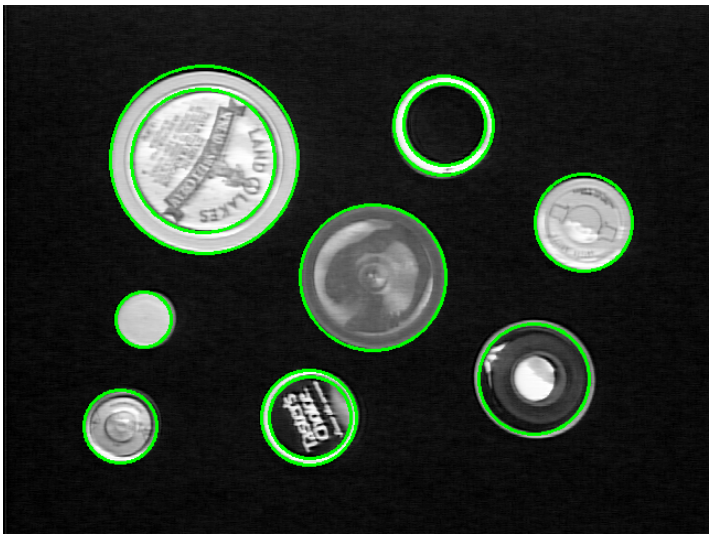
Khalil Dupoux
Computer Vision
Friday, March 29, 2024

Results:

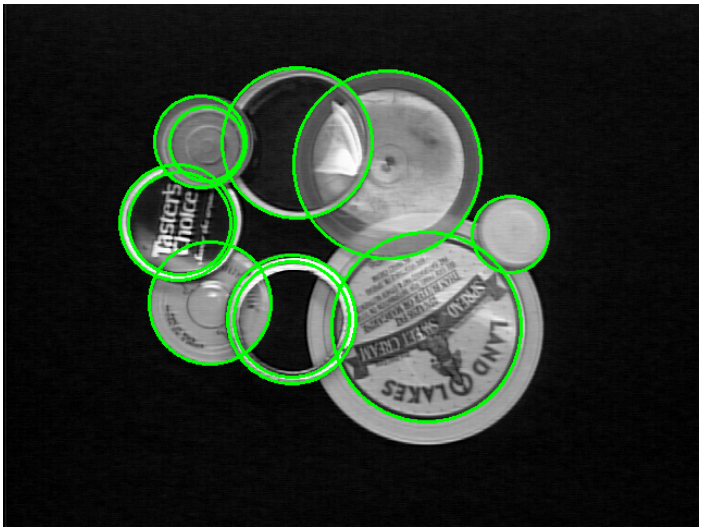
Output Image 1:



Groups:

x ,	y,	Radius,	Threshold
277 ,	374 ,	43.0 ,	0.66
106 ,	382 ,	33.0 ,	0.65
181 ,	141 ,	65.0 ,	0.61
335 ,	247 ,	66.0 ,	0.59
399 ,	109 ,	45.0 ,	0.57
400 ,	108 ,	37.0 ,	0.57
481 ,	339 ,	50.0 ,	0.55
526 ,	197 ,	44.0 ,	0.54
127 ,	285 ,	25.0 ,	0.52
182 ,	140 ,	85.0 ,	0.51
278 ,	374 ,	37.0 ,	0.51

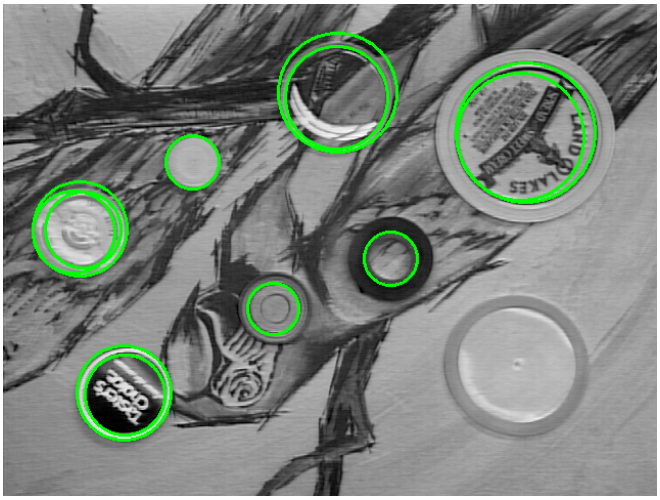
Output Image 2:



Groups:

x ,	y,	Radius,	Threshold
265 ,	289 ,	53.0 ,	0.62
265 ,	289 ,	59.0 ,	0.5733333333333334
160 ,	200 ,	53.0 ,	0.5666666666666667
190 ,	274 ,	56.0 ,	0.5466666666666666
389 ,	296 ,	87.0 ,	0.5266666666666666
466 ,	211 ,	35.0 ,	0.52
162 ,	201 ,	47.0 ,	0.4866666666666667
181 ,	126 ,	42.0 ,	0.47333333333333333
187 ,	128 ,	34.0 ,	0.47333333333333333
353 ,	147 ,	86.0 ,	0.4666666666666667
270 ,	127 ,	69.0 ,	0.46

Output Image 3:



Groups:

x ,	y,	Radius,	Threshold
119 ,	381 ,	39.0 ,	0.6133333333333333
511 ,	125 ,	68.0 ,	0.6066666666666667
326 ,	91 ,	50.0 ,	0.5933333333333334
118 ,	379 ,	46.0 ,	0.58
185 ,	154 ,	26.0 ,	0.5666666666666667
379 ,	248 ,	26.0 ,	0.5466666666666666
264 ,	297 ,	25.0 ,	0.5066666666666667
327 ,	86 ,	58.0 ,	0.49333333333333335
78 ,	220 ,	37.0 ,	0.48
69 ,	222 ,	40.0 ,	0.47333333333333333
506 ,	129 ,	62.0 ,	0.4666666666666667
75 ,	219 ,	46.0 ,	0.4666666666666667

READ ME:

In order to run my source file its important to have main.py and all the testing images in the same folder. The following commands from the terminal are the ones I used to get my results:

Image 1:
python main.py ./circles1.png --r_min 25 --r_max 90 --delta_r 1 --num_thetas 100 --min_edge_threshold 250 --max_edge_threshold 300

bin_threshold: 0.5

Image 2:
python main.py ./circles2.png --r_min 25 --r_max 100 --delta_r 1 --num_thetas 150 --min_edge_threshold 150 --max_edge_threshold 250

bin_threshold: 0.46

Image 3:
python main.py ./circles3.png --r_min 25 --r_max 100 --delta_r 1 --num_thetas 150 --min_edge_threshold 150 --max_edge_threshold 250

bin_threshold: 0.46

It's also important to note that in order to change the bin_threshold variable you must change the default value on line 95; I did this to get around a bug.

Write-up Description of Algorithm:

The find_hough_circles function first assumes the image given in the parameters is in gray scale. Some other parameters given to the function are:

- r_min - Min radius circle to detect.
- r_max - Max radius circle to detect.
- delta_r - Delta change in radius from r_min to r_max.
- num_thetas - Number of steps for theta from 0 to 2PI.
- bin_threshold - Thresholding value in percentage to shortlist candidate for circle. (not used)
- min_edge_threshold - Minimum threshold value for edge detection.
- max_edge_threshold - Maximum threshold value for edge detection.

First it calculates cos(theta) and sin(theta), it then evaluates and keep the candidate circles dx and dy ready for different delta radius based on the the parametric equation of circle:

x = x_center + r * cos(t)
y = y_center + r * sin(t)
where (x_center,y_center) is Center of candidate circle with radius r. t in range of [0,2PI)

Using two for loops two parse through the height and the width of the image to find any white pixels. Once one is found it will find and vote for circle from the candidate circles passing through this pixel. It then sorts the accumulator based on the votes for the candidate circles, and shortlists the best candidates. The algorithm goes through a final preprocessing stage in order to remove any circles in close proximity, or any duplicates.

Challenges:

The removing of duplicate circles was a challenge for me, but I managed with using the final step of preprocessing to remove them. Another challenge was avoiding any false circles, or false readings, and to avoid that I had to play around with some of the parameters, especially: num_thetas, bin_threshold, min_edge_threshold, and max_edge_threshold. These 4 parameters had the most effect on avoiding unnecessary output. r_min and r_max stayed the same throughout each experiment because the image was of the same sized circles varying in size.

Performance:

I'd say my algorithm performs well for test 1 and 2, but for 3 it fails to recognize the circle in the bottom left, and when running the program and looking at the Edge image provided as a test image, we can see that the circle is not complete, therefore making it harder for the algorithm to spot it. Some of the circles generated may seem like duplicates because they are inside another circle, but I chose to leave them because the algorithm was able to detect the different circles created by the grooves in the coin, or the difference colored metals on the coin.