Visual Recognition: Assignment - 1

Kandarp Dave
Dave.Kandarp@iiitb.ac.in
IMT2022115

1 Introduction

In this assignment, I worked with OpenCV libraries in Python to apply various computer vision techniques. The assignment was divided into two tasks, which are explained in the subsequent sections. Click here to view the input images, Python scripts, and the output images.

2 Part 1 : Edge Detection and Image Segmentation

2.1 Detect all coins in the image

In order to detect the coins and outline them, Canny edge detection was used. The steps involved are:

1. Convert RGB image to grayscale image.
2. Remove noise in the image by convolving the grayscale image with a suitable kernel. I used a 5x5 Gaussian kernel with a standard deviation (S.D.) of 1.5. The dimensions of the kernel and S.D. were decided empirically. Blurring the image too much resulted in the edges not being properly detected. Also, the algorithm detected many false edges if the blurring was not done. For example, edges were detected even in the interior of the coins.
3. After removing noise, the Canny edge detector was applied to obtain the edges. The advantages of using this algorithm are:
    - Low error rate.
    - Edge points are well localized. This means that the pixel marked as an edge by the algorithm is not too far from the pixel that is a true edge.
    - The detector only returns one point for each true edge point. The detector does not identify multiple edge pixels where only a single edge point exists.

4. For hysteresis thresholding performed by the Canny algorithm, a low threshold of 100 and a high threshold of 200 was chosen empirically. Changing the thresholds affected the continuity of the edges. Improper thresholds gave broken and false edges.

2.2 Segmentation of each coin

For region segmentation, the watershed algorithm was used. The steps involved in this are:

1. Obtain the binary image from the grayscale image. The thresholds were chosen such that the coins are highlighted as much as possible.
2. Apply morphological gradient processing for noise removal in the binary image. The operations applied are:
   (a) First, do dilation so that the bright regions (coins in this case) get expanded.
   (b) Calculate the distance of each white pixel in the binary image to the closest black pixel.
   (c) Now, we obtain the foreground area and the unknown area. The foreground area represents the region that is sure to be part of a coin. A lower threshold of *0.01 * dist.max()* was chosen empirically. On subtracting this region from the region obtained in step 1, the unknown area is obtained.
3. The connected components in the foreground area are marked as 1 while the unknown area pixels are marked as 0.
4. Using the original image and the connected components, the watershed algorithm is applied to highlight the regions which correspond to a coin.

One observation is that since the coins on the upper-left corner are close to each other, they are shown to be connected in the segmented image.

2.3 Count the total number of coins and isolate each coin

To count the total number of coins, we use the output of the Canny algorithm. The Canny algorithm returns a binary image in which the edges are highlighted in white, while the rest of the image is black. Thus, to count the total number of coins, we only need to know how many closed loops of white color are present in the binary image.

This can be found out using the *findContours* method. The input to this method is the output of the Canny algorithm. Additionally, we also provide the following parameters:

- Contour retrieval mode = $cv2.RETR\_EXTERNAL$ to retrieve only the outermost contours.
- Contour approximation method = $cv2.CHAIN\_APPROX\_SIMPLE$ to remove unnecessary points and compress the contours.

The algorithm correctly outputs 18 as the total number of coins.

To isolate individual coins from the image, I iterated through each contour. Since each contour is the boundary of a coin, we need to obtain the position of the boundary in the image. This is done using the *boundingRect* function. It returns the following:

- (x, y) = Top left corner position of the bounding rectangle
- (w, h) = Width and height of the bounding rectangle, respectively.

Using this, we can extract the corresponding portion of the original image to obtain an individual coin.

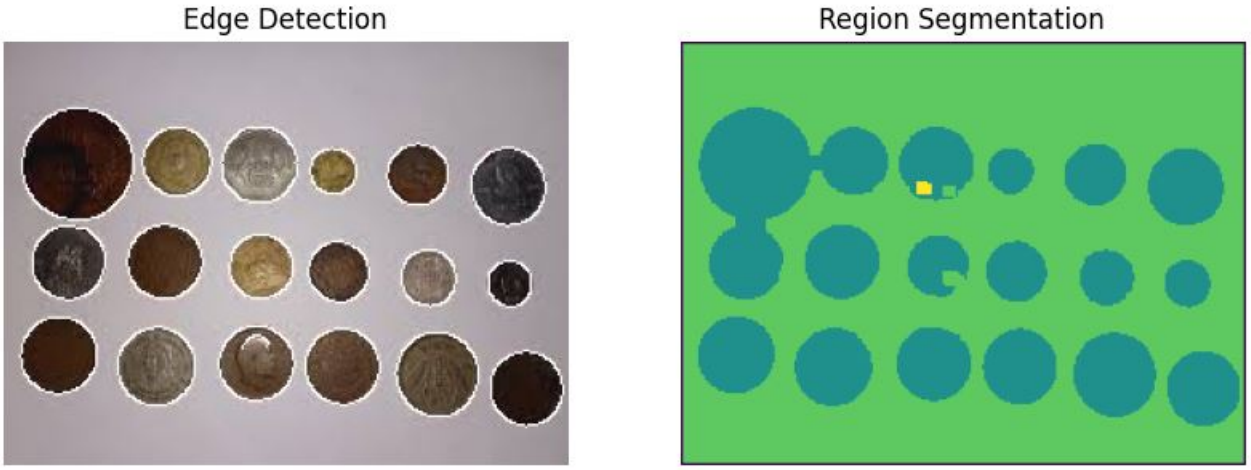

Figure 1: The first image shows the result of edge detection. The edges are shown in white. The second image shows the result of region segmentation using watershed algorithm.

Figure 4: Figure showing isolated coins (Not in the same order as in the original image)

3 Part 2 : Panorama Creation

To create a panorama, three different overlapping images were used. First, keypoints were identified in each of the three images. Then, these images were stitched together to create a panorama.

3.1 Extract key points

In order to stitch the images together, there must be some common keypoints in the images. In order to find these keypoints, I used SIFT. The steps involved in this are:

1. Convert the RGB image to grayscale.
2. The scale space is generated by creating multiple octaves and blurring the image in each octave using a Gaussian filter. This is done to detect features at different scales.
3. From the scale space, the difference of Gaussians is computed and extrema is found. These are the initial keypoints. The accuracy of these keypoints can be further improved by using interpolation and Taylor expansion.
4. Keypoints that have low contrast or those that are poorly localized are eliminated.
5. A histogram-based procedure is used to compute keypoint orientations.
6. For each keypoint, a 128-dimensional feature vector is generated using the histogram generated in the previous step.

(a) Keypoints in input image 1.

(b) Keypoints in input image 2.

(c) Keypoints in input image 3.

Figure 5: Figure showing the keypoints in each of the three input images.

3.2 Feature Matching

Using the keypoints and descriptors obtained in the previous step, I tried to match the feature descriptors between the adjacent images in the panorama. For this, the brute force matcher was used. The steps involved are:

1. For each descriptor in the first image, find the best matches in the second image.
2. Sort the matches obtained in step 1 based on their distance.
3. Show the best 20 matches. Other matches were not shown as they were not accurate.

3.3 Image stitching

The input images were stitched together by the *Stitcher* class of OpenCV. The *stitch* method of this class performs the following operations:

1. Detects keypoints in input images matches keypoints between overlapping images.

(a) Matching keypoints between images 1 and 2.

(b) Matching keypoints between images 2 and 3.

Figure 6: Figure showing the matching keypoints.

2. A homography matrix is calculated which aligns images based on their overlapping regions.
3. The images are then warped to align them in a common coordinate frame.
4. Finally, the overlapping areas are blended to remove visible seams.

Figure 7: Panorama obtained using the three overlapping images.