

---

# BBM418 Introduction to Computer Vision Lab.

## Assignment 1

### Circle by Using Hough Transform

---

Ayşe Balci  
21726985  
Department of Computer Engineering  
Hacettepe University  
Ankara, Turkey  
b21726985@cs.hacettepe.edu.tr

#### Overview

The Hough transform is a feature extraction technique used in computer vision and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. In this assignment, I am going to start using any edge detection method to obtain edge points from an image, and then I will implement Hough Transform to detect circles.

#### 1 Part 1 - Edge Detection

Edge detection is an image-processing technique, which is used to identify the boundaries (edges) of objects, or regions within an image. I started to assignment by using Canny edge detector that is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.

I apply Gaussian Blur, that is mainly used to blur the image and remove sharp noise in the images, to the input image. This will be the input to Canny edge detector. Then I applied Canny edge detection from OpenCV library. Canny edge detector has two values of threshold, on optimization I found that 75 and 150 as ideal thresholds.

- Read the input image:

```
inputImage = cv2.imread(filename, 1)
```

- Gaussian Blurring of Gray Image to reduce noise:

```
gauss = cv2.GaussianBlur(inputImage, (3, 3), 0)
```

- Using OpenCV Canny Edge detector to detect edges:

```
cannyImage = cv2.Canny(gauss, 75, 150)
```

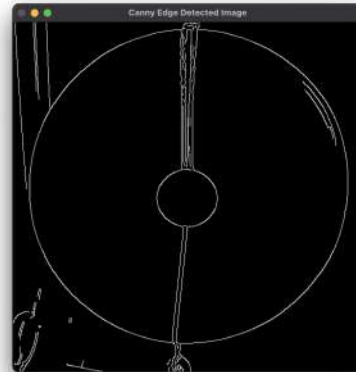


Figure 1: Examples of Canny Edge Detection

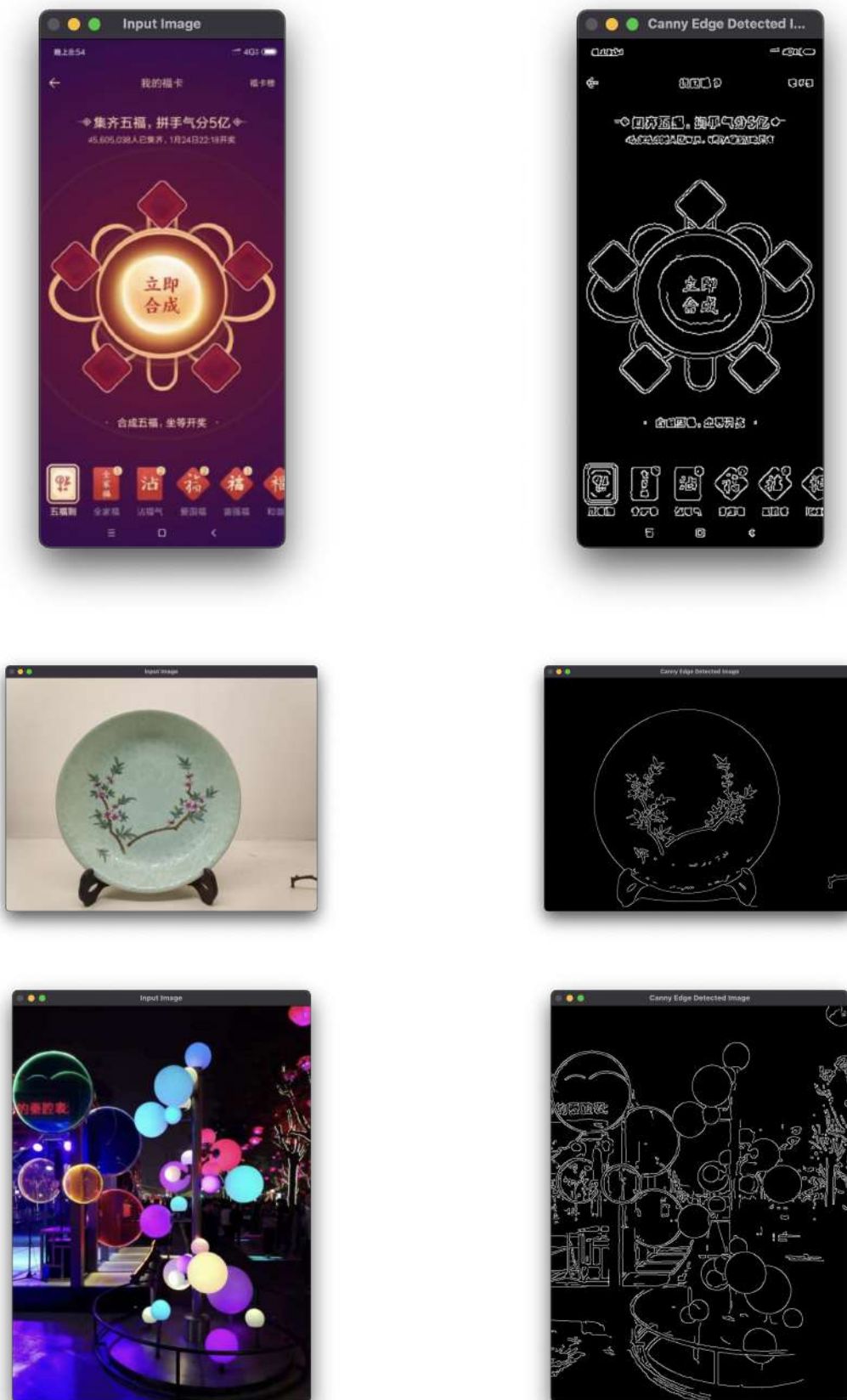


Figure 2: Examples of Canny Edge Detection

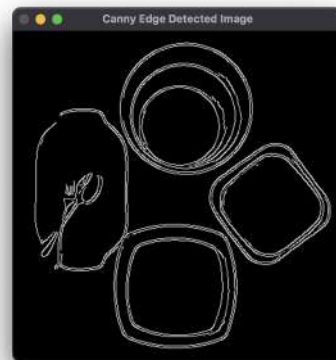
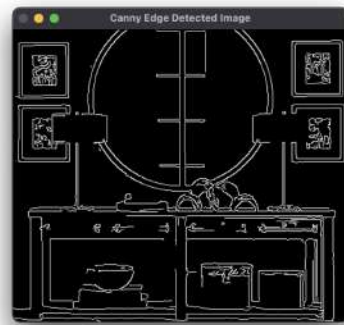
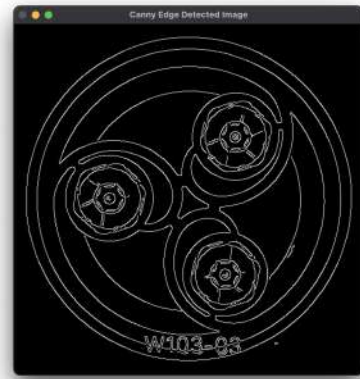
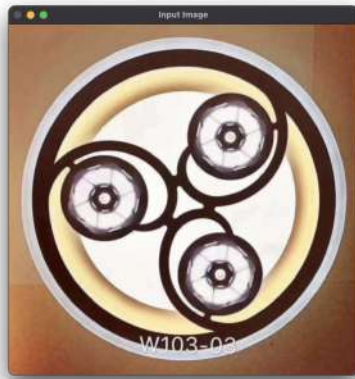


Figure 3: Examples of Canny Edge Detection



Figure 4: Examples of Canny Edge Detection

## 2 Part 2 - Hough Transform

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the classical Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, etc. The Hough transform uses a voting procedure that is carried out using accumulation arrays. Hough transform can be implemented to detect many shapes in an image. In this assignment, I implemented Hough transform to detect circles in an image.

Some values should be given by the user at the beginning of the program, otherwise default values will be used. These are:

- Minimum radiues
- Maximum radius
- Threshold

The project traverse the image multiple times for all values of radius between minimum and maximum radius value of circles and accumulate the hits at every pixel. The position where there is most number of hits, must be the center of the circle of that particular radius.

After the Gaussian Blur and Canny edge detection in the first part, there is an image just contains black and white pixels.

### 2.1 Implementation of Algorithm

I created accumulator list and filled it with radius,x and, y points by polar coordinate equations mentioned below. The list travel between minimum radius and maximum radius, and theta is increase 10 between 0 and 360 degrees.

The formulas to find the points circle centered M(a, b).

$$a = x - r * \cos(\theta)$$

$$b = y - r * \sin(\theta)$$

```
def CreateAccumulatorList(image, height, width, radius_min, radius_max):
    accumulator = np.zeros((radius_max, height, width))
    for r in range(radius_min, radius_max):
        for m in range(height):
            for n in range(width):
                if image[m][n] == 255:
                    for theta in range(0, 360, 10):
                        x0 = int(m - r * math.cos(theta * math.pi / 180))
                        y0 = int(n - r * math.sin(theta * math.pi / 180))
                        if x0 > 0 and x0 < height and y0 > 0 and y0 < width:
                            accumulator[r][x0][y0] += 1
    return accumulator
```

After the Accumulator array was created I normalized it to obtain a standard threshold value between 0 to 1. This threshold specifies the occlusion ration and in turn the sensitiveness of circle detection.

```
for r in range(radius_min, radius_max):
    accumulator[r] = accumulator[r] / max_value # Normalize values
```

Then, since not to draw too many circles at one particular pixel value, slide with 30 in image height, and width values, in every time I consider accumulator list 30x30 part. When accumulator list elements exceed threshold, I draw circle in original image.

```
def DetectHoughCircles(image, accumulator, height, width, radius_min, radius_max, threshold):

    slide_pixel_amount = 30
    max_values = np.zeros((radius_max, slide_pixel_amount, slide_pixel_amount))
    max_value = np.amax(accumulator)

    final_circles = []
    for r in range(radius_min, radius_max):
        accumulator[r] = accumulator[r] / max_value # Normalize values

    for i in range(0, height - slide_pixel_amount, slide_pixel_amount):
        for j in range(0, width - slide_pixel_amount, slide_pixel_amount):
            max_values = accumulator[:, i:i + slide_pixel_amount, j:j + slide_pixel_amount]

            max_val = np.where(max_values == max_values.max())
            r, x, y = max_val[0][0], max_val[1][0], max_val[2][0]

            if max_values.max() > threshold:
                accumulator[:, i:i + slide_pixel_amount, j:j + slide_pixel_amount] = accumulator[
                    :, i:i + slide_pixel_amount, j:j + slide_pixel_amount]
                accumulator[r][x + i][
                    y + j]

            for r in range(radius_min, radius_max):
                for q in range(i, i + slide_pixel_amount):
                    for w in range(j, j + slide_pixel_amount):
```



```

        if accumulator[r][q][w] > 0.9:
            cv2.circle(image, (w, q), r, (0, 255, 0), 2)
            final_circles.append((float(w), float(q), float(r)))
            #print(w, q, r)

    return image, final_circles

```

## 2.2 Calculating Intersection over Union (IoU)

The algorithm draws more than one circle for every circles in original image. I used a method to eliminate these circles.

In above code I had created a list for final circles. For every circle in original image, I am looking at every circle draw in final. I calculate the distance with the known circle to each center in the final circle list. If the distance is less than 3, then I am looking at the radii. If the distance between the radii is also less than 3, I am considering this circle as a true draw circle.

I added all true draw circles to a list. Then I calculated the intersection areas of all the circles in this list for the radius, x, and y points, and by sending the radius, x, and y points of the original image to a function. And finally, in above formula I calculated the Intersection over Union

$$IoU = AreaofOverlap / AreaofUnion$$

```

for line in lines:
    x = float(line.strip().split(' ')[0])
    y = float(line.strip().split(' ')[1])
    r = float(line.strip().split(' ')[2])
    mainCircleArea = math.pi * r * r
    list = []
    for detected_x, detected_y, detected_r in final_circles:

        distanceBetweenCenters = math.sqrt(abs(x - detected_x) ** 2 + (abs(y - detected_y) ** 2))

        if (distanceBetweenCenters < 3 and abs(r - detected_r) < 3):
            findedCircleCount += 1
            list.append((detected_x, detected_y, detected_r))

    if len(list) > 0:
        print('for ', str(x), ' ', str(y), ' ', str(r), ' detected circles are:')
        for detected_x, detected_y, detected_r in list:
            intersection = areaOfIntersection(x, y, r, detected_x, detected_y, detected_r)
            detectedCircleArea = math.pi * detected_r * detected_r
            iou_line = intersection / (mainCircleArea + detectedCircleArea - intersection)
            iou += iou_line
            print(detected_x, ' ', detected_y, ' ', detected_r, ' detected line Intersection')
            count += 1
    else:
        print('for ', line, ' no circle detected')
        couldNotFindCircleCount += 1
        count += 1

```

I calculated are of intercession between two circles with below code.

```

def areaOfIntersection(x0, y0, r0, x1, y1, r1):
    distance = math.sqrt((x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0))

    if (distance > r1 + r0):

```

```

    return 0

elif (distance <= abs(r0 - r1) and r0 >= r1):
    # Return area of circle1
    return math.pi * r1 * r1

elif (distance <= abs(r0 - r1) and r0 < r1):
    # Return area of circle0
    return math.pi * r0 * r0

else:
    phi = (math.acos((r0 * r0 + (distance * distance) - r1 * r1) / (2 * r0 * distance))) * 2
    theta = (math.acos((r1 * r1 + (distance * distance) - r0 * r0) / (2 * r1 * distance))) * 2
    area1 = 0.5 * theta * r1 * r1 - 0.5 * r1 * r1 * math.sin(theta)
    area2 = 0.5 * phi * r0 * r0 - 0.5 * r0 * r0 * math.sin(phi)

# Return area of intersection
return area1 + area2

```

## 2.3 Results

## 2.4 Conclusion

The average IOU score for all images is 0.5546657831124544.

## References

Your references here.

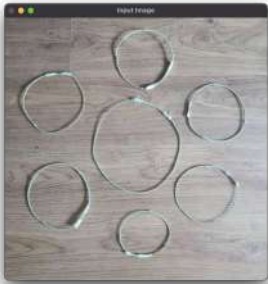
[1] [https://en.wikipedia.org/wiki/Canny\\_edge\\_detector](https://en.wikipedia.org/wiki/Canny_edge_detector)

[2] [https://en.wikipedia.org/wiki/Hough\\_transform](https://en.wikipedia.org/wiki/Hough_transform)

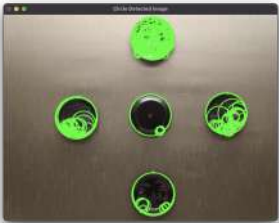


IoU value	Input Image	Detected Circles
-----------	-------------	------------------

0.9616909188630635



0.7792345799718526



0.7792345799718526













0.9562050445379743



0.9934755511628088



IoU value	Input Image	Detected Circles
0.810333247057329		
0.8870282239852926		
0.9562050445379743		
0.9611184623165501		
0.972724020240628		

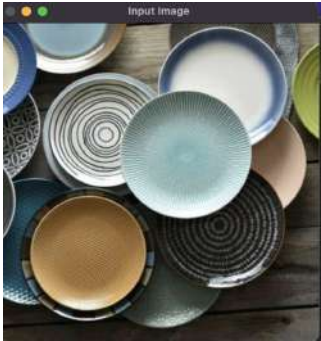
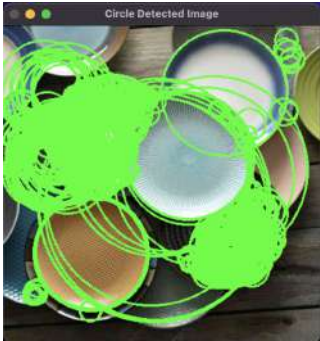
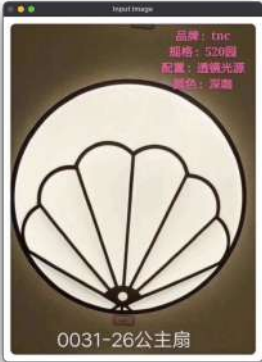




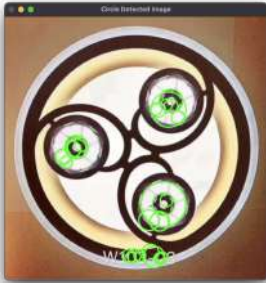

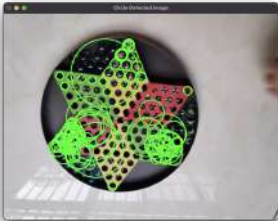
IoU value	Input Image	Detection Fails Circles
0.45974799253862225		
0.0		
0.0		
0.0		
0.2123132213		

Table 1: Circle detection fails