AIN433 - Report on Computer Vision Lab Assignment 2

Student Name:	Emre Çoban
Student ID:	2200765028
Date:	02.12.2023
Email:	b2200765028@cs.hacettepe.edu.tr

1 Introduction

A powerful technique, widely employed in image analysis, computer vision, and digital image processing, is represented by the Hough transform. Its role as an approach for feature extraction centers on the identification of instances of objects within a specific class of shapes. Imperfect instances of objects are detected by the Hough transform through the implementation of a voting procedure in a parameter space. Local maxima in an accumulator space are pinpointed, and this space is explicitly constructed during the transformation process.

The primary objective of this project is to foster familiarity with the Hough Transform and Histogram of Oriented Gradients (HoG) features. In the initial phase, the application of the Hough Transform to detect circles in provided images will be emphasized. Subsequently, in the second phase of the project, this approach will be employed to recognize coins based on their HoG features.

2 Part 1: Edge Detection and Hough Circle Transform

2.1 Edge Detection

The initial step includes the acquisition of edge maps for the images, facilitated by the subsequent procedures:

1. **Image Resizing**: The original image undergoes resizing to a maximum dimension of 296 pixels while maintaining the aspect ratio.

```
# Keeping original dimensions to later convert.
original_height, original_width = image.shape[:2]

# Calculate the resize ratio based on the image size
resize_ratio = min(1.0, 296.0 / max(original_height, original_width))

# Resize the image
resized_image = cv2.resize(image, None, fx=resize_ratio, fy=resize_ratio)
```

2. **Grayscale Conversion**: The resized image is converted to grayscale, simplifying the subsequent processing.

```
# Convert the image to grayscale
gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
```

3. **Blurring**: Application of a Gaussian blur to the grayscale image (kernel size: 7x7, sigma: 1) reduces noise and enhances edge detection.

```
# Blurring image for better edge detection
blurred = cv2.GaussianBlur(gray, (7, 7), 1)
```

4. **Canny Edge Detection**: Implementation of the Canny edge detection algorithm on the blurred image with thresholds set at 50 and 150.

```
# Apply Canny edge detection
edges = cv2.Canny(blurred, 50, 150)
```

Edge Detection Details: The Canny edge detection algorithm was chosen for its effectiveness in detecting edges with low error rates. We set the thresholds at 50 and 150 to capture a wide range of edges, balancing sensitivity and specificity. The chosen parameters were determined through experimentation to achieve optimal results.

2.2 Hough Circle Detection

Circle detection in the edge-detected image is performed using the Hough Circle Transform.

2.2.1 Parameters

- `min_radius`: The minimum radius of circles for detection.
- `max_radius`: The maximum radius of circles for detection.
- `skip_value`: The step size for radius values.
- `min_distance_penalty`: The minimum distance penalty to suppress overlapping circles (default: 0.8).
- `threshold`: The minimum number of votes for a circle to be considered valid (default: 140).

2.2.2 Implementation Details

1. **Initialization:** An accumulator array is created to store votes for circle parameters, including center coordinates and radii.

```
# Hough Transform for Circle Detection
accumulator = np.zeros((gray.shape[0], gray.shape[1], max_radius - min_radius + 1))
```

2. **Theta and Radius Iteration:** The algorithm iterates over theta values from 0 to 360 and radius values from min_radius to max_radius, utilizing the skip value for efficiency.

```
cos_theta = np.cos(np.radians(np.arange(360)))
sin_theta = np.sin(np.radians(np.arange(360)))

# For each possible circle (theta and radius):
for theta in range(0, 360):
    for r in range(min_radius, max_radius + 1, skip_value):
```

```
        # ... (code for iteration)
```

3. **Edge Point Transformation:** For each combination of theta and radius, edge points in the image are transformed to accumulate votes in the accumulator array.

```
edge_points = np.column_stack(np.where(edges > 0))

# For each possible circle (theta and radius:
for theta in range(0, 360):
    for r in range(min_radius, max_radius + 1, skip_value):
        a = np.round(edge_points[:, 0] - r * cos_theta[theta]).astype(int)
        b = np.round(edge_points[:, 1] - r * sin_theta[theta]).astype(int)

        # Check if point is valid
        valid_points = np.where((a >= 0) &
        (a < edges.shape[0]) & (b >= 0) & (b < edges.shape[1]))

        a = a[valid_points]
        b = b[valid_points]

        # Accumulate votes for each circle
        accumulator[a, b, r - min_radius] += 1
```

4. **Thresholding:** Potential circles are identified by finding coordinates in the accumulator array with votes above a specified threshold.

```
circle_coordinates = np.where(accumulator >= threshold)
```

5. **Distance Penalty:** Overlapping circles are suppressed by applying a distance penalty. Circles too close to each other are filtered out.

```
circles_resized = apply_distance_penalty(circles_resized, min_distance_penalty)
```

6. **Resize Transformation:** The detected circles are transformed back to the original image dimensions.

```
circles_original = [
(int(x / resize_ratio), int(y / resize_ratio), int(radius / resize_ratio))
    for x, y, radius in circles_resized
]
```

2.2.3 Distance Penalty Application

To address overlapping circles, a distance penalty is applied, removing circles that are too close. Circles are sorted by radius in descending order, retaining only the outer circle for easier counting in Part 2.

```
# Sort circles by radius in descending order
# Retain only the outer circle of coins for later counting
```

```
sorted_circles = sorted(circles, key=lambda x: x[2], reverse=True)

selected_circles = []

for circle in sorted_circles:
    x, y, r = circle
    # Check distance differences
    if not any(
        np.sqrt((x - cx) ** 2 + (y - cy) ** 2) < min_distance_penalty * (r + cr)
        for cx, cy, cr in selected_circles
    ):
        selected_circles.append(circle)
```

While experimenting with different sorting methods, including sorting by accumulator votes before applying the distance penalty, it was observed that sorting by radius in descending order produced more accurate results. This method demonstrated effectiveness in selecting the outer circles of coins, which is crucial for accurate counting and identification in Part 2 of the project. Hence, this sorting strategy was chosen for its superior performance in retaining relevant circles.

The `apply_distance_penalty` function takes a list of detected circles and a minimum distance penalty. It suppresses overlapping circles by checking the distance between circle centers and keeping only those that satisfy the specified distance condition.

2.3 Image Processing Pipeline

The `process_images` function processes each image in the input folder, detects circles using the implemented Hough Circle Detection, draws the detected circles on the original image, and saves the result in the output folder. The function accepts parameters such as the input and output folders, minimum and maximum radii, and skip value for radii.

Three datasets, namely "Train," "TestV," and "TestR," are processed by the code.

2.4 Results of Part 1

The comparison between original images and their corresponding images after circle detection is provided in the visualization section, featuring a subplot layout that displays the original image, the image with drawn circles, and the edge map obtained during the processing.

Excellent performance in identifying outer circles in the training images is demonstrated by the applied method, with accurate results produced for all instances. Efforts were made to optimize parameters during the experimentation phase for the test images, including the radius, to detect almost all outer coin circles. While success in achieving this objective is generally observed, there might be cases where the detected circles have slightly larger or smaller radii. This variability could be a result of the penalty function, as it sorts them first by radius, prioritizing the biggest circles. The choice of this method was based on its superior ability to generally find outer circles of the coins, leading to its continuation in this manner.

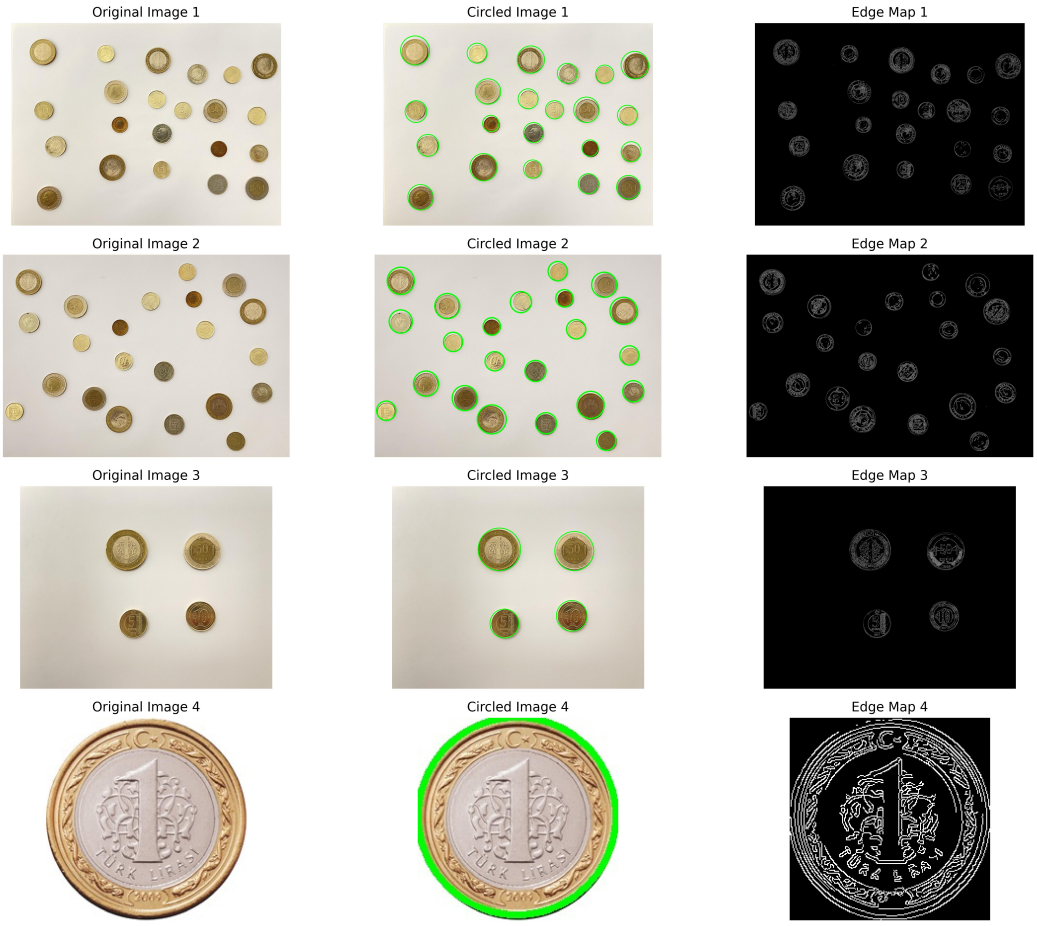

Figure 1: Results of Hough Circle Transform

3 Part 2: Coin Recognition using HoG Features and SVM

3.1 HoG Feature Extraction

The second phase of the project involves the recognition of coins using the Histogram of Oriented Gradients (HoG) features. This intricate process unfolds through the following key steps:

3.1.1 HoG Calculation

Coin recognition relies on the crucial aspect of HoG calculation, and it involves the following steps:

1. **Convert Image to Grayscale**:

   The first step is to convert the input color image (`img`) to grayscale. This conversion simplifies subsequent processing.

   ```
   # Convert the image to grayscale
   gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   ```

   The resulting grayscale image (`gray_img`) retains essential intensity information.

2. **Calculate Gradients**:

   Gradients provide information about the intensity change in both the x and y directions. The Sobel operator is applied to calculate these gradients.

   ```
   # Calculate gradients
   grad_x, grad_y = cv2.Sobel(gray_img, cv2.CV_64F, 1, 0, ksize=1),
                    cv2.Sobel(gray_img, cv2.CV_64F, 0, 1, ksize=1)
   ```

   The resulting `grad_x` and `grad_y` represent the intensity gradients in the horizontal and vertical directions, respectively.

3. **Calculate Magnitude and Direction of Gradients**:

   The magnitude and direction of gradients are essential for understanding the structure of objects in the image. These values are computed from the gradient components.

   ```
   # Calculate magnitude and direction of gradients
   magnitude = np.sqrt(grad_x**2 + grad_y**2)
   direction = np.arctan2(grad_y, grad_x) * (180 / np.pi) % 180
   ```

   The `magnitude` represents the overall intensity change, while `direction` indicates the orientation of the intensity change.

4. **Create Histogram Bins Based on Orientations**:

   To capture the distribution of gradient orientations, histogram bins are created based on the computed orientations.

   ```
   # Create histogram bins based on orientations
   bins = np.int32(num_orientations * direction / 180.0)
   ```

   The orientations are quantized into bins to form a histogram, providing a concise representation of the dominant gradient directions.

A crucial role in characterizing the structure and texture of coins is played by these HoG features, enabling effective recognition in the subsequent phases of the project.

3.1.2 SVM Training

To train a Support Vector Machine (SVM) classifier for coin detection using Histogram of Oriented Gradients (HOG) features, the following steps are taken:

1. **Function Definition**:

   The training process starts with the definition of a Python function named `train_svm`. This function is responsible for training the SVM classifier and returns the trained SVM model.

   ```
   def train_svm():
       """
       Trains a SVM classifier for coin detection using HOG features.

       Returns:
           svm_model: The trained SVM model.
       """
   ```

2. **Initialize Lists for Features and Labels**:

   Two lists, namely `train_features` and `train_labels`, are initialized to store features and corresponding labels for training.

   ```
   # Lists to store features and labels for training
   train_features = []
   train_labels = []
   ```

3. **Process Images in the "Train" Folder**:

   Each file in the 'Train' folder, where each file represents an image for training, is iterated through by the code

   ```
   # Process each image in the "Train" folder
   for filename in tqdm(os.listdir("Train")):
   ```

4. **Read and Resize Image**:

   Each image is read using OpenCV, and then it is resized to a consistent size (256x256 pixels).

   ```
   # Read the image
   image = cv2.imread(os.path.join("Train", filename))

   # Resize the image to a consistent size
   image = cv2.resize(image, dsize=(256, 256))
   ```

5. **Extract HOG Features**:

   Histogram of Oriented Gradients (HOG) features is extracted from the resized image using the `custom_hog` function.

   ```
   # Extract HOG features from the image
   features = custom_hog(image)
   ```

6. **Append Features and Labels to Lists**:

   Extracted HOG features and corresponding labels are appended to the `train_features` and `train_labels` lists.

   ```
   # Add the features and labels to the training data
   train_features.append(features)
   train_labels.append('_'.join(filename.split("_")[:-1]))
   ```

7. **Train SVM Model**:

   An SVM model is trained using a linear kernel with the specified parameters, including the regularization parameter ($C = 1.0$).

   ```
   # Train an SVM model using a linear kernel
   svm_model = SVC(kernel="linear", C=1.0, random_state=42)
   svm_model.fit(train_features, train_labels)
   ```

   The `random_state` parameter is set for reproducibility.

3.2 Coin Detection and Recognition

At the core of the coin recognition process, the detect_and_draw_coins function skillfully orchestrates the detection and recognition of coins in test images. Initially, the application of the Hough Circle Transform is executed, as elucidated in the first part of the assignment. Following this, the circled regions identified by the Hough Transform are encapsulated within bounding boxes defined by coordinates (x+r, x-r, y+r, y-r).

Subsequently, these bounding boxes undergo a resizing operation to dimensions of 256x256, mirroring a step that was similarly undertaken during the training phase. This meticulous sequence of operations seamlessly integrates the insights gained from the Hough Circle Transform with the refined adjustments acquired from the adept training of the SVM classifier.

```
# Example Usage
test_folder = "TestV"
output_folder = "TestV_HoG"
min_radius = 7
max_radius = 30

trained_svm_model = train_svm()
detect_and_draw_coins(test_folder, output_folder, trained_svm_model,
                      min_radius, max_radius)
```

Same process was applied for TestR folder too.

3.3 Results of Part 2

In the results section, it was observed that accurate identification of coin classes by the model occurs when the coins are large in the images. However, as the coins decrease in size, a higher probability of misclassification is encountered. The potential root cause for this issue may be attributed to the limited number of examples for each class in the training data, with only five instances per class. The model's challenge in correctly identifying smaller coins is likely influenced by the scarcity of data for each respective class.

Figure 2: Results of Coin Recognition using HoG Features and SVM (Exemplary Outcome Test V)

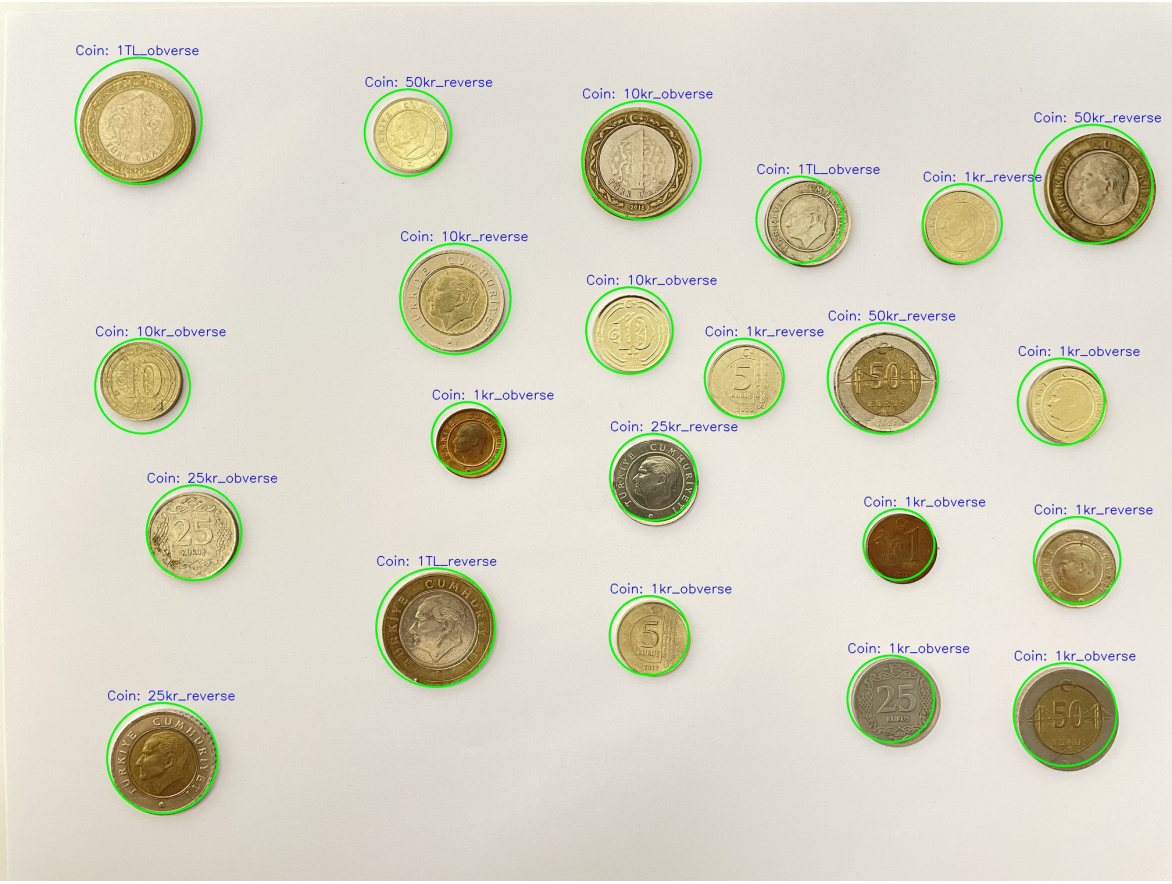

Figure 3: Results of Coin Recognition using HoG Features and SVM (Challenging Scenario Test V)

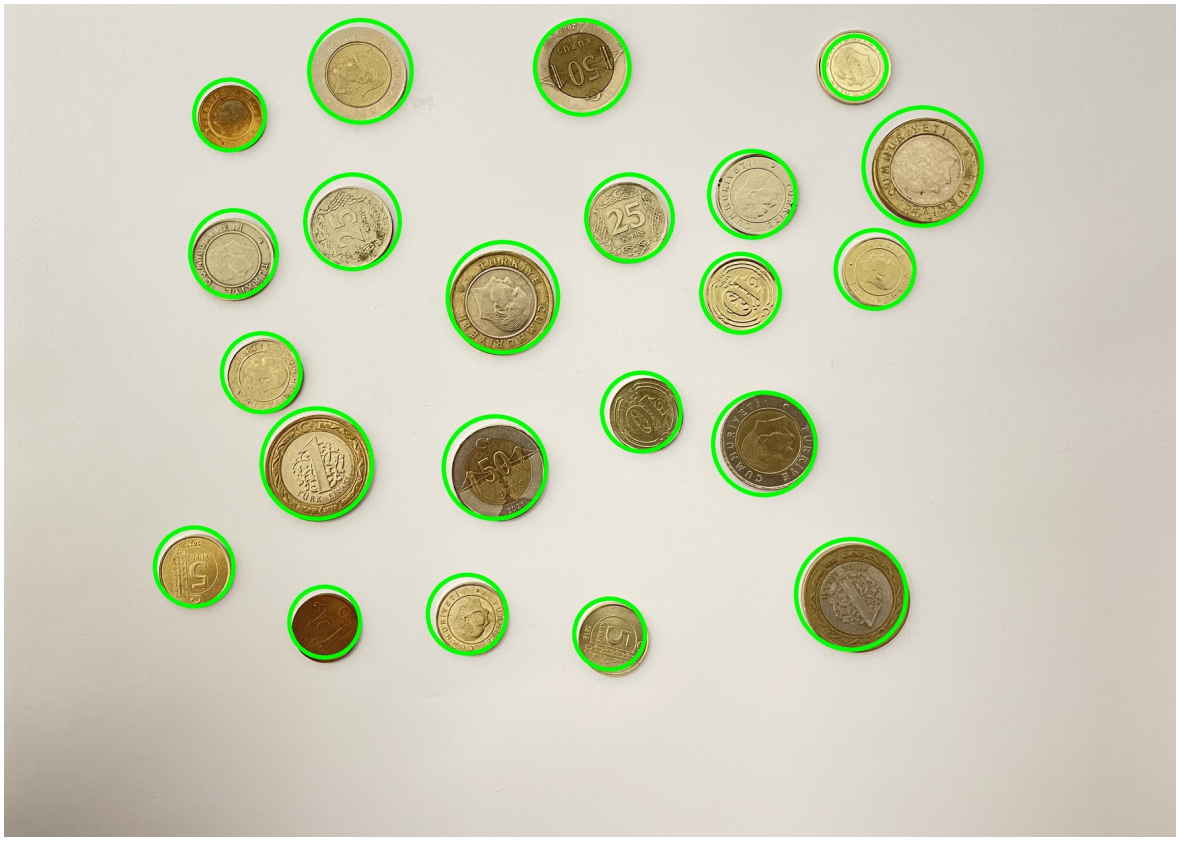

Figure 4: Results of Coin Recognition using HoG Features and SVM (Bonus: Test R Folder)