CS472-Assignment3

Iason Tzimas - csdp1333

April 23, 2024

1 Hough Transform Impplementation

First, the acquired images are preprocessed. The original images are of dimensions $(4032, 3024)$, and they are rescaled with a scaling factor of 0.25, which leads to the dimensions of $(1008, 756)$. Then, the homemade Canny Edge detector was applied with parameters $thresh_1 = 20$, $thresh_2 = 85$, $kernel = 3$. The $Canny()$ function was extended to also return the gradient angle and magnitude matrices, as well as a list of detected edge pixels that will be reused during the Hough Transform implementation.

The hough transform is carried out through the function is structured as follows:

Input Parameters

- *canny*: The canny edge detection output
- *edge_pixels*: A list of detected edge pixels
- *angle*: The gradient angle array
- *radius_start*: The first radius to start voting for
- *radius_stop*: The last radius to vote for
- *old_height*: The height of the input image
- *old_width*: The width of the input image
- *scaling_factor*: The scaling factor defining the grid on which to perform voting on
- *param*1: A hyperparameter that defines the factor with which to multiply the current radius to retrieve the lower score bound for detected circles

and returns:

- $fine\_tuned\_circles$: The circles that have been optimized to match the most edge pixels on the original image space
- *circles* : The detected circle prior to fine-tuning
- *hough_tensor*: A 3D Tensor containing the hough circle votes for each cell of the 3D grid
- *candidate_pixels*: The pixels that are considered good candidates for being circle centers

The function additionally creates to internal parameters:

- *param*2
- *suppression_dim*

$param2$, is used as a global threshold to classify detected maxima as circles or not. $suppression\_dim$, is the dimension of the 3D hypercube used to perform non-maxim-suppression and is explicitely set as:

$$suppression\_dim = int(100 \times scaling\_factor) \tag{1}$$

This is based on the idea that distant circle center pixels appear closer as the scaling factor decreases, meaning that one also has to reduce the kernel size of the non-maxima-suppression algorithm to adequately detect them. The above factor of 100 was empirically selected. The algorithm then calculates the grid dimensions for the voting space as:

$$height = int(scaling\_factor \times old\_height) + 100 \tag{2}$$

$$width = int(scaling\_factor \times old\_width) + 100 \tag{3}$$

The addition of 100 is to allow for the voting space to extend beyond the image boundaries to allow for the detection of corner cases where the circle center is outside the visible region.

Then, the algorithm iterates through all edge pixels generated by the $Canny()$ function. For, each one of them, the algorithm runs through all the possible radii, calculates the voting pixel and votes there. The angle matrix is used to cast votes along the gradient line extensions and not everywhere in a perimeter of a detected edge.

The voting pixel calculation is carried out as follows. In the outter loop, the values below are calculated for efficiency:

$$fact1 = np.sin(angle[pix[0], pix[1]]) \tag{4}$$

$$fact2 = np.cos(angle[pix[0], pix[1]]) \tag{5}$$

,where:

- pix: Is one of the detected edge pixels with original coordinates (not voting space coordinates)

Then, for each radius the following operations are carried out:

$$y = round((fact1 * rad + pix[0]) * scaling\_factor) + 50 \tag{6}$$

$$x = round((-fact2 * rad + pix[1]) * scaling\_factor) + 50 \tag{7}$$

Essentially, $fact1 * rad$ retrieves the x component of the radius in the original coordinate space. The addition of the pix coordinates accounts for the relative position of the pixel. The multiplication with the scaling factor gives rise to the relative voting pixel at voting space. And the addition with 50 accounts for the augmentation of the voting space by 100 to facilitate corner cases detection. A vote is then cast to the identified pixel. The vote is normalized by the scaling factor. The idea behind that is that in a coarser scale more votes are concentrated into one pixel than in a finer scale. The normalization with the scaling factor is done to allow for the invariance of the algorithm with respect to the various thresholds it uses for circle detection. Finally, to save time and not have to run through the entire tensor to perform non-maxima-suppression, candidate pixels are extracted on the fly with respect to the following criteria:

- $hough\_tensor[y, x, i] \geq param1 * rad$: Essentially, this a radii invariant threshold. The idea is that a certain circle holds a number of votes proportional to each radius and thus we can classify the circularity based on a threshold normalized by the radius. Param1 is a hyperparameter given by the user.
- $hough\_tensor[y, x, i] > param2$. A global threshold to discard any small spurious features and noise
- If the vote pixel is within the detection boundaries.

Having done that, each one of the candidate pixels is passed to the non-maxima-suppression process.

Finally, a fine tuning idea is implemented that has two stages. First, it iterates through a neighborhood of a detected circle and extracts the inner product of the Canny Edge Image with a circle of width 3, with parameters given by the detected circle plus the small perturbations of the neighborhood. This is done to give a fine-tuned circle that matches the actual circle edge better. Then, at the second stage, an eraser brush of width $w = 3$ pixels is applied to the location of the fine tuned circle, to prevent subsequent circles from using it as a support. This worked pretty fine, experimentally, as in many cases a False-Positive would use a combination of other circle perimeters to justify its existence. This was greatly mitigated with the above addition.

This process is visualized through the course of the following Figures:

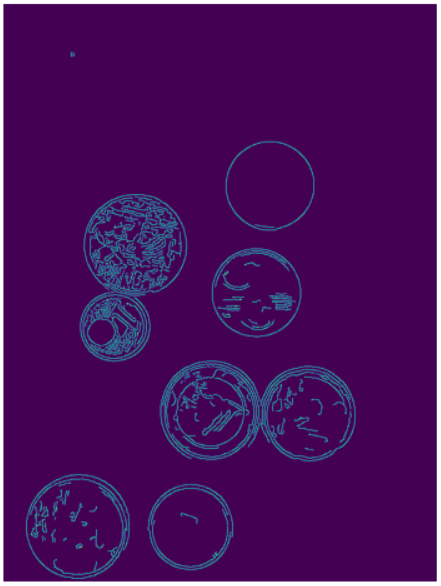

Figure 1: First Circle Removal

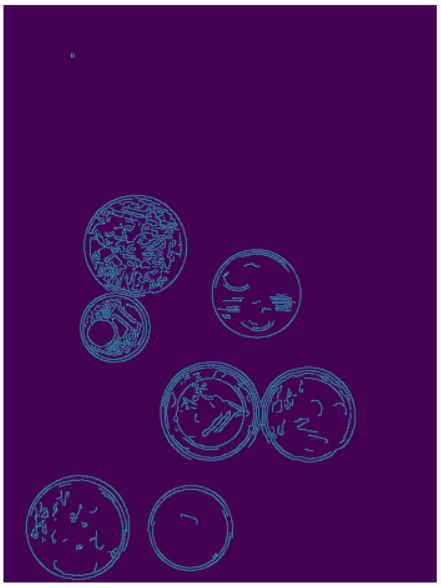

Figure 2: Second Circle Removal

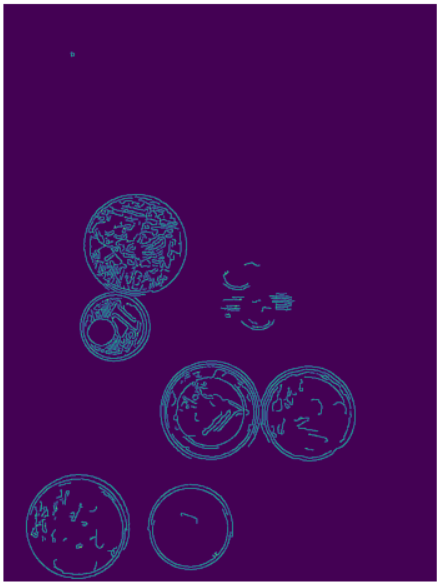

Figure 3: Third Circle Removal

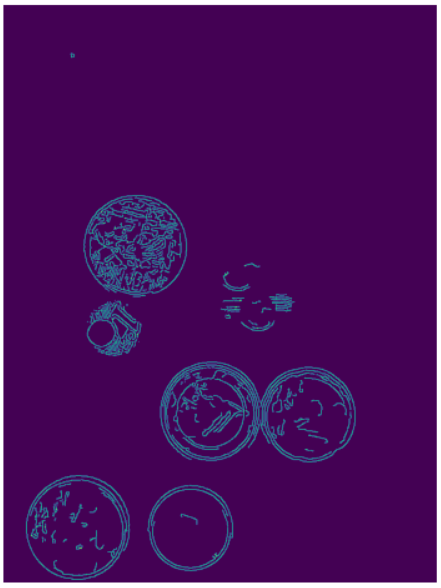

Figure 4: Forth Circle Removal

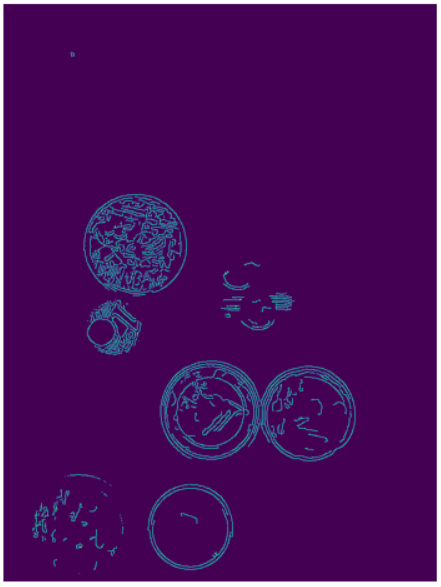

Figure 5: Fifth Circle Removal

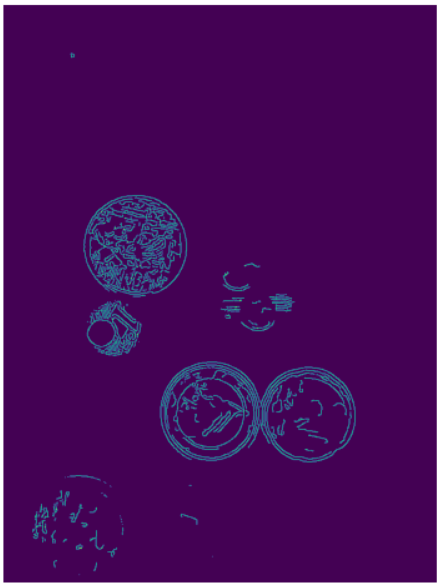

Figure 6: First Circle Removal

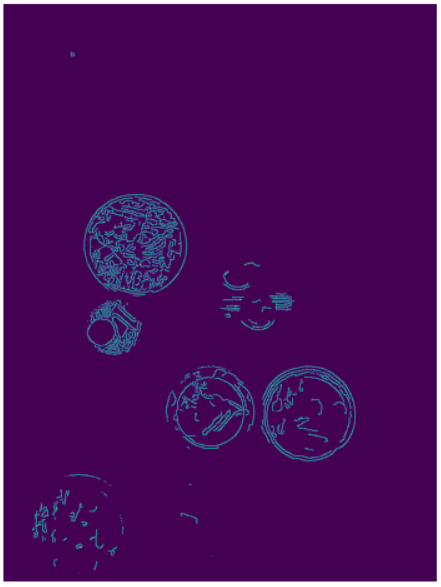

Figure 7: Sixth Circle Removal

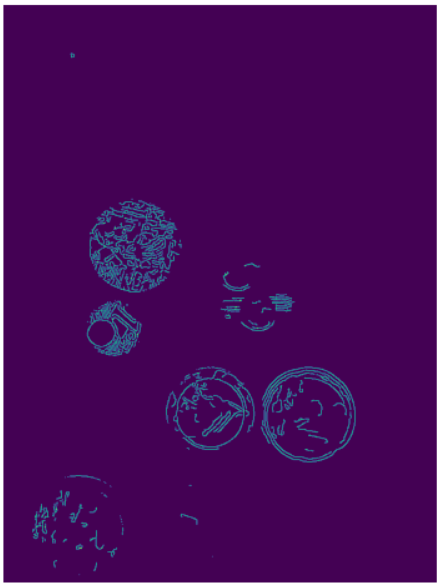

Figure 8: Seventh Circle Removal

Also, the priority for the deletion of edge pixels is with respect to the hough score of each circle. That is higher scoring circles remove their edges first so that they cannot be used by other less scoring ones.

This concludes the Hough Transform function.

The algorithm was run with the empirically chosen parameters that follow:

- $resize\_factor = 0.25$
- $gaussian\_kernel = 3$
- $thresh1 = 10$
- $thresh2 = 65$
- $canny\_kernel = 3$
- $radius\_start = 10$
- $radius\_end = 200$
- $scaling\_factor = 0.23$
- $param1 = 0.1$

Some outputs of the algorithm for various input images are presented below:

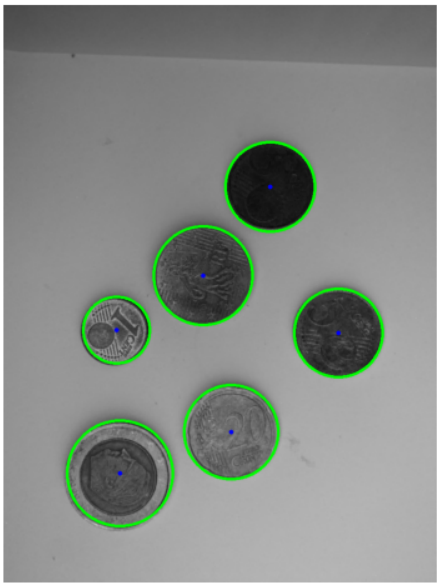

Figure 9

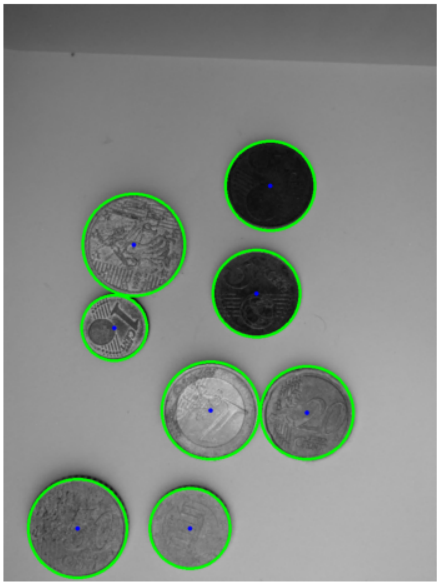

Figure 10

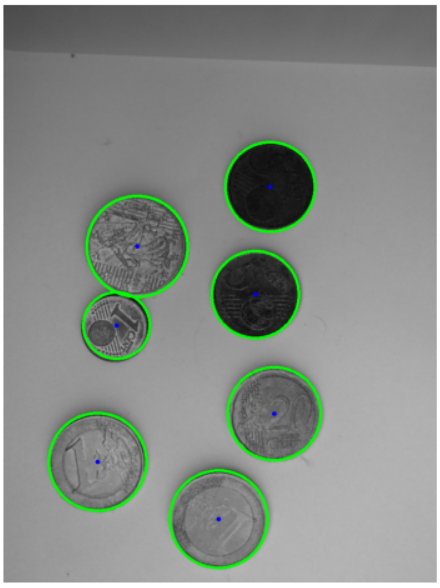

Figure 11

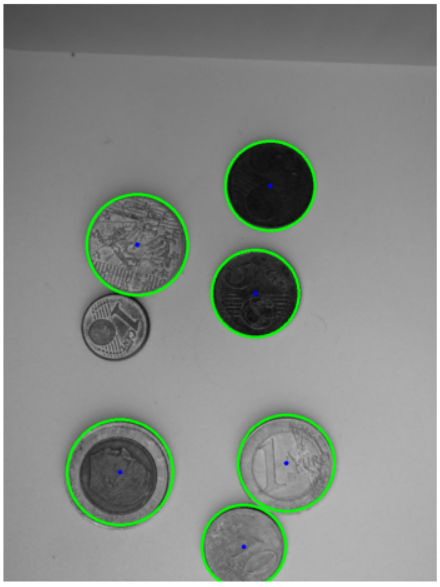

Figure 12

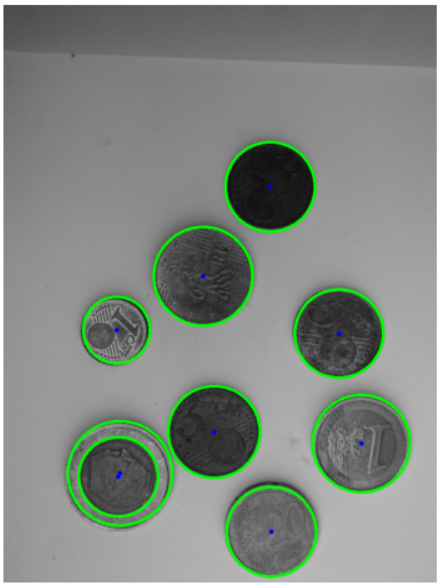

Figure 13

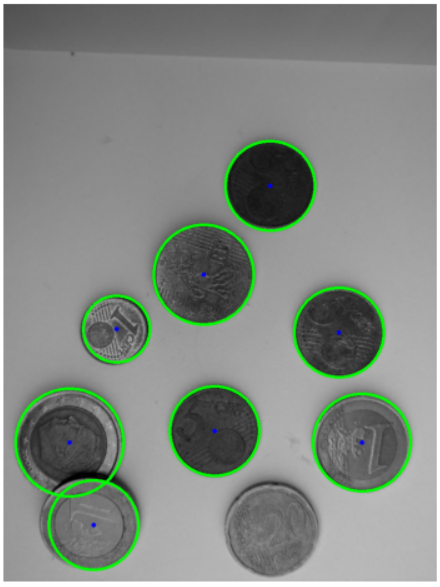

Figure 14

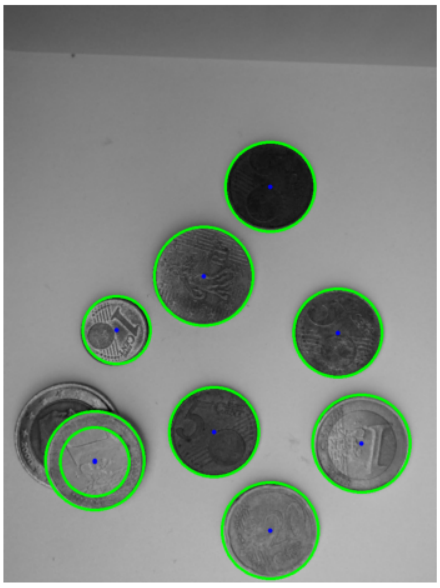

Figure 15

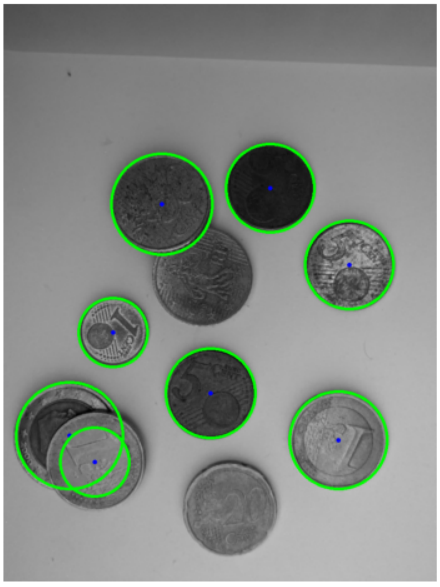

Figure 16

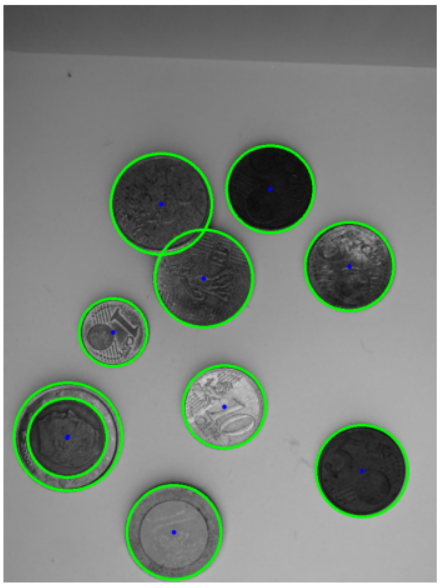

Figure 17

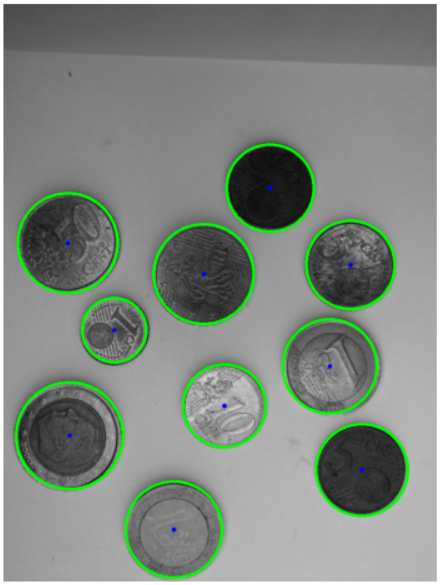

Figure 18

The results contain some failure cases, where non-overlapping coins could not be detected, overlapping coins could not be detected and concentric circles where detected or not for the cases of the 1euro and 2euro coins. The circle detection algorithm is also run for a textured background and the results are the following

Figure 19

Figure 20

Figure 21

The results are far inferior with the same selection of function input parameters. The parameters are tweaked a bit to allow for an easier edge pixel detection and the outputs for the same input images are as follows:

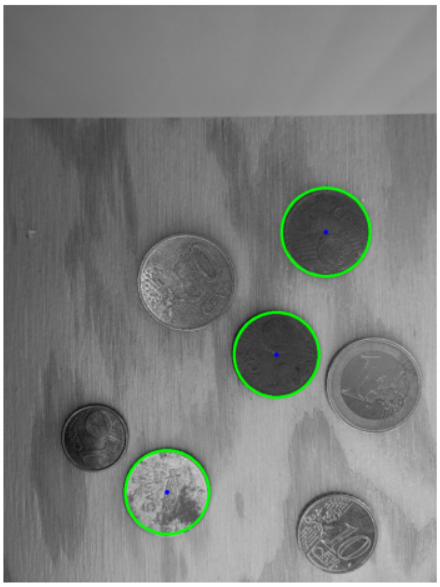

Figure 22

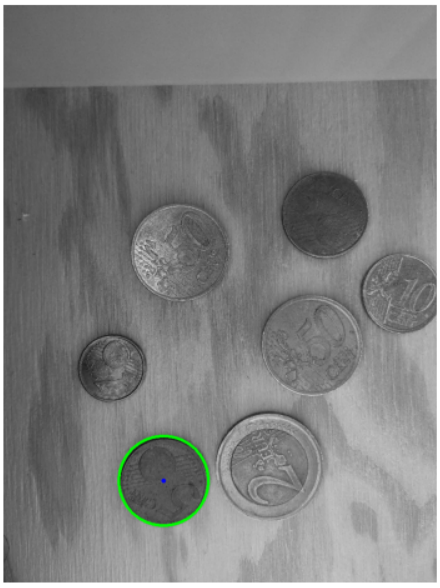

Figure 23

Figure 24

No improvement is observed whatsoever. SIFT feature, Generalized Hough Transform, Clustering Voting, could all be used to ameliorate the results.

2 Feature Extraction and Machine Learning

A total of 150 coin instances where collected for all coins. For each coin, a mask was applied for the corresponding detected circle. Then, for the masked region the following features were extracted:

- HSV value moments
- RGB value moments
- Gradient Magnitude histogram
- Gradient Angle histogram
- First Order Statistics (FOS) from PyFeats
- Gray Value Histogram
- Gray Level Difference Statistics (GLDS) from PyFeats
- Statistical Feature Matrix (SFM) from PyFeats
- Fractal Dimensional Texture Analysis (FDTA) features from PyFeats
- Detected Circle Radius

This lead to a total of 256 engineered features for each sample. The generated features were then reduced by means of variance thresholding. Then, the features were normalized by mean removal and std division. Further feature selection was implemented using the greedy SelectKBest method of SkLearn that iteratively calculates the ANOVA F-values for the provided features.

The 20 best features where retained. Then, an SVM model was trained with Hyperparameters $C = 100$ and an rbf kernel, to produce an average of 0.91 classification accuracy in a 10-Fold validation set.

The above pipeline was saved into an sklearn model, available as "Models/svc.pkl". Note that the above Machine Learning approach was chosen due to the ambiguity of the detected radius by itself. By incorporating texture, color and edge statistics the accuracy of the coin classification was vastly increased. Notice, however, that the classification does not offer any advantages for the Hough Transform Circle detection process. It just produces better classifications for the already detected circles.

The results for the Circle-Detection + Classification process are as follows in multicoin images:

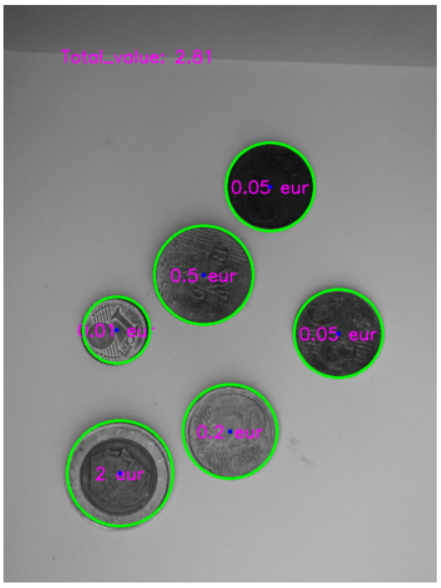

Figure 25

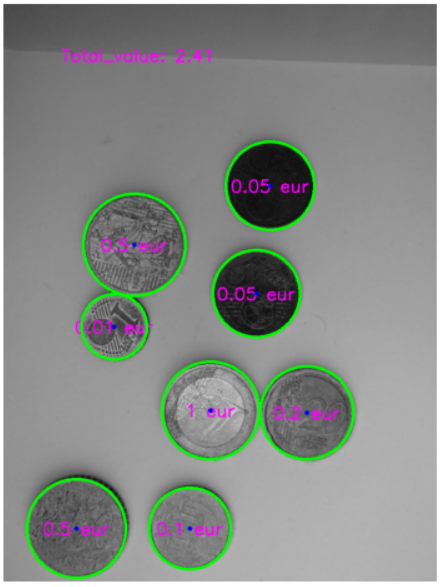

Figure 26

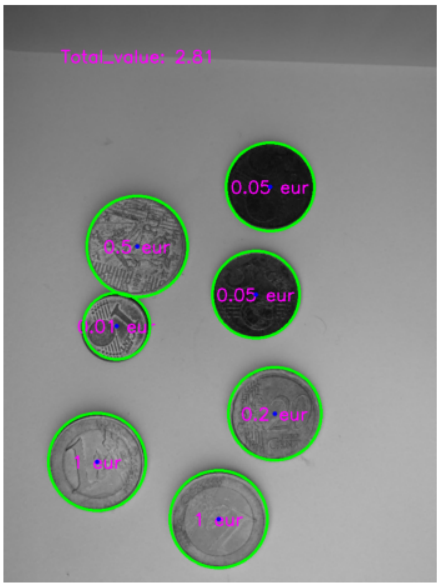

Figure 27

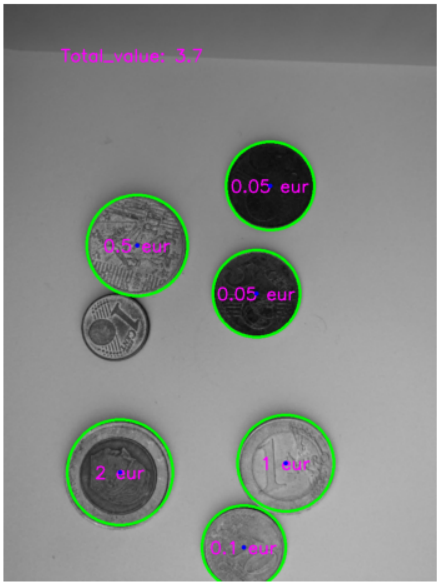

Figure 28

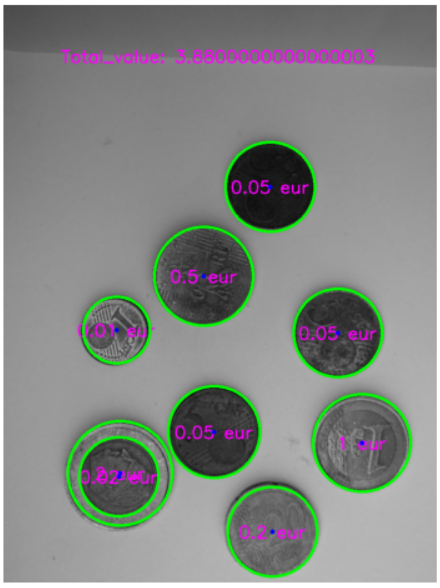

Figure 29

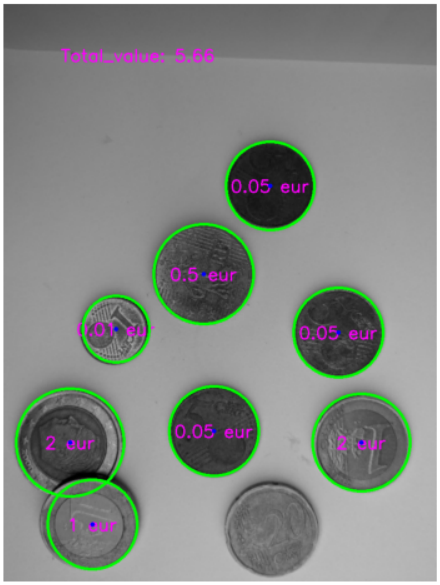

Figure 30

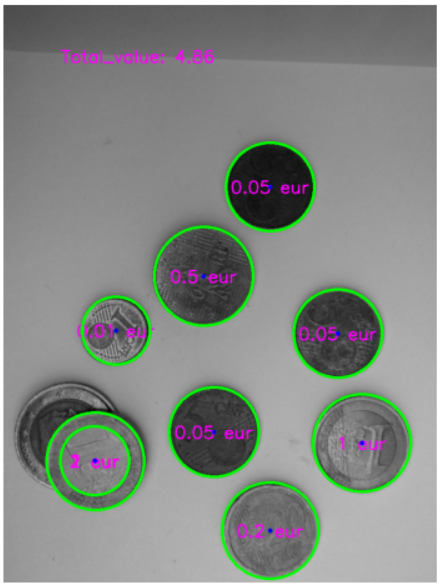

Figure 31

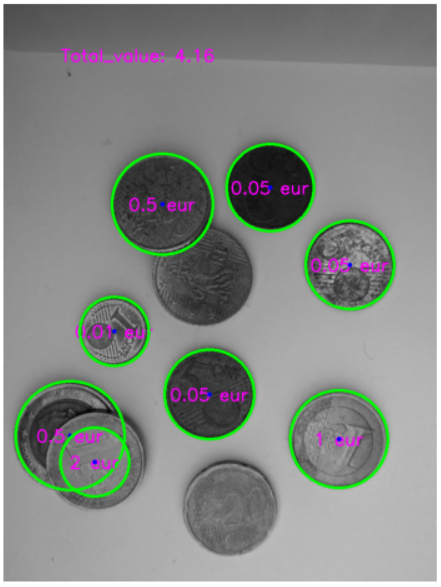

Figure 32

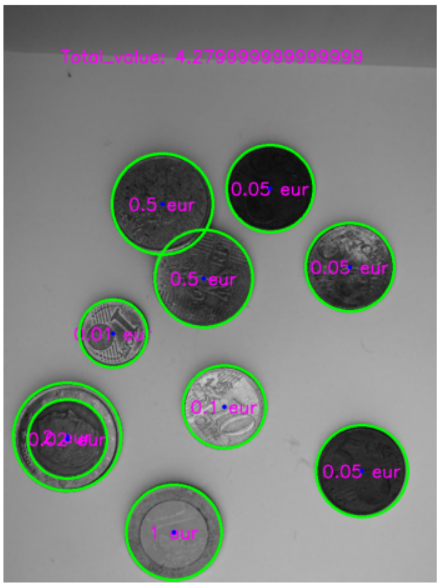

Figure 33

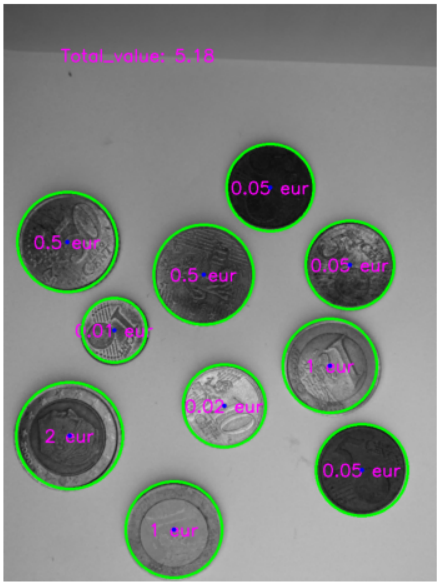

Figure 34