Mid-term Report

Perception for Autonomous Robots

Instructors:
DR. SAMER CHARIFA

Student:
RISHABH SINGH;
rsingh24@umd.edu

Semester:
SPRING 2022

Course code:
ENPM673

Contents

List of Figures

1 Introduction

This report is for the mid-term for the session of Spring 2022.

2 Problem 1

2.1 Seperating the coin using the concept of Morphology

Problem Description: Assuming the image 1 represents an x-ray of old coins. Read the image seen below using OpenCV taking into consideration that this is a binary image.

1. Write a program that will separate each coin from each other, so that the output image has the coins completely separated.

2. Write a program that will automatically count how many coins are there in the image (from step 1) . Note: You are allowed to use any built-in function in OpenCV. However, you are required to explain each step you took in your solution (Use pipeline or a block diagram).

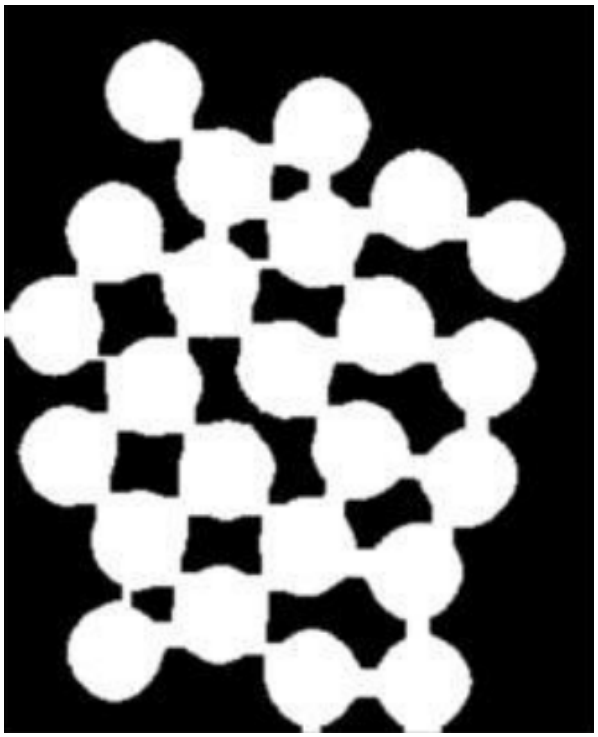

Figure 1: Provided Image

Pipeline followed:

Step 1: Import the image in default format

Step 2: Converting the image into gray scale using cv2.cvtColor and then converting it into binary using cv2.threshold. This is done to assure that the image is binary.

Step 3: The next step is to apply morphology but before that the things need to be decided are the type of morphology and the type of kernel to be used. Since, we just have to separate the coins without eroding the coins much, opening appeared to be the best method for this task. An opening is an erosion followed by a dilation. Performing an opening operation allows us to remove small blobs from an image: first an erosion is applied to remove the small blobs, then a dilation is applied to regrow the size of the original object.

Step 4: It's obvious in image processign that the filter and the kernel should be alike what we want as output from the processed image. The suitable inbuilt kernel with same shaoe that I found was using cv2.getStructuringElement(cv2.MORPHELLIPSE,(40,40)). The kernel size has to be tuned to get the correct output as shown in **??**.

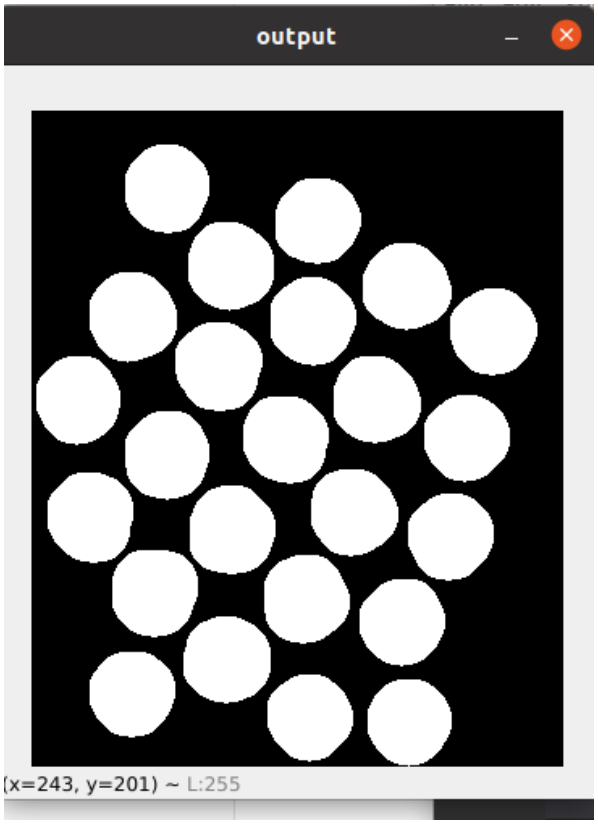

Figure 2: Seperated Coins

Step 5: Next step was to count number of coins in the image which was done by counting the number of connected components in the image which can be done using the inbuilt function cv2.connectedComponents() method. This checks the image with 4 or 8 way connectivity - returns N, the total number of labels [0, N-1] where 0 represents the background label. The output was 25 including the background label. Hence, the total coins found was found by subtracting 1 from the total, giving out 24 coins in the image.

(a) Left Image

(b) Right Image

Figure 3: Provided set of images

3 Problem 2

3.1 Seperating the coin using the concept of Morphology

Problem Description: Given two images, 3 shown below, find the matching points between these two images and stitch them together by finding the homography between them. Write the pipeline that you used to solve this problem and attach the code solution.

Pipeline followed:

Step 1: Since, SIFT is no more available in the latest opencv, I had to create a custom environment to do this problem. OpenCV version 3.4.2 was installed. (without realising that ORB can be used)

Step 2: Importing the images using cv2.imread() method and then converting it to RGB format for plotting it. Since, cv2.imshow() is not compatible with this version of openCV, hence pyplot was utilized.

Step 3: Performing pre-processign operations like using the grayscale image, and gaussian blur to decrease the noises in the images.

Step 4: Create an object using cv2.xfeatures2d.SIFT˙create() class and hen finding keypoints and descriptors in both the images using sift.detectAndCompute() method.

Step 5: Here, keypoints are also objects, it was required to take the pt attribute out of this class which was used to get source and destination points later in the program after selecting the ids of good descriptors.

Step 6: This step is known as the registratio step, where we match the featurs in the two image. Matching features together is actually a fairly straightforward process. We simply loop over the descriptors from both images, compute the distances, and find the smallest distance for each pair of descriptors. Since this is a very common practice in computer vision, OpenCV has a built-in func-

tion called cv2.DescriptorMatcher˙create that constructs the feature matcher for us. The BruteForce value indicates that we are going to exhaustively compute the Euclidean distance between all feature vectors from both images and find the pairs of descriptors that have the smallest distance using the lowe's ratio and stored in a list.

Step 7: The minimum good matches required are 4 for performing homography but I kept it to 10 for best results. Now using the descriptors of the left image (destination), its queryid and trainid were used to take the best keypoints selected from both left and right image, and stored as destination points and source points.

Step 8: Now using these points and RANSAC, the homography is performed with 4 best inliers using cv2.findHomography() method.

Step 9: The final part is to warp the right image to the a new frame in order to perform stitching. A new frame using the widht of both the images and height of either of the image is created and the right image is warped on to this frame using cv2.warpPerspective() method.

Step 10: Now on the left part of the new frame the left image is copied as is to get the sticthed image as in 4.

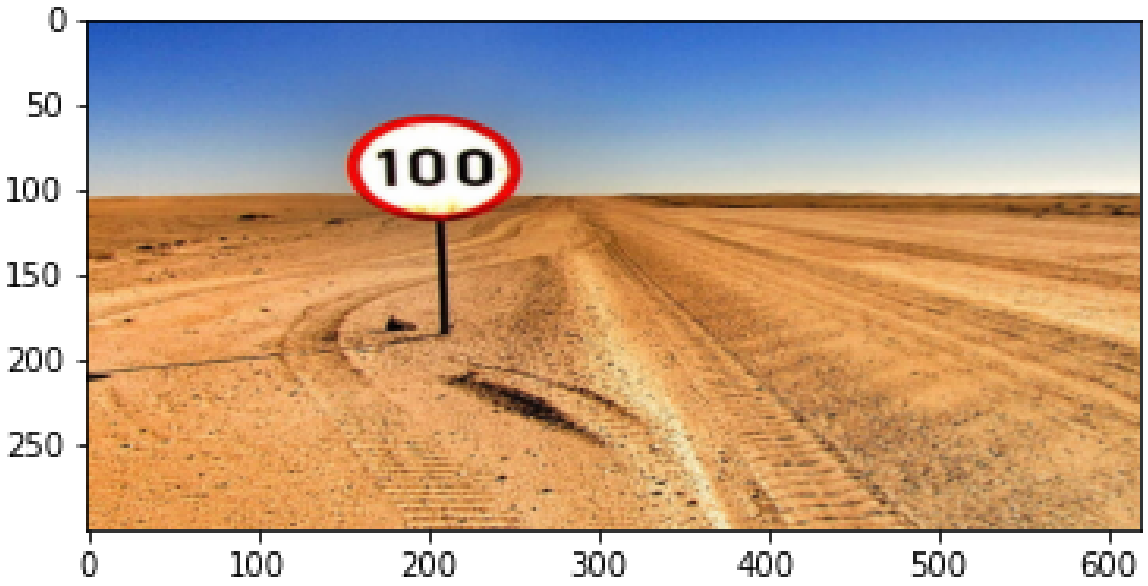

Figure 4: Stitched Image

4 Problem 3

4.1 Camera Calibration

Part 1: The minimum number of matching points required are 6. The reason is the projection matrix is a $4X3$ matrix which has 12 unknown elements. Each matching points gives two equations, hence to solve for 12 unknown variables 6 matching points are required.

Part 2 and 3: Pipeline and mathematics Required for camera calibration

Step 1: Using a known object as shown below 5, in which all the 3D dimensions are known in the world coordinate systems and can be matched with the 2D coordinates of image sensor coordinate system, the required number (minimum 6) of matching points are recorded after defining the corresponding origins and units (pixel in image frame and mm in world frame).

Figure 5: Board for camera calibration

Step 2: Now, using homogenous coordinates we know that the points in the world frame can be matched to the camera plane using the extrinsic matrix of size $4X3$ as shown below:

$$\tilde{\mathbf{x}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Figure 6: Mapping of camera coordinates to world coordinates

Step 3: Now projection matrix, $P = K[R|T]$, where K is the camera instrinsic matrix, R is the Rotation Matrix and T is the translational vector. The $[R|T]$ is known as extrinsic matrix as shown above:

Step 4: Extrinsic matrix is utilized to map the world coordinate to the camera 3D coordinates (pin hole), when multiplied by camera intrinsic matrix this can be used to form projection matrix as shown below 8:

$$\text{Extrinsic Matrix: } M_{ext} = \begin{bmatrix} R_{3\times3} & \mathbf{t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 7: Extrinsic Matrix

Camera to Pixel

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

World to Camera

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Figure 8: Mapping of camera coordinates to pixel/sensor coordinates

Step 5: Now $\tilde{U} = M_{int} * M_{ext} * \tilde{x}_w = P * \tilde{x}_w$, where P is the projection matrix.

Step 6: Now, a new $4X3$ matrix is generated with 12 unknown variables as shown below 15:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Figure 9: Projection Matrix

Step 7: Now using the recorded matching points the projection matrix is calculated by expanding the linear equations as shown below 10:

$$u^{(i)} = \frac{p_{11}x_w^{(i)} + p_{12}y_w^{(i)} + p_{13}z_w^{(i)} + p_{14}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

$$v^{(i)} = \frac{p_{21}x_w^{(i)} + p_{22}y_w^{(i)} + p_{23}z_w^{(i)} + p_{24}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

Figure 10: Linear equations

Step 8: The projection matrix is reshaped as $12X1$ matrix and after re-arranging, all the points are used and to form a $8X12$ matrix A, to form a homogenous equation as shown 11:

Step 9: Now this homogenous equation can be solved using either the SVD or else by minimising the least square constraint, which says $min(P^T * A^T * A * p)$ such that $p^T * P = 1$, which gives

$$\begin{bmatrix} x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1x_w^{(1)} & -u_1y_w^{(1)} & -u_1z_w^{(1)} & -u_1 \\ 0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1x_w^{(1)} & -v_1y_w^{(1)} & -v_1z_w^{(1)} & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u_ix_w^{(i)} & -u_iy_w^{(i)} & -u_iz_w^{(i)} & -u_i \\ 0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & -v_ix_w^{(i)} & -v_iy_w^{(i)} & -v_iz_w^{(i)} & -v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_nx_w^{(n)} & -u_ny_w^{(n)} & -u_nz_w^{(n)} & -u_n \\ 0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_nx_w^{(n)} & -v_ny_w^{(n)} & -v_nz_w^{(n)} & -v_n \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 11: Homogenous Equation

out a loss function as $L(p, \lambda) = p^T * A^T * A * p - \lambda(p^T * p - 1)$.

Now on minimizing this loss function w.r.t to p we get the equation $2 * A^T * A * p$ - $2 * \lambda * P = 0$. Which says that the value of p is the eigen vector corresponding to the minimum eigen value of $A^T * A$.

Step 10: Now, after finding all the 12 parameters of projection matrix, we need to find the camera calibration matrix which is the instrinsic matrix. Now we know that camera instrinsic matrix is an upper triangular matrix and rotation matrix is the orthonormal matrix. In this case, these matrices can be find out using the QR factorization. The matrix that needs to be considered out of projection matrix to find intrinsic and rotation matrix is the first 3 columns of projection matrix as shown below 12:. So, QR factorization of this matrix using the numpy can give us Q which is the rotation matrix and R will give the camera instrinsic matrix, which needs to be normalized by dividing the whole matrix by last element. Now, all the 5 parameters of camera calibration matrix are found out. Rest of the details are explained in code.

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = KR$$

Figure 12: Rotation and Intrinsic Matrix

Output:

```
proejction matrix un-normalized
 [[ 3.6223365863e-02 -2.2152108044e-03 -8.8324291540e-02  9.5408888146e-01]
 [-2.5383318934e-02  8.3055570388e-02 -2.8001630884e-02  2.6882701254e-01]
 [-3.4922232238e-05 -3.2718480857e-06 -3.9566760551e-05  1.2605374980e-03]]
```

Figure 13: Un-normalized Projection Matrix

```
rotation matrix
 [[-8.1894518264e-01 -5.7387086228e-01  1.0105719602e-03]
 [ 5.7387120896e-01 -8.1894556054e-01  6.6346860384e-05]
 [ 7.8952889047e-04  6.3427259426e-04  9.9999948717e-01]]
```

Figure 14: Rotation Matrix

```
calibration matrix
 [[ 338.4668660605 -378.6068635272 -430.5346424518]
 [  -0.            510.7546157805 -563.3383191868]
 [  -0.             -0.              1.          ]]
```

Figure 15: Normalized Calibration Matrix

5 Problem 4

5.1 K-mean clustering for colour segmentation

Problem Description: Implement K-means algorithm to separate the image below 17, based on color, into 4 classes. Note: You are NOT allowed to use any built-in function, implement your code from scratch. Explain each step you take.

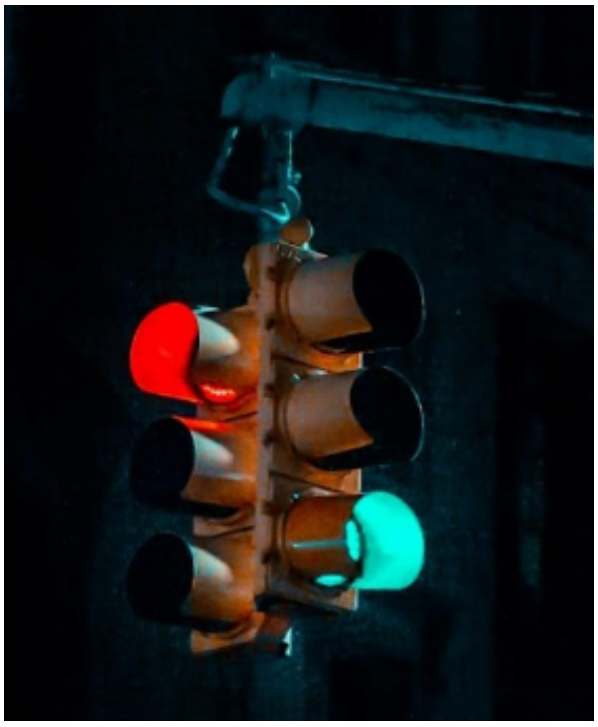

Figure 16: Input Image for Colour Segmentation

Pipeline followed:

Step 1: Read the image in default format and flatten the image for easy manipulations.

Step 2: Define the number of clusters or segments you want to create. This is 4 in our case.

Step 3: Assume the first centroid with either 4X3 random array of RGB values or create thm manally. I started with random but the results were changing each time, which could take time to converge also. Hence, I have created an input centroid manually.

Step 4: Defining the k-mean clustering algorithm as shown below in simple flowchart:

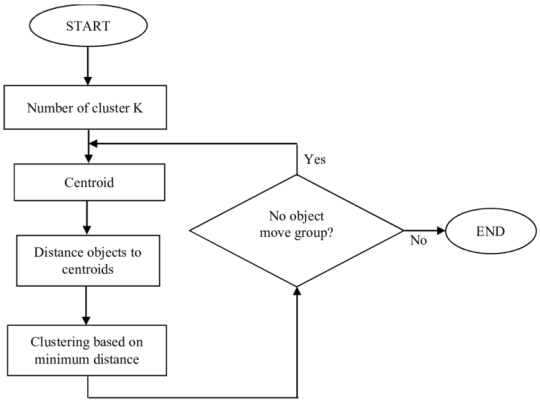

Figure 17: Flow chart for k-mean (1)

Step 5: An array of the same size as the flattened image is created which stores the cluster or the segment number. This is updated each time based on the new centroid calculated.

Step 6: A method to generate new centroids is created which counts the number of pixel in each segment, and then average their RGB values and creates a new centroid of size 4X3, which is again recursvely passed to the k-mean method as defined above.

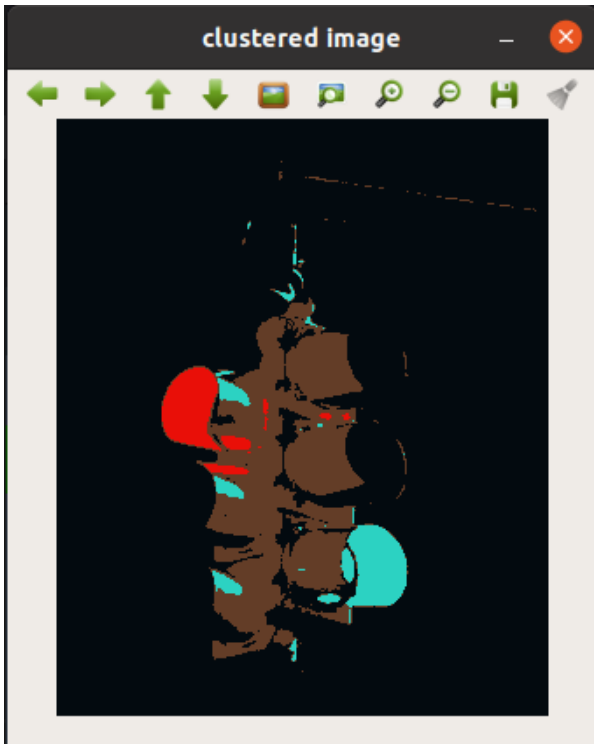

Figure 18: Colour Segmented Image

Step 7: In each iteration the centroids are compared and if the values are not changed, the algorithm stops considering convergence. Maximum iterations are also considered to stop the algorithm from infinite or long loops to decrease the time for output. The output is as shown in 18.

6 Bibliography

[1] Alhadi Bustamam, Hengki Tasman, N. Yuniarti, Frisca, and Ichsani Mursidah. Application of k-means clustering algorithm in grouping the dna sequences of hepatitis b virus (hbv). volume 1862, page 030134, 07 2017.