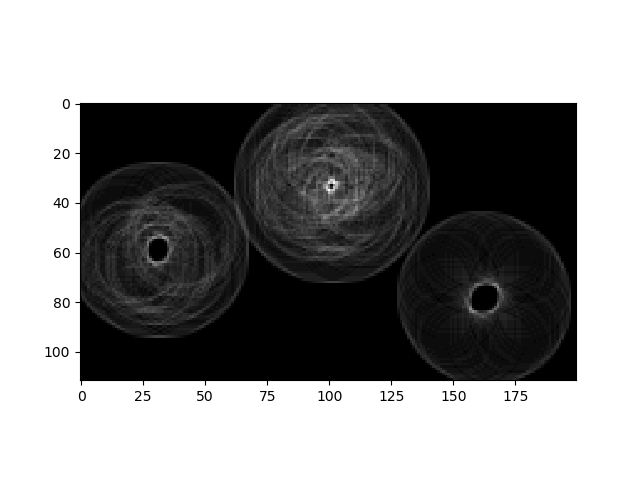
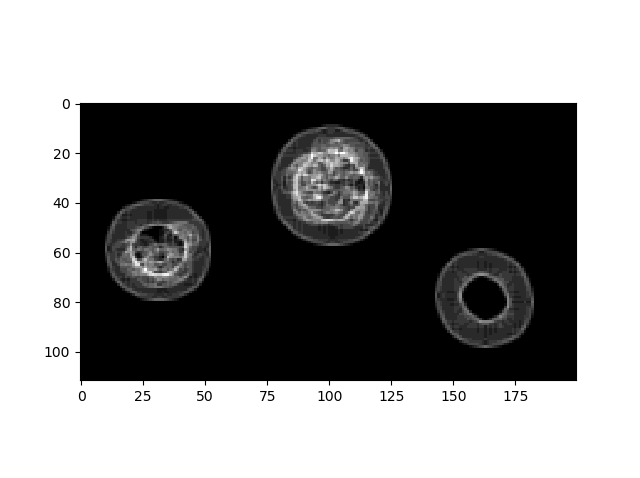
**Accumulator Array Slices (Single Coin)**



****

**The accumulator array slices shown above is for the single coin input showing slices at r = 5, r = 20 and r = 100 respectively.**

**Accumulator Array Slices (Multiple Coin)**

****

**The accumulator array slices shown above is for the multiple coin input showing slices at r = 5, r = 20 and r = 100, and r = 150 respectively.**

**Assumptions**

The following assumptions were made for the custom Hough Transform algorithm:

1) Images were less than 400x400 in size. While the algorithm does work with images larger than 400x400, the computation takes too long and the results shown here uses 400x400 images

2) Coins are separated by at least 1 pixel width (no coins are superimposed)

**Code**

# implementation using custom algorithm

import cv2

import sys

import numpy as np

import math

from matplotlib import pyplot as plt

def findCircles(*accumulator*, *threshold*, *minRadius*):

def findMax (*array*):

max = np.unravel\_index(np.argmax(array,*axis*=None), array.shape)

return (max)

circle = []

x = y = r = 0

while accumulator[(findMax(accumulator))] > threshold:

(x,y,r) = findMax(accumulator)

circle.append((x,y,r))

accumulator[(x,y,r)] = 0

return circle

imgfile = 'images/test1\_400.jpg'

image = cv2.imread(imgfile)

output = image.copy()

imgBlur = cv2.medianBlur(image, 5)

gray = cv2.cvtColor(imgBlur,cv2.COLOR\_BGR2GRAY)

# print(image)

imagegray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray,30,50)

[ximg,yimg] = np.nonzero(edges)

edgePts = np.vstack([ximg,yimg]).T

# print(edgePts)

xMax = gray.shape[0]

yMax = gray.shape[1]

rMax = *int*(math.sqrt((xMax)\*\*2 + (yMax)\*\*2))

threshold = *int*(.9\*57)

# print(rMax)

acc = np.zeros((xMax,yMax,rMax), *dtype* = *int*)

for i in edgePts:

for a in range(xMax):

for b in range(yMax):

r = *int*(math.sqrt((i[0] - a)\*\*2 + (i[1] - b)\*\*2))

acc[a,b,r] += 1

# circle = []

results = findCircles(acc,threshold, 5)

if results is not None:

# circles = np.round(circles[0, :]).astype("int")

for (x, y, r) in results:

cv2.circle(output, (y,x), r, (0, 0,255), 2)

plt.imshow(acc[:,:,5], cmap="gray")

plt.show()

cv2.imshow("output", np.hstack([image, output]))

cv2.waitKey()

cv2.destroyAllWindows()

**Analysis**

While this is not the best implementation of the customized Hough Transform to find a circle, the implementation of using polar coordinates as well as gradient direction to limit the number of voting in the accumulator array did not function as intended.

The customized hough transform takes longer than the built-in function but it was easier to control the parameters. The only parameters required was a threshold of the accumulator to determine the location of the circles. Additionally, the built-in function does not include an ability to visualize the accumulator array.