VR Report Assignment1

Prateek Rath

February 2025

Introduction

In this assignment, we learn basic image processing, segmentation techniques. Part2 involves implementing image stitching. Here, we will be analyzing the work done.

Part1

Canny Edge detection

First we convert the image to grayscale. After this the image is blurred and then binarized via Ostu's binarization. Finally, canny edge detector is applied to get edges. These edges are then used to generate contours.
Note that this method cannot be reliable to count coins that are really close to one another. In such cases the edge detector may not detect the edges between neighboring coins that are touching.

Region based Methods

Region based methods can be either top down or bottom up. Top down methods start with the entire image grouped into a single region and then the image is segmented to form different regions. The segmentation is usually based on the size and homogeneity of each cluster. Top down approaches usually require some domain or semantic knowledge about the image. Although we know that we are segmenting coins, it is unclear as to how to go about such an approach. Hence we focus on a bottom up approach in this assignment.
First, we use k means with k=2 to separate the foreground from the background. Assuming that most coins are similar in color, and different from the background, it works pretty well.
We might also be able to differentiate coins from one another by also using position instead of only r,g,b attributes. However, it seems uncertain as areas between coins also have boundaries. Moreover the elbow method gives an optimal value of k=2 for the image.
Next, we use the watershed algorithm to get segmented outputs for each coin.

The goal is to assign each pixel to either the background label or a specific coin label. The philosophy behind the watershed algorithm comes from an analogy to rain flooding a series of valleys. As rain fills the valleys, the cells in the valleys are grouped together and the differences remain at peak points.
The steps used are as follows:

1. Get the image to grayscale
2. Use Ostu's binarization to get the binary image
3. Perform morphological opening to remove noise(erosion + dialation)
4. Compute the sure background via dialation
5. Compute the distance transform which is the distance of every point from the nearest black point
6. Obtain the sure foreground
7. Get the unknown area by subtracting the sure background and the sure foreground
8. Obtain initial markers by finding connected components in the sure foreground
9. The labels are the unique markers
10. Now, for every point in the unknown area, a call to cv2.watershed() either gives it a label or considers it to be the background
11. Obtain contours for each region corresponding to a certain region

Counting Number of Coins

Here we can follow two approaches.
One is to count number of coins after drawing contours from the output of the Canny Edge Detector. The problem with this is that canny edges strongly rely on the extent of blurring(sigma) and the kernel size used. For some sizes coins are detected while others count artifacts also. If coins are touching and too much blurring is applied, they may get counted as a single coin.
The other method is also to count contours but in the watershed algorithm. This works best with a standard 3*3 morphological kernel. It counts the coins correctly for both 'coins.jpeg' and 'coins2.jpeg' unlike canny which makes a mistake in the second image.

Part 2

We follow the standard image stitching algorithm which has the following steps:

1. Call sift on the first and second image and obtain key points and descriptors. Note that in the code first and second are referred to as left and right.

2. Using a brute force matcher, obtain the matches between the images. Brute force is better than all tree like matchers, it is just slower. We stick with it to get the best results possible.

3. Compute the homography from the first image to the second image. Modify the homography to include a translation to preven the image from going out of bounds.

4. Warp the first image to the coordinate frame of the second image. Do this by computing new locations for the four corners. See the code for more clarity.

5. Translate the second image appropriately for merging with the first one.

6. Loop over every pixel in the right image. If the pixel isn't present in one of the input images take the other. Otherwise average the pixel values.

7. The stitched image is ready.

We see a few black lines due to averaging the images. This could be improved via taking a smoother weighted averaging. Also note that the code takes a little time to run, so be patient.