****

**KUET** | **Khulna University of Engineering & Technology**

**Image Segmentation, Canny Edge Detection, Hough Circle Transform**

**Prepared for**

Dr. Sk. Md. Masudul Ahsan

CSE 6243

Advanced Digital Image Processing

**Prepared by**

Md. Shameem Hossain

Roll: 1807560

**Submission Date: 21-04-2019**

**Implementation:**

Algorithm Steps:

**Step 1:**

Take an image and resize the image .

my\_image = Image.open(sys.argv[1])  
input\_image = my\_image.resize((420, 300))  
output\_image = Image.new("RGB", input\_image.size)

**Step 2:**

Now Find the circles edges of the given image by applying canny edge detection algorithm.

for x, y in canny\_edge\_detector(input\_image):  
 draw.point((x, y), (255, 255, 255))

output\_image.show()

The following function canny\_edge\_detector take an image and it transform the image into grayscale ,then compute the blur which remove the noise and compute the gradient.

def canny\_edge\_detector(input\_image):  
 input\_pixels = input\_image.load()  
 width = input\_image.width  
 height = input\_image.height  
  
 # Transform the image to grayscale  
 grayscaled = compute\_grayscale(input\_pixels, width, height)  
  
 # Blur it to remove noise  
 blurred = compute\_blur(grayscaled, width, height)  
  
 # Compute the gradient  
 gradient, direction = compute\_gradient(blurred, width, height)  
  
 # Non-maximum suppression  
 filter\_out\_non\_maximum(gradient, direction, width, height)  
  
 # Filter out some edges  
 keep = filter\_strong\_edges(gradient, width, height, 20, 25)  
  
 return keep

# The function Transform the image to grayscale  
def compute\_grayscale(input\_pixels, width, height):  
 grayscale = np.empty((width, height))  
 for x in range(width):  
 for y in range(height):  
 pixel = input\_pixels[x, y]  
 grayscale[x, y] = (pixel[0] + pixel[1] + pixel[2]) / 3  
 return grayscale  
  
# The function Blur it to remove noise  
def compute\_blur(input\_pixels, width, height):  
 # Keep coordinate inside image  
 clip = lambda x, l, u: l if x < l else u if x > u else x  
  
 # Gaussian kernel  
 kernel = np.array([  
 [1 / 256, 4 / 256, 6 / 256, 4 / 256, 1 / 256],  
 [4 / 256, 16 / 256, 24 / 256, 16 / 256, 4 / 256],  
 [6 / 256, 24 / 256, 36 / 256, 24 / 256, 6 / 256],  
 [4 / 256, 16 / 256, 24 / 256, 16 / 256, 4 / 256],  
 [1 / 256, 4 / 256, 6 / 256, 4 / 256, 1 / 256]  
 ])  
  
 # Middle of the kernel  
 offset = len(kernel) // 2  
  
 # Compute the blurred image  
 blurred = np.empty((width, height))  
 for x in range(width):  
 for y in range(height):  
 acc = 0  
 for a in range(len(kernel)):  
 for b in range(len(kernel)):  
 xn = clip(x + a - offset, 0, width - 1)  
 yn = clip(y + b - offset, 0, height - 1)  
 acc += input\_pixels[xn, yn] \* kernel[a, b]  
 blurred[x, y] = int(acc)  
 return blurred  
  
# The function Compute the gradient  
def compute\_gradient(input\_pixels, width, height):  
 gradient = np.zeros((width, height))  
 direction = np.zeros((width, height))  
 for x in range(width):

for y in range(height):  
 if 0 < x < width - 1 and 0 < y < height - 1:  
 magx = input\_pixels[x + 1, y] - input\_pixels[x - 1, y]  
 magy = input\_pixels[x, y + 1] - input\_pixels[x, y - 1]  
 gradient[x, y] = sqrt(magx\*\*2 + magy\*\*2)  
 direction[x, y] = atan2(magy, magx)  
 return gradient, direction

# The function Non-maximum suppression  
def filter\_out\_non\_maximum(gradient, direction, width, height):  
 for x in range(1, width - 1):  
 for y in range(1, height - 1):  
 angle = direction[x, y] if direction[x, y] >= 0 else direction[x, y] + pi  
 rangle = round(angle / (pi / 4))  
 mag = gradient[x, y]  
 if ((rangle == 0 or rangle == 4) and (gradient[x - 1, y] > mag or gradient[x + 1, y] > mag)  
 or (rangle == 1 and (gradient[x - 1, y - 1] > mag or gradient[x + 1, y + 1] > mag))  
 or (rangle == 2 and (gradient[x, y - 1] > mag or gradient[x, y + 1] > mag))  
 or (rangle == 3 and (gradient[x + 1, y - 1] > mag or gradient[x - 1, y + 1] > mag))):  
 gradient[x, y] = 0

# The function Filter out some edges  
def filter\_strong\_edges(gradient, width, height, low, high):  
 # Keep strong edges  
 keep = set()  
 for x in range(width):  
 for y in range(height):  
 if gradient[x, y] > high:  
 keep.add((x, y))  
  
 # Keep weak edges next to a pixel to keep  
 lastiter = keep  
 while lastiter:  
 newkeep = set()  
 for x, y in lastiter:  
 for a, b in ((-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)):  
 if gradient[x + a, y + b] > low and (x+a, y+b) not in keep:  
 newkeep.add((x+a, y+b))  
 keep.update(newkeep)  
 lastiter = newkeep  
  
 return list(keep)

**Step 3:**

For image segmentation, Changed the background color of image and circles background color of the image.

width, height = output\_image.size  
start = (int(0 \* width), int(0 \* height))  
any\_color = (255, 100, 150, 0)  
ImageDraw.floodfill(output\_image, xy=start, value=any\_color)  
output\_image.show()

**Step 4:**

Now take the user inputs of min. radius , max. radius, steps and threshold of the image. Calculate the points from the strong edges of circle. As a reminder, the parametric equation of a circle of radius rand center (a,b) is:

x=a+r⋅cos(t)

y=b+r⋅sin(t)

with t∈[0,2π)

rmin = int(sys.argv[2])  
rmax = int(sys.argv[3])  
steps = int(sys.argv[4])  
threshold = float(sys.argv[5])  
  
print("Min Radius = " + str(rmin) + ", Max Radius = " + str(rmax) + ", Steps = " + str(steps) + ", Threshold = " + str(threshold))  
points = []  
for r in range(rmin, rmax + 1):  
 for t in range(steps):  
 points.append((r, int(r \* cos(2 \* pi \* t / steps)), int(r \* sin(2 \* pi \* t / steps))))

**Step 5:**

Using the edges given by the Canny edge detector and for each possible circle, we count the number of edges that are part of each circle. However, instead of iterating over all the pixels of all circles, it is faster to iterate over the coordinates of each strong edge (x,y)and compute the coordinates of the center of all the circles that pass by that point. This is done using the equation above by setting rand t. For each of these circles, we increment a counter (a,b,r)(a,b,r).

In order to select which circles are good enough, we use two criteria: a threshold (here, at least 40% of the pixels of a circle must be detected) and we exclude circles that are too close of each other (here, once a circle has been selected, we reject all the circles whose center is inside that circle).

Mainly here is calculating the voting and draw red colour circles on the given image circles.

acc = defaultdict(int)  
for x, y in canny\_edge\_detector(input\_image):  
 for r, dx, dy in points:  
 a = x - dx  
 b = y - dy  
 acc[(a, b, r)] += 1  
  
circles = []  
for k, v in sorted(acc.items(), key=lambda i: -i[1]):

x, y, r = k  
 if v / steps >= threshold and all((x - xc) \*\* 2 + (y - yc) \*\* 2 > rc \*\* 2 for xc, yc, rc in circles):  
 print(v / steps, x, y, r)  
 circles.append((x, y, r))  
  
for x, y, r in circles:  
 draw\_result.ellipse((x - r, y - r, x + r, y + r), outline=(255, 0, 0, 0), width=3)  
  
  
output\_image.show()

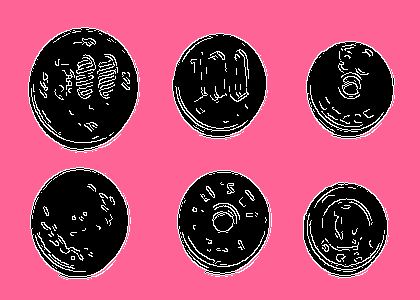
**Input:** (please provide the input image name, Min. Radius, Max. Radius, Steps and threshold)

python segment\_main.py input.jpg 10 110 110 0.3



**Output:**

**Min Radius = 10, Max Radius = 110, Steps = 110, threshold = 0.3**

****

