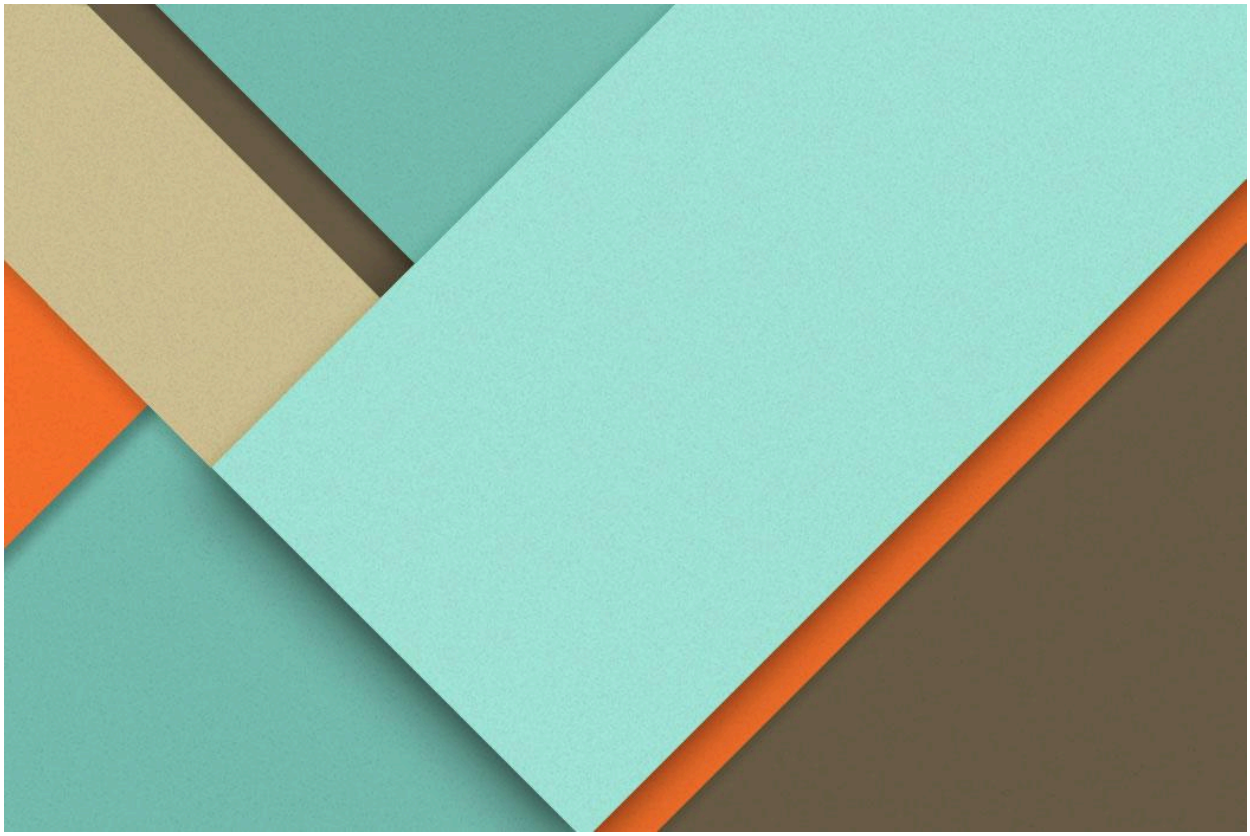


Visual Recognition Assignment-1

02.03.2025

—

Ishan Jha
IMT2022562
IIT Bangalore
Visual Recognition

GitHub Repository Link:

https://github.com/Ishaniiitb/VR_Assignment1_Ishan_IMT2022562

Question-1

The aim of this question is to use computer vision techniques to detect, segment, and count coins from an image containing scattered Indian coins.

Dataset

The images used as a dataset for this task are stored under the folder “CoinInputs” in the GitHub repository whose link is given above. The following two images are the dataset:

One of the images only contains coins, while the second image contains certain other objects as well to test if the code is able to distinguish between coins and other objects.

Edge Detection

In order to perform edge detection in the image the following different techniques were implemented:

Sobel Kernels:

The ***Sobel operator*** was executed using the cv2.Sobel function. A kernel size of 3 was selected to enhance edges while keeping noise at a manageable level. It was applied separately along the X and Y axis, after which the gradient magnitude was computed. However due to the presence of a large number of false edges due to large noise margins, we were getting more than the actual number of coins. In addition to the above problem,

due to noise margins being large, it kept detecting the key (in the second image in the dataset) as a coin.

Laplacian Operator:

The ***Laplacian operator*** was executed via cv2.Laplacian function of the OpenCV package of python. A kernel size of 3 yielded distinct, sharp edges while also accentuating unwanted background textures, whereas increasing the kernel size to 5 resulted in smoother edges but caused significant blurring and loss of finer details. Ultimately, the Laplacian method, since it involves the second derivative of a function, proved inefficient due to its high sensitivity to noise.

Canny Edge Detection:

Finally, the ***Canny Edge Detector*** (cv2.Canny) was implemented, which provided the best results. Firstly we smooth the edges in the image using a Gaussian Blur function in OpenCV(cv2.GaussianBlur()) with the kernel size being 5x5. In contrast to previous above mentioned techniques, the Canny algorithm produces clearly defined edges with minimal noise, making it the favored option for subsequent processing.

Segmentation and Counting:

After edge detection, a kernel of size 3x3 is created with all its entries being '1'. This kernel will now be used for ***morphological transformations***. A **morphological closing operation** is a common image processing technique used to clean up binary images (images with only black and white pixels, typically resulting from edge detection or thresholding). In short, morphological transformations are used for:

1. Closing small holes or gaps inside objects,
2. Connect broken lines or edges.
3. Smooth out object boundaries.

In contour detection or object recognition, small gaps in edges can cause the algorithm to detect multiple disconnected shapes instead of one complete object, thus morphological transformation is an important step.

After segmentation, the regions were processed using contour detection (cv2.findContours) to isolate each coin's boundaries. The code then applied area constraints to filter out any unwanted small objects. Furthermore, a circularity verification step was applied using the formula $(4\pi \times \text{Area}) / (\text{Perimeter}^2)$ to ensure that only circular objects (coins) were selected. This measure effectively removed false positives, such as reflections or objects that were not coins. ***This step ensured that the box of cards in the second input image was not classified as a coin***.

Images:

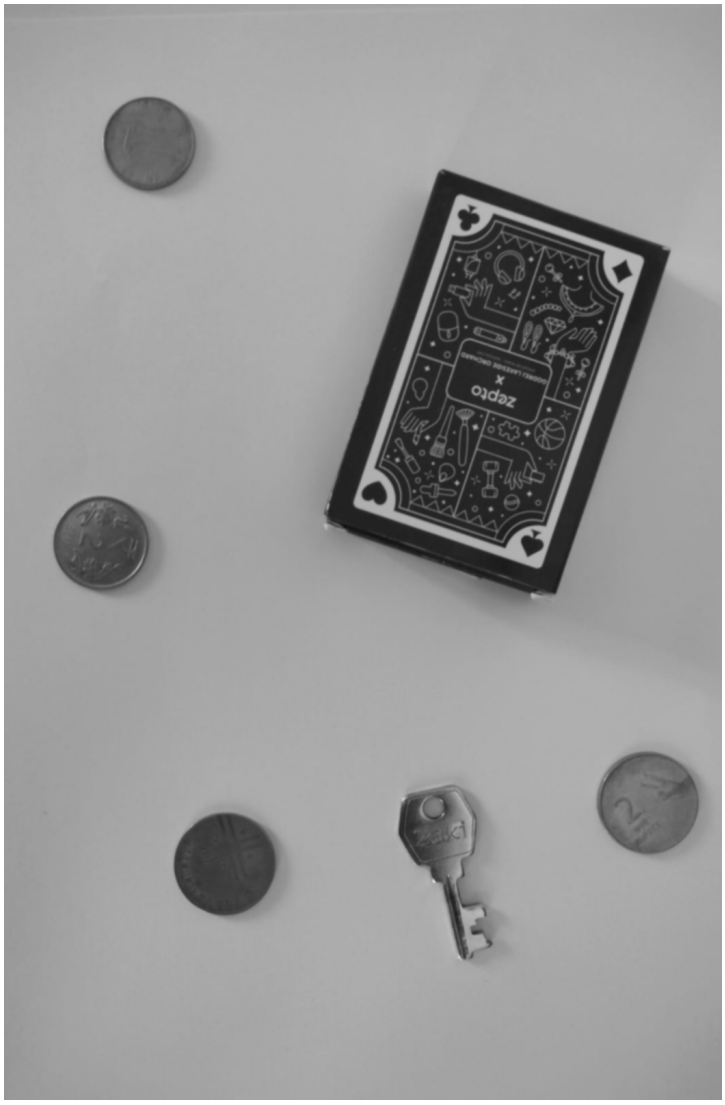


Image obtained after converting the image to grayscale and applying Gaussian Blur to it.

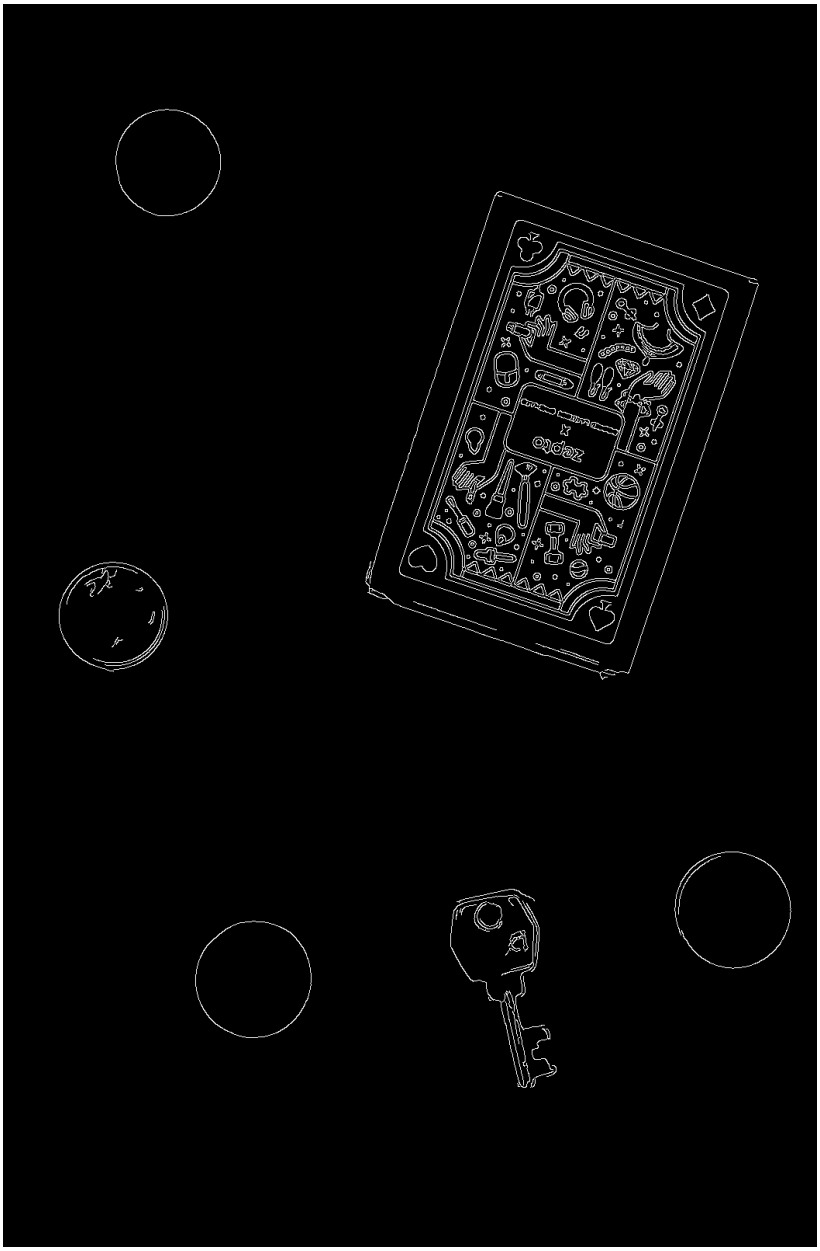


Image obtained after performing Canny Edge Detection

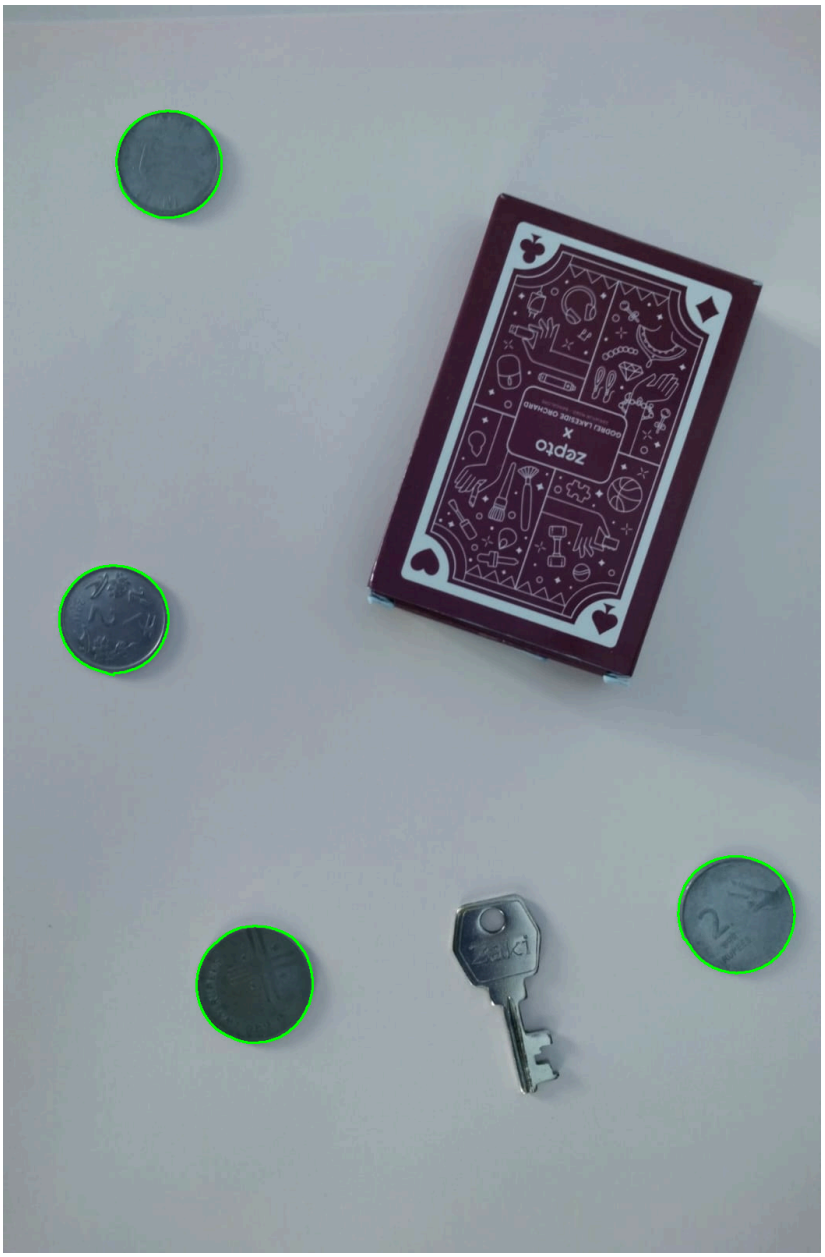


Image Segmentation to detect and segment all the coins

Coins after image segmentation and displaying total number of coins(4 in this case)

Question-2

In the second question of the assignment we were supposed to create a stitched panorama from multiple overlapping images. In addition to that we were supposed to detect key points in overlapping images and display those keypoints.

Dataset

The images used as a dataset for this task are stored under the folder “PanoInputs1” and “PanoInputs2” in the GitHub repository whose link is given above. The following images are the dataset (PanoInputs2):

The above 6 images are present in a folder named “PanoInputs2” in the GitHub repository.

Key Point Detection:

Detecting key points is essential for image stitching as it enables the identification of unique, repeatable features within overlapping images. In this implementation, the ***Scale-Invariant Feature Transform (SIFT)*** algorithm extracts these key points from the input images. The reason why SIFT transformation is used because the outputs of this algorithm remain constant despite changes in the image with regards to:

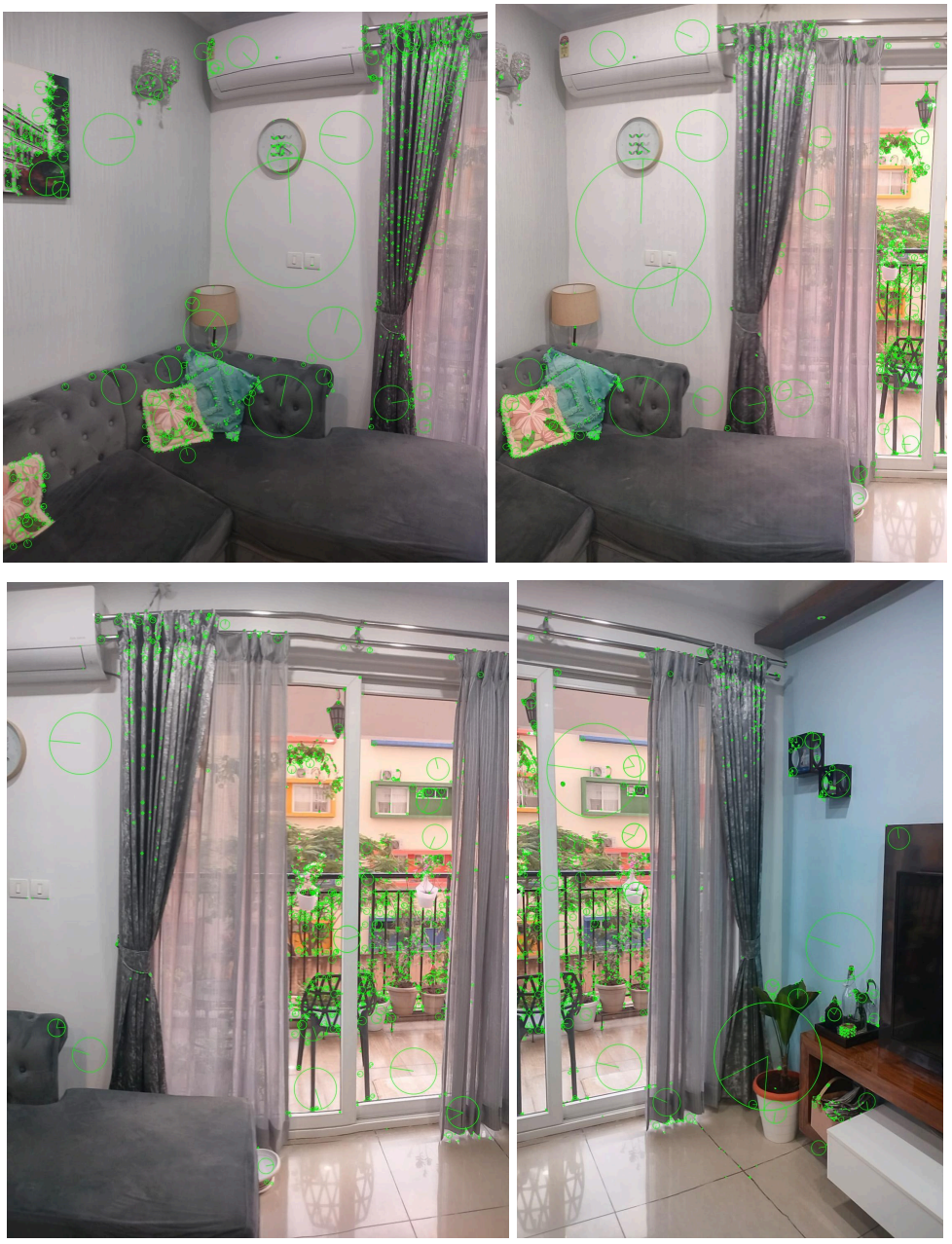
1. Scale of Image
2. Rotation of Image
3. Image Illumination

The SIFT detector begins by converting the image to grayscale before applying a ***difference of gaussians*** operation to identify potential key points. These key points are then refined by discarding weak features based on a contrast threshold. Each detected key point is assigned an orientation derived from local image gradients, ensuring rotation invariance. Lastly, a ***128-dimensional descriptor*** is generated for each key point, capturing local texture details and enhancing robustness for image matching.

The number of keypoints per image is set by us with the help of the "***nfeatures"*** parameters of the ***cv2.SIFT_create()*** function of OpenCV. After extracting the key points and descriptors, the ***Brute Force Matcher (BFMatcher)*** is employed to compare the descriptors between successive images with respect to the ***L2 norm***. After combining multiple images, the warping process may introduce unwanted black borders. To eliminate these, a contour detection technique is used to identify the region of interest, and the final stitched image is then cropped accordingly.

Images:

The following are the input images along with their keypoints.



The following is one example of keypoint matching which is used while stitching images

The final output after removing black spaces