# AIN433 Introduction to Computer Vision Lab. Assignment-2

**Alperen Ozcelik**
21992995
Department of Artificial Intelligence Engineering
Hacettepe University
Ankara, Turkey
`b21992995@cs.hacettepe.edu.tr`

## Overview

Our goals in this assignment were to gain practice in the importance of edge detection, hough transform, and cross correlation. Our inputs in the assignment were coins. In the first part of the assignment, we applied edge detection and hough circle transform, and in the second part, we find the offsets of the template images on the ground truth with cross correlation.

## 1 PART 1: Edge Detection and Hough Transform

Hough circle transform is an image transform that allows for circular objects to be extracted from an image, even if the circle is incomplete. The transform effectively searches for objects with a high degree of radial symmetry, with each degree of symmetry receiving one "vote" in the search space. By searching a 3D Hough search space, the transform can measure the center and radius of each circular object in an image.

### 1.1 My Approach

Our aim in the first part of the assignment is to find the circles on the coin images. While doing this, we had to implement Hough Circle Transform and use one of the edge detection methods. I used Canny edge detection because as we saw in our Computer Vision lesson, it is a more useful and preferred method than other edge filters.

First, I stored the original and canny edge filtered images on the arrays by reading the entire dataset and determining the appropriate thresholds for canny edge detection.

Then I wrote my function called CircleDetection. In this function, we give the picture with canny edge detection, the minimum and maximum radius values in the circles we want to find, and the minimum number of votes that a value in the accumulator array which we can use as an optional threshold as parameters.

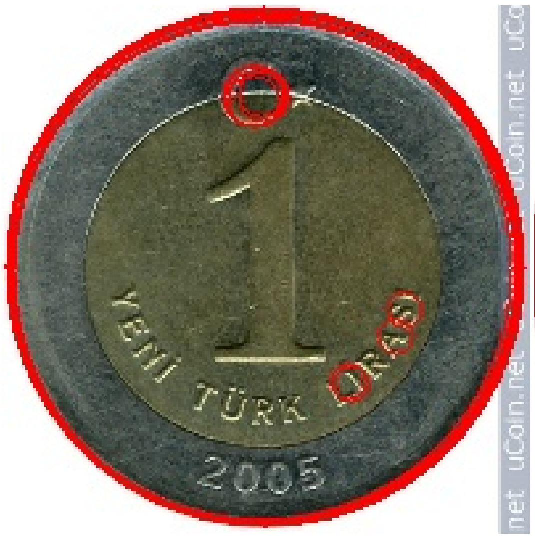
After I create an accumulator matrix, the Accumulator matrix is to find the intersection point in the parameter space. The element in the accumulator matrix denotes the number of circles in the parameter space that passes through the corresponding grid cell in the parameter space. The number is also called the "voting number". For each "edge" point in the original space, we can formulate a circle in the parameter space and increase the voting number of the grid cell which the circle passes through. This process is called “voting”. I had to do this voting process with 3 nested for loops as in the pseudocode we saw in the Computer Vision course.

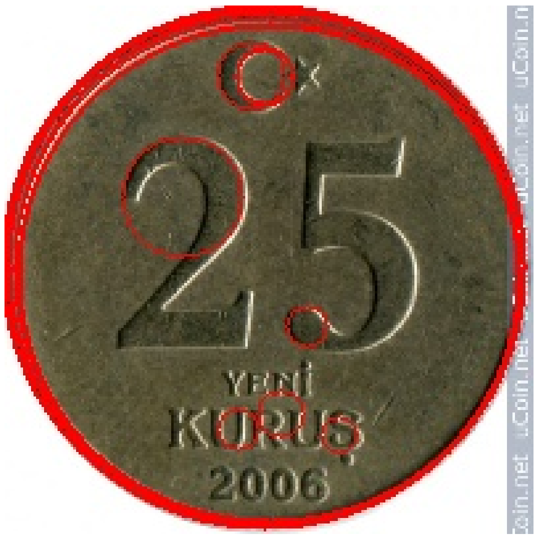
In the outermost for loop, there is the radius range that circles can take. In the second for loop, we look at all the pixels in the picture, and in the innermost for loop, we calculate the theta angle values of the bo points. With the next little formula, each point has a value of a, b, and r, and it votes in the accumulator matrix. as the same values come, the number of votes in the accumulator matrix increases by one.

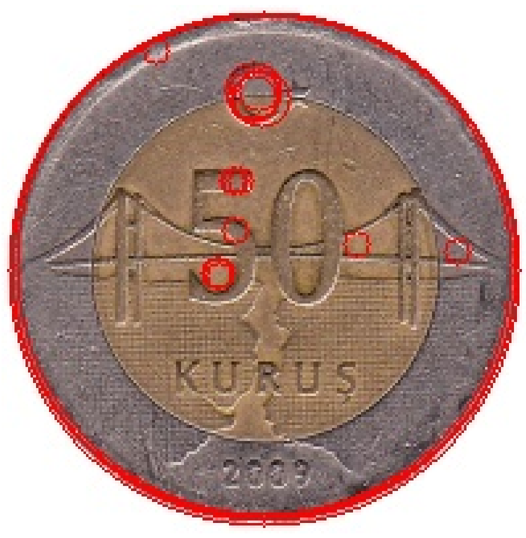
After voting, we can find local maxima in the accumulator matrix. The positions of the local maxima are corresponding to the circle centers in the original space.

Finally, all I have to do is draw these circles with red color on the original pictures and save the picture and show it to the screen. While showing it to the screen, I showed the original picture, the picture with canny edge detection and hough circle transform applied image together.

### 1.2 Part-1 Samples


(a) output-1


(b) output-2


(c) output-3

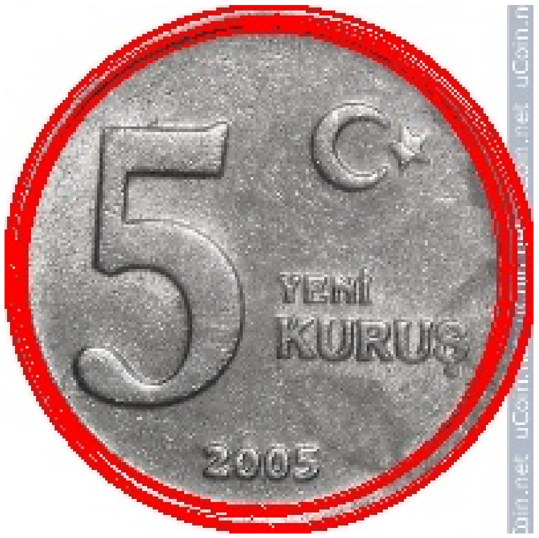

(d) output-4

Figure 1: My output images in part-1

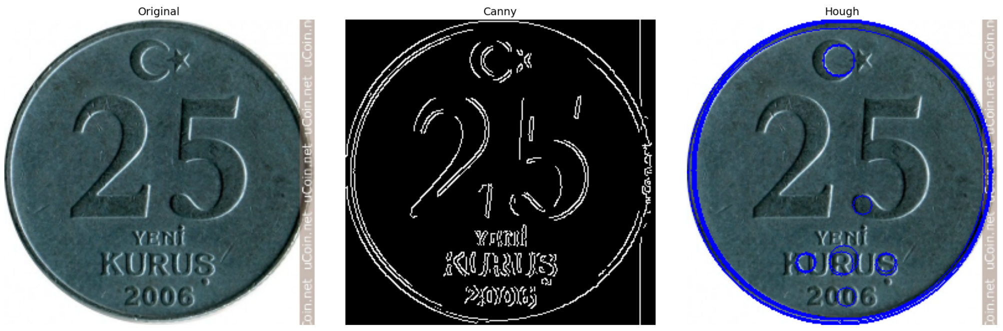


Figure 2: My output on screen

### 1.3 Discussion

As seen in the examples, its performance looks good in some pictures. In some pictures, he can find circles in unrelated places. It also finds more than one circle in each example outside the coin. These problems are problems that can be in the algorithm and it takes time to solve them.

For example, for circles of similar radius and similar centers, a single circle can be drawn. Apart from this, the method can become much more efficient by applying methods such as min-max suppression. But the most challenging part of this assignment is the time constraint created by 3 nested for loops.

Although I have halved the Image shape, in my current code it takes about 20 minutes for each image. Therefore, even if I make a mistake in coding, I have to wait half an hour to see if I have made a mistake. Because of this, I don't have much opportunity to optimize, but my code implements the algorithm exactly.

Finally, after I wrote my code, I ran my computer for about 3 hours and I was able to get the first 5 outputs, so I couldn't put as many image samples in report as you wanted.

## 2 PART 2: Cross Correlation

Correlation is the process of moving the template around the image area and computing the value in that area. This involves multiplying each pixel in the template by the image pixel that it overlaps and then summing the results over all the pixels of the template. The maximum value indicates the position where template best matches in image.

### 2.1 My Approach

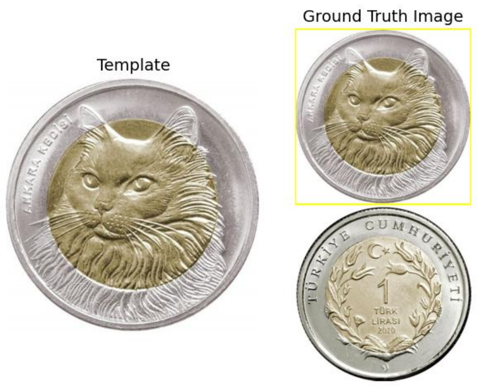
Our aim in the second part of the assignment is to find the offsets of the template images which are coins on the ground truth. While doing this, we had to implement cross correlation. For this purpose, I created a class called OffsetDetector. This class basically consists of 3 functions.

Correlation is intensity dependant, so i was think that it's better to use normalized cross-correlation. My calculate cross correlation function takes a piece of our image with the same size as the template as a parameter, and gives us the normalized cross-correlation value as output.

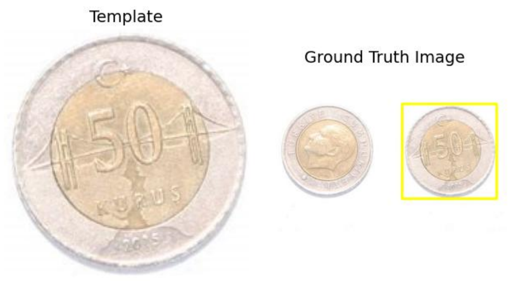
In the detect offset function, I first applied grayscale to both the image and the template, then I used a filter that navigates the image from the top left to the bottom right, and after sending the value to the calculate cross correlation, I returned the largest match value between the two images and returned the point that top left of the matching part.

My last function is called plot offset. This function takes the name of our output image as a parameter. It takes the coordinates of the upper left point of the square to be drawn, returned from the detect offset function, and encloses an area in the template size with a yellow square and displays the picture and saves it.

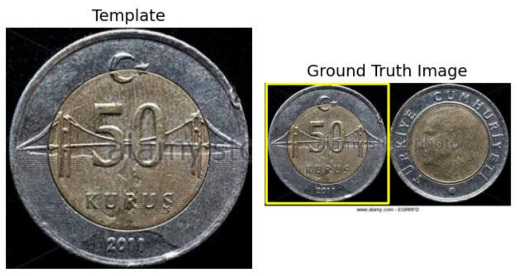
### 2.2 Part-2 Samples:



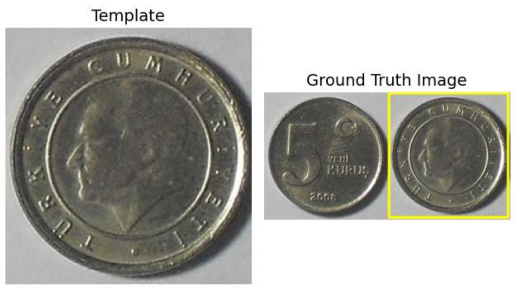
(a) output-1



(b) output-2



(c) output-3



(d) output-4

Figure 3: My outputs in part-2

## 3 Notes

In the first part, we had to give an output by applying the hough circle transform to all the pictures in the dataset one by one, but since it takes approximately 20 minutes for each picture to come out, I left the hough circle transform of a single picture as an example in my code.

To print all images one by one, all you have to do is write line 62 (in the comment line) instead of line 63 of the code.

In second part, our main goal was to search the entire dataset for the given template, find the picture the template is in, and mark the template in that picture.

I wrote a code block for this purpose, and it matches and marks 4 template pictures with the pictures they belong to as desired, but since this process takes about 40 minutes on my computer, I wrote the process as a comment line and you can easily see it in my code and you can run it that way.

Since this process takes a lot of time on my computer, I printed the segmentation map by directly matching the template images with the images they belong to. So you can easily see the output in 2-3 minutes. But if you wish, you can automatically find which template images match which image in the dataset, by running the above code block.

## References

[1] https://www.cis.rit.edu/class/simg782/lectures/lecture$_1 0/lec782_0 5_1 0.pdf$

[2] https://www.researchgate.net/publication/286592885$Approach_to_accurate_circle_detection_Circular_Hough_Transform_and_Local_Maxi$

[3] https://docs.opencv.org/4.x/da/d22/tutorial$_py_canny.html$

[3] https://medium.com/mlearning-ai/image-template-matching-using-cross-correlation-2f2b8e59f254