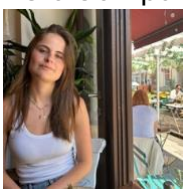
Neta Nakdimon
322216128
BS.c Computer Science

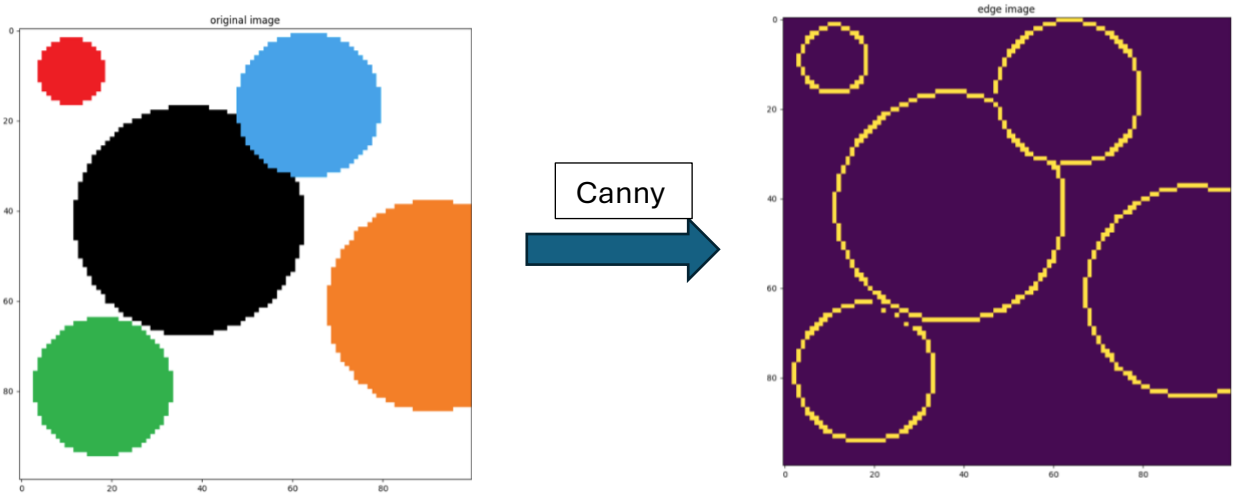


Hough Transform – Ex4b

Part A – Circle Hough Transform

The Circle Hough Transform is a popular method in image processing for detecting circles in images. It builds on the general Hough Transform, focusing specifically on circular shapes. The process involves voting in a parameter space and identifying the most likely circle centers and sizes(radii) by analyzing peaks in an accumulator matrix. Below, I explain the steps as implemented in the code.

i. Edge detection:
Used Canny to detect edges, which are possible circle boundaries (cv2.Canny)



ii. Accumulator Matrix Init:

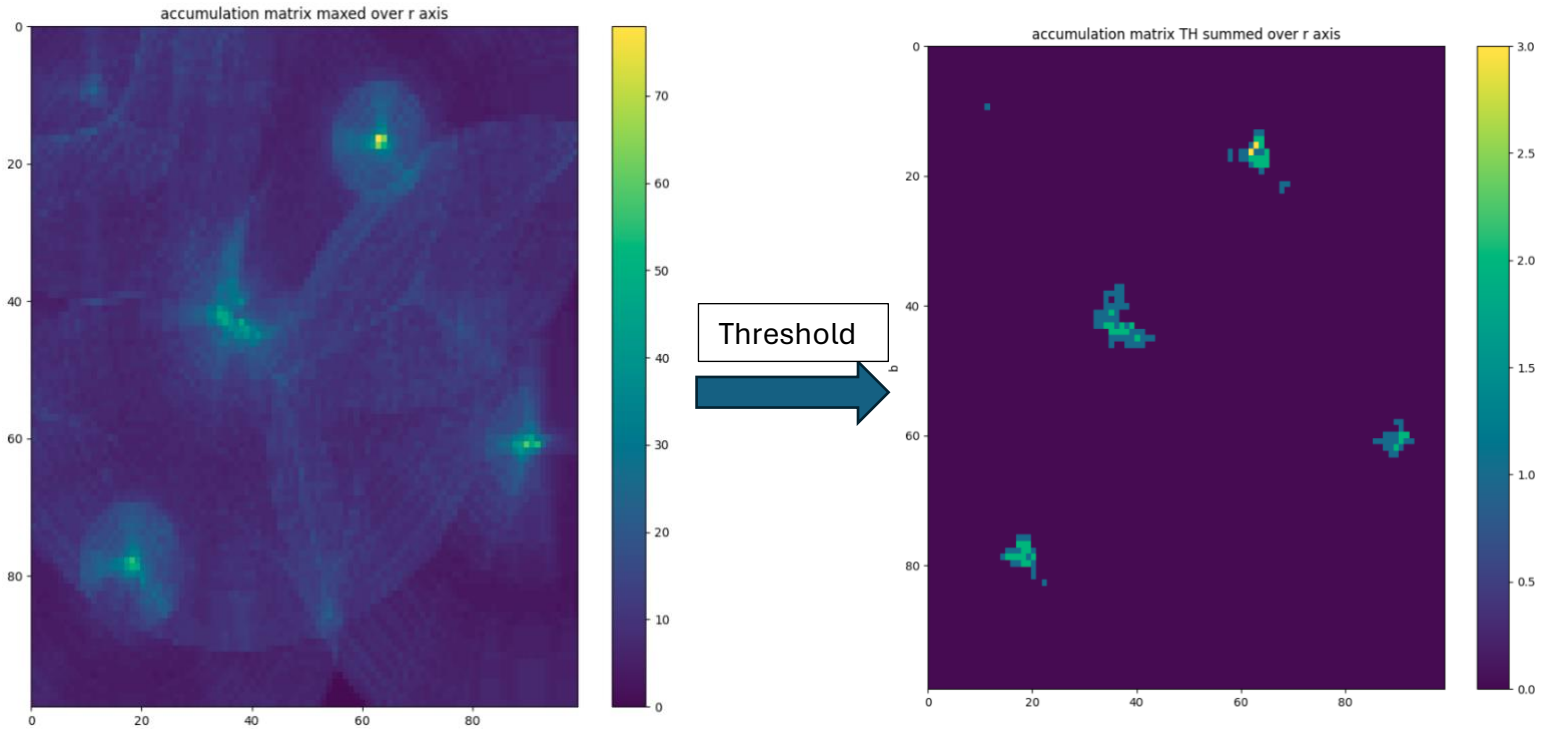
- Created a 3D matrix(acc_mat) where two dimensions represent potential circle centers and the third represents the possible radii of the circles.
- r_vec is an array of radii with range 1 to r_max = 25, which is incremented by r_step = 1.

```
acc_mat = np.zeros( shape: (rows, cols, len(r_vec)), dtype=np.int32)
```

iii. Voting in the Accumulator Matrix:

- Identified edge points: Located all pixels identified as edges using **np.argwhere.**
- Calculated circle votes: For each edge points (x, y) computed possible circle centers (a, b) for different radii using the formula:
  $r0 = \sqrt{(a0 - x)^2 + (b0 - y)^2}$
  If the radius is within bound, increment the corresponding cell in the accumulator matrix.

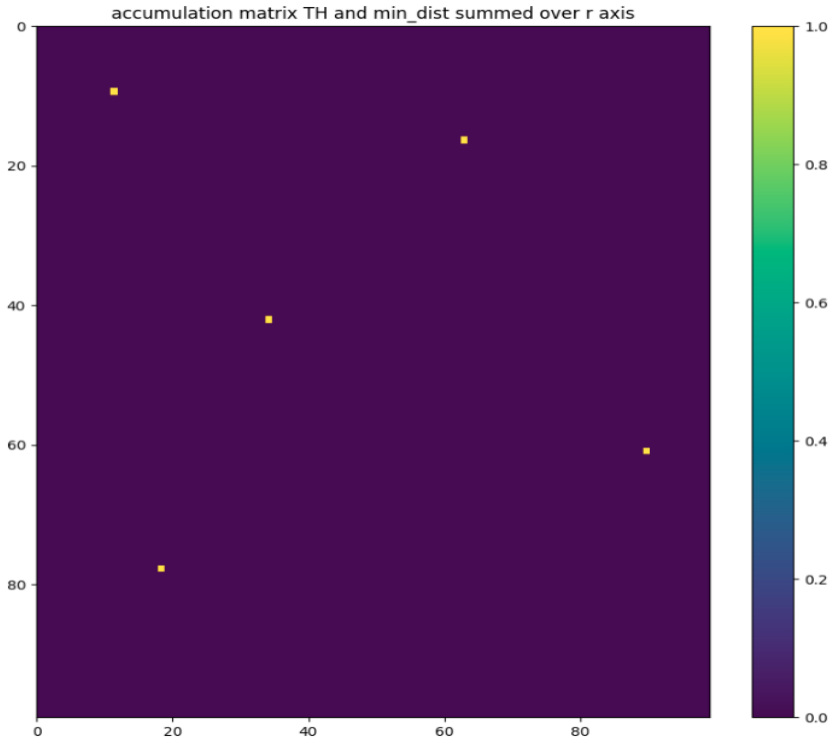
```
for yx in edge_inds:
    x = yx[1]
    y = yx[0]
    print("running on edge:" + str(yx) + "...")

    for a_ind, a0 in enumerate(a):
        for b_ind, b0 in enumerate(b):
            # TODO: find best corresponding r0 (1 line)
            r0 = np.sqrt((a0 - x) ** 2 + (b0 - y) ** 2)
            # something to make it faster
            if r0 > rmax:
                continue

            # TODO: find best index in r dimension (1 lin
            radius_index = np.argmin(np.abs(r_vec - r0))
            # TODO: update accumulation matrix (1 line)
            acc_mat[b_ind, a_ind, radius_index] += 1
```



iv. <u>Threshold and noise reducing:</u>

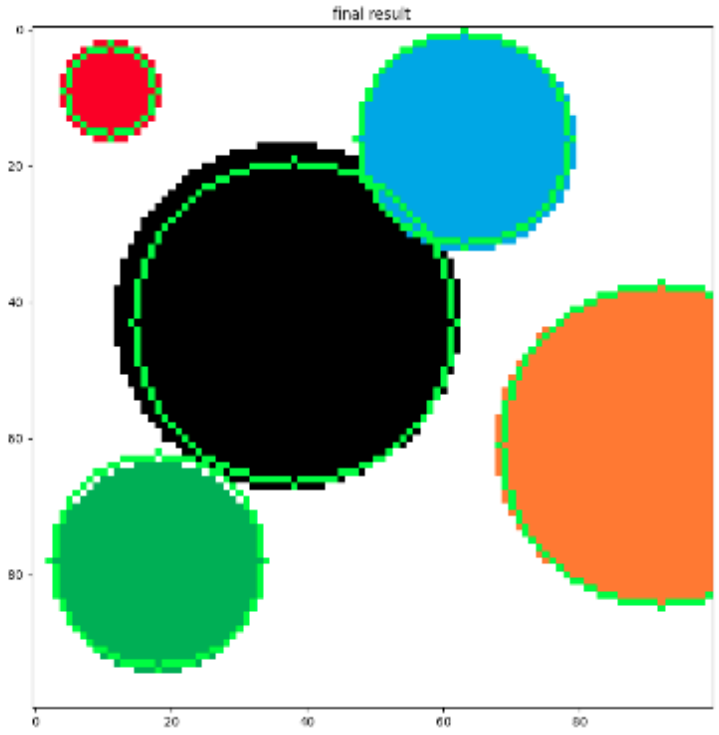
- Apply threshold: Only kept entries in acc_mat with votes higher than a set threshold (TH).
- Searched neighboring bins and eliminated those with fewer votes if they are too close.

```
# if the two above are neighbors (below min_dist) - delete the less important
if ((r0 - r1) * r_step) ** 2 + ((a0 - a1) * a_step) ** 2 + (
    (b0 - b1) * b_step
) ** 2 < min_dist**2:
    if acc_mat[b0, a0, r0] >= acc_mat[b1, a1, r1]:
        # TODO: one line fill here
        acc_mat_th_dist[b1, a1, r1] = 0
    else:
        # TODO: one line fill here
        acc_mat_th_dist[b0, a0, r0] = 0
```

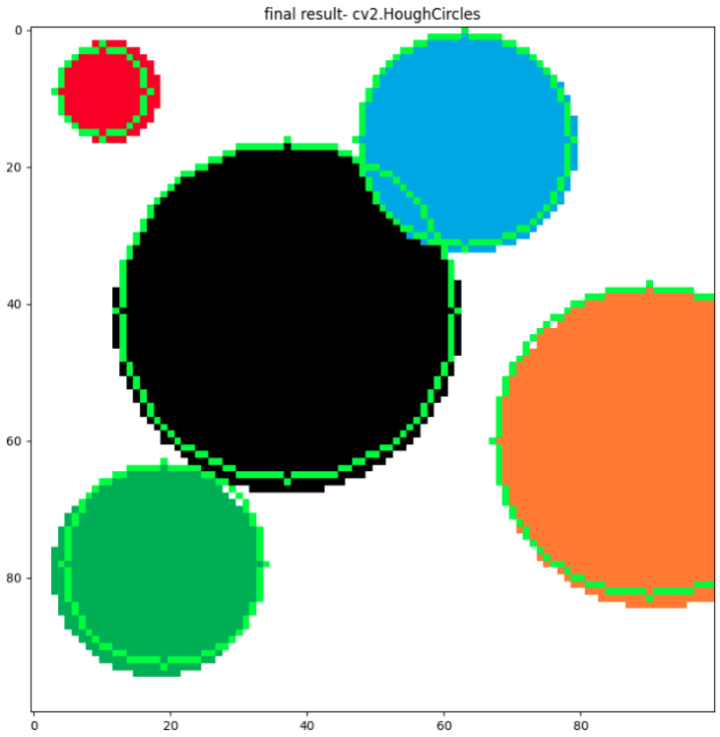


v. <u>Detect circles:</u>

- Extract circle parameters: Identified bins in the accumulator matrix with high votes (iterating over acc_mat)



My final result



Cv2 result

Part B – Coins

The code is used to detect and classify coins in an image based on their size using OpenCV's cv2.HoughCircles. The implementation focuses on parameter tuning and visualization for optimal detection.

i. acc_ratio: Represents the resolution of the accumulator array
   a. I set its value to 1 – meaning the accumulator has the same resolution as the input image.
ii. min_dist: Minimum distance between the centers of detected circles.
   a. This parameter should be at least the diameter of the smallest coin to prevent overlapping detections of the same coin. I set it to 70.
iii. canny_upper_th: Upper threshold for the Canny edge detector used in the Hough Transform.
   a. Higher values detect fewer, but more certain edges.
   b. I started with a typical value – 100 (read online it's a common th) and then adjusted it till I got to 500 which fitted the most.
iv. acc_th: Threshold for the accumulator, which determines the confidence level for circle detection.
   a. I started with a small value and after fine-tuning the other parameters I adjusted it (30 at last) to get rid of the irrelevant circles.
v. minRadius, maxRadius: The coins radiuses (size in pixels).
   a. I tried and errored (+looked online on coins sizes) until I got to the values minRadius = 15 and maxRadius = 70

My workflow was:
To detect coins in the image, I started by applying the Hough Circle Transform, using initial parameter values to identify potential circles. At this stage, my goal was to ensure that most coins are detected, even if it means including some irrelevant circles.
Next, I fine-tuned the parameters for accuracy:

- I adjusted the **minimum and maximum radius** values to match the expected sizes of the coins, ensuring the detected circles correspond to actual coins.
- I increased the **minimum distance** between detected circles to avoid overlapping detections within the same coin.
- I set the **edge detection threshold** (canny_upper_th) to balance sensitivity and noise filtering.
- Finally, I refined the **accumulator threshold** to eliminate weak or irrelevant circles, keeping only the most accurate detections.

Once the circles are detected, I classified the coins by their radii. Instead of converting the radius to physical units, I compared the pixel values directly against predefined thresholds

for each coin type. For example, smaller radii correspond to Dimes, while larger ones are classified as Nickels, Quarters, or Dollars.