VR Assignment 1

Soumik Pal (MT2024153)

README :

As per the assignment given, the repository contains 2 Python scripts, coin_detection.py & panorama_stitching.py for detecting coins in an image and stitching together overlapping images to create a panorama.

The scripts were developed using Python 3.12.6 inside a virtual environment. The repository contains both the input images that were used to develop & test these scripts, as well as subdirectories storing the output images obtained from running these scripts.

Modules needed to run the code :

- Numpy (Version 2.2.3)
- Opencv-python (Version 4.11.0.86)

Steps To Run the Scripts:

- Download the repository on your local machine
- Ensure Python 3 and the required modules listed above are installed with their correct versions.
- Ensure the folder containing the Python 3 Executable is added to the PATH environment variable.
- Open a terminal in the downloaded repository folder and in the terminal, run the commands :
  <path to the Python Executable> coin_detection.py
  <path to the Python Executable> panorama_stitching.py

Strategies I chose to tackle the assignment questions:

1. Coin Edge Detection & Segmentation :

    a. For Edge Detection, I first converted the image to grayscale and then I smoothed the image with a 7x7 Gaussian kernel.
    b. Then I applied a 3x3 Sobel filter, a 3x3 Laplacian filter and a Cammy edge detector with threshold values 100 & 200, to detect the edges and compare their edge maps.
    c. I highlighted the edges obtained by each method by adding them to the original image
    d. For region based segmentation, I chose to threshold the image on a fixed global threshold.
    e. Then I determined all closed continuous contours on the thresholding output binary image.
    f. Then iterating over all the contours, discarding the contours corresponding to noise, by checking that the contourArea is sufficiently big, I created a mask for each contour marking the region of interest inside the contour with pixel values of 255. Elsewhere the mask is all 0.
    g. Finally, I performed bitwise AND operation on the grayscale image with itself but with the masks for each contour. The masks ensured that only within the region of interest, that is within the corresponding coin boundary, the grayscale pixel values of the input image were kept intact. Outside the pixels were all 0, thereby isolating each coin.

2. Panorama Stitching :

    a. I chose to use SIFT features to describe each of the 6 input images in terms of keypoints and corresponding 128 dimensional descriptor vectors and then perform matching based on these descriptor feature vectors to be able to stitch them into a panorama.
    b. The SIFT detector object detected all the keypoints and descriptor SIFT feature vectors for each of the images.
    c. Then I used a Stitcher object to do the stitching. It internally matches the descriptors based on similarity measures like L2 distance, to stitch the images together and finally outputs the panorama image.

OpenCV Methods Used:

1. coin_detection.py :

    a. cv2.imread() - To load the input image file.
    b. cv2.resize() - To resize an image to a specific size.
    c. cv2.cvtColor() - To convert RGB Input image to grayscale.
    d. cv2.GaussianBlur() - To perform image smoothing before edge detection.
    e. cv2.Sobel() - To apply 3x3 Sobel filters on the blurred image
    f. cv2.magnitude() - To get the magnitude from the gradients along the. x and y directions.
    g. cv2.imshow() - To display an image.
    h. cv2.waitKey() - To keep the window, created by the imshow() method, open till the time configured expires or a key is pressed.
    i. cv2.imwrite() - To save an image to the specified path.
    j. cv2.add() - To add 2 images, i.e. edge maps to the input image to highlight edges.
    k. cv2.Laplacian() - To apply a Laplacian kernel of size 3x3 on the input image.
    l. cv2.Canny() - To apply Canny edge detection algorithm on the input image.
    m. cv2.destroyAllWindows() - To remove all currently opened image display windows.
    n. cv2.threshold() - To perform image thresholding on the input image and produce a binary output image
    o. cv2.findContours() - To find contours on the binary image.
    p. cv2.contourArea() - To calculate areas of found contours. Used in this script to remove noise contours and only work on the contours of the coins.
    q. cv2.drawContours() - In our script, for every contour, this function draws the contour with a pixel value of 255 on a mask of the same size as the input image, which is all 0’s elsewhere. The flag thickness=cv2.FILLED is used to ensure all pixels of the mask inside the contours are also having the value 255, which marks the region of interest.

r. cv2.bitwise_and() - Performs bitwise AND of 2 images, only in the region of interest specified by the mask. Elsewhere the pixel value is considered 0. In our script, we have used to perform bitwise AND of the grayscale input image with itself but with the masks computed by drawContours(). This ensures for each mask, only the respective coin is isolated.

2. panorama_stitching.py :

a. cv2.SIFT_create() - To create a SIFT (Scale-Invariant Feature Transform) feature detector and descriptor extractor.
b. cv2.imread() - To load the input image files.
c. sift_obj.detectAndCompute() - To detect keypoints in the input images and compute their corresponding descriptor vectors.
d. cv2.drawKeypoints() - To draw the keypoints detected by SIFT feature detector onto the input images.
e. cv2.resize() - To resize an image to a specific size.
f. cv2.imshow() - To display an image.
g. cv2.waitKey() - To keep the window, created by the imshow() method, open till the time configured expires or a key is pressed.
h. cv2.imwrite() - To save an image to the specified path.
i. cv2.destroyAllWindows() - To remove all currently opened image display windows.
j. cv2.Stitcher_create() - To create a stitcher object, which is used to perform image stitching (panorama creation).
k. stitcher.stitch() - To perform the stitching process, using the SIFT keypoints and descriptors by matching the keypoints across the input images, then aligning them and blending them together to create a panorama output image.

Input Images Used :

1. coin_detection.py :

2. panorama_stitching.py :

Image 1

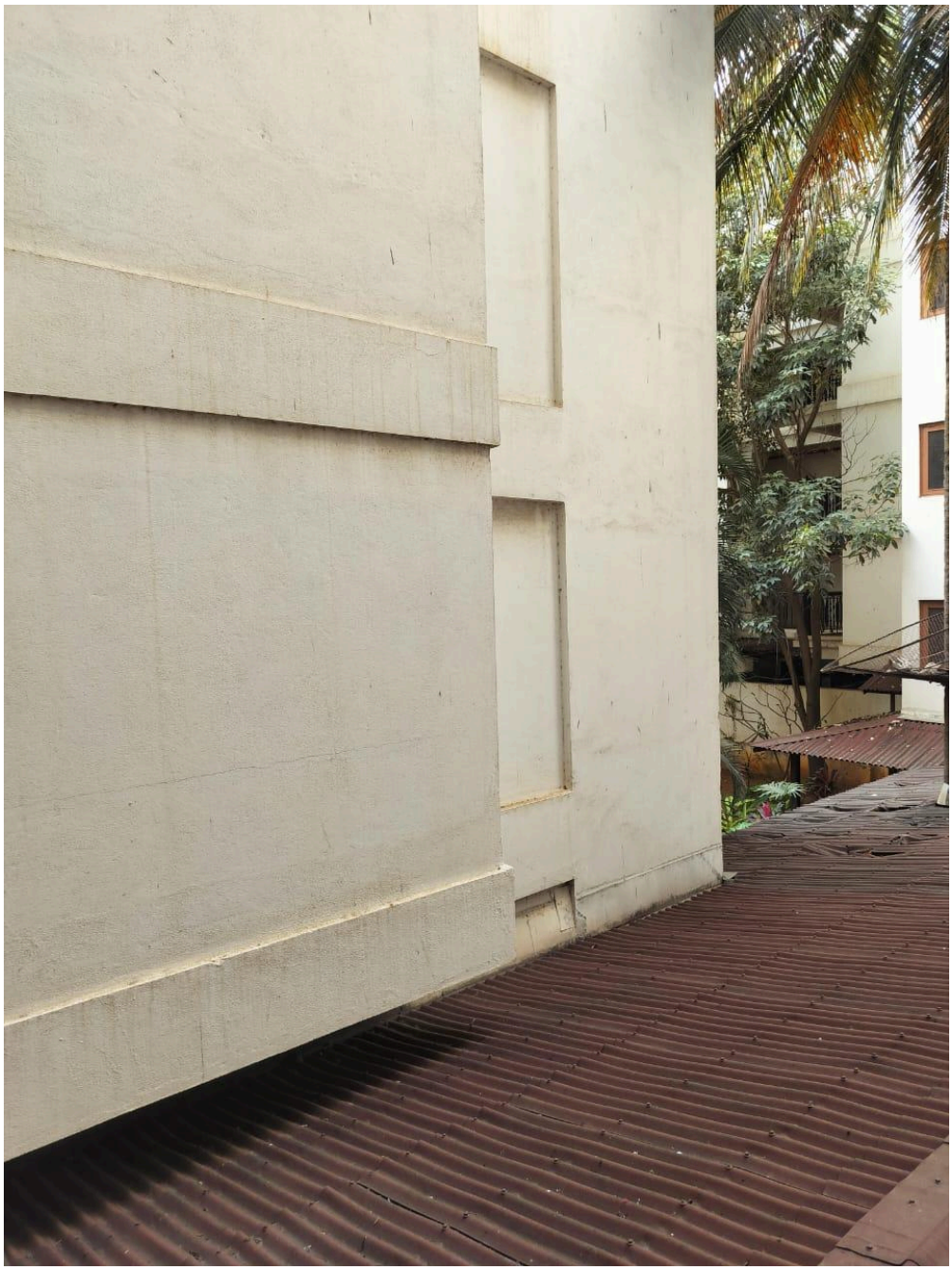


Image 2 :

Image 3 :

Image 4 :

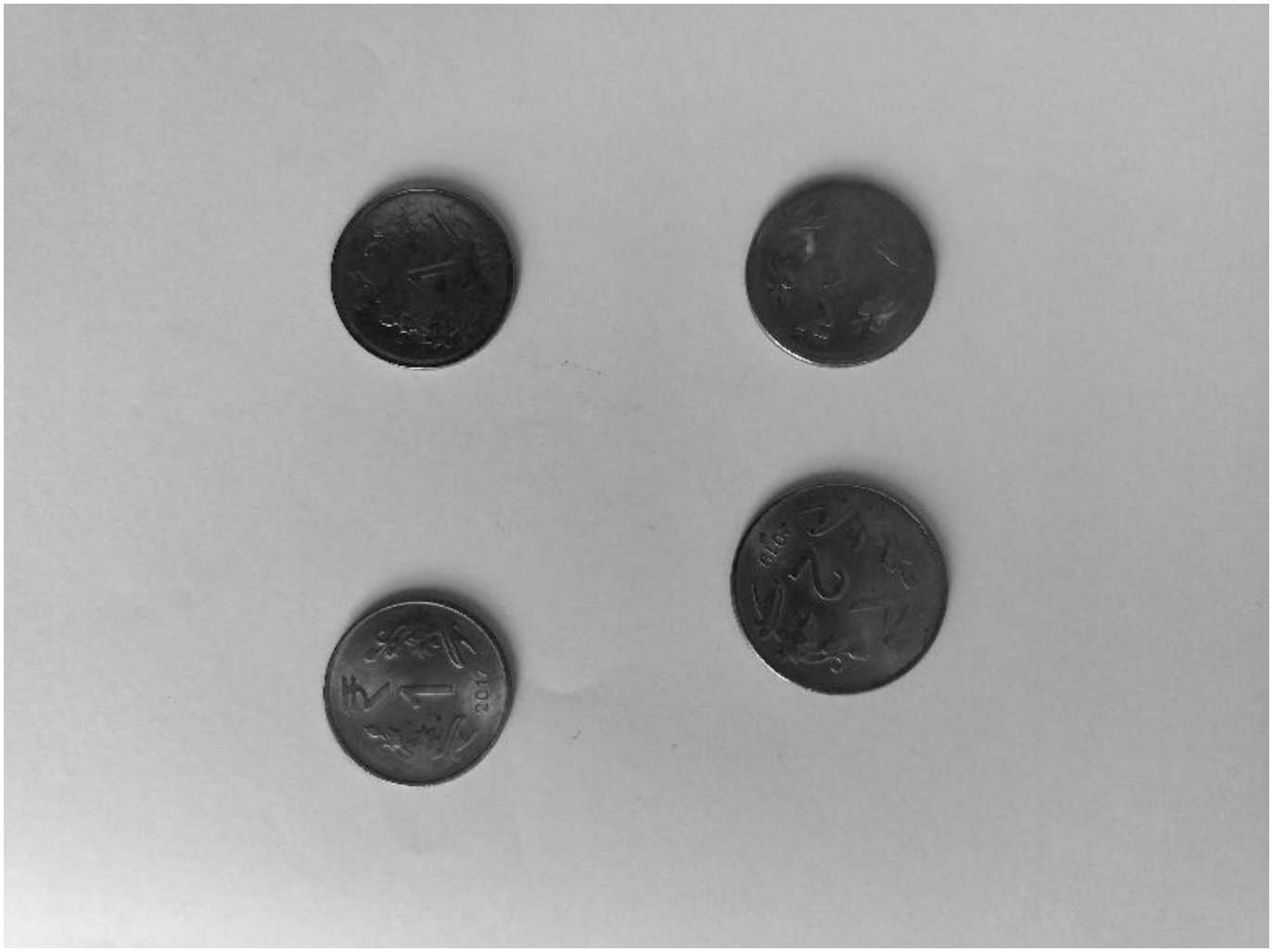
Image 5 :

Image 6 :

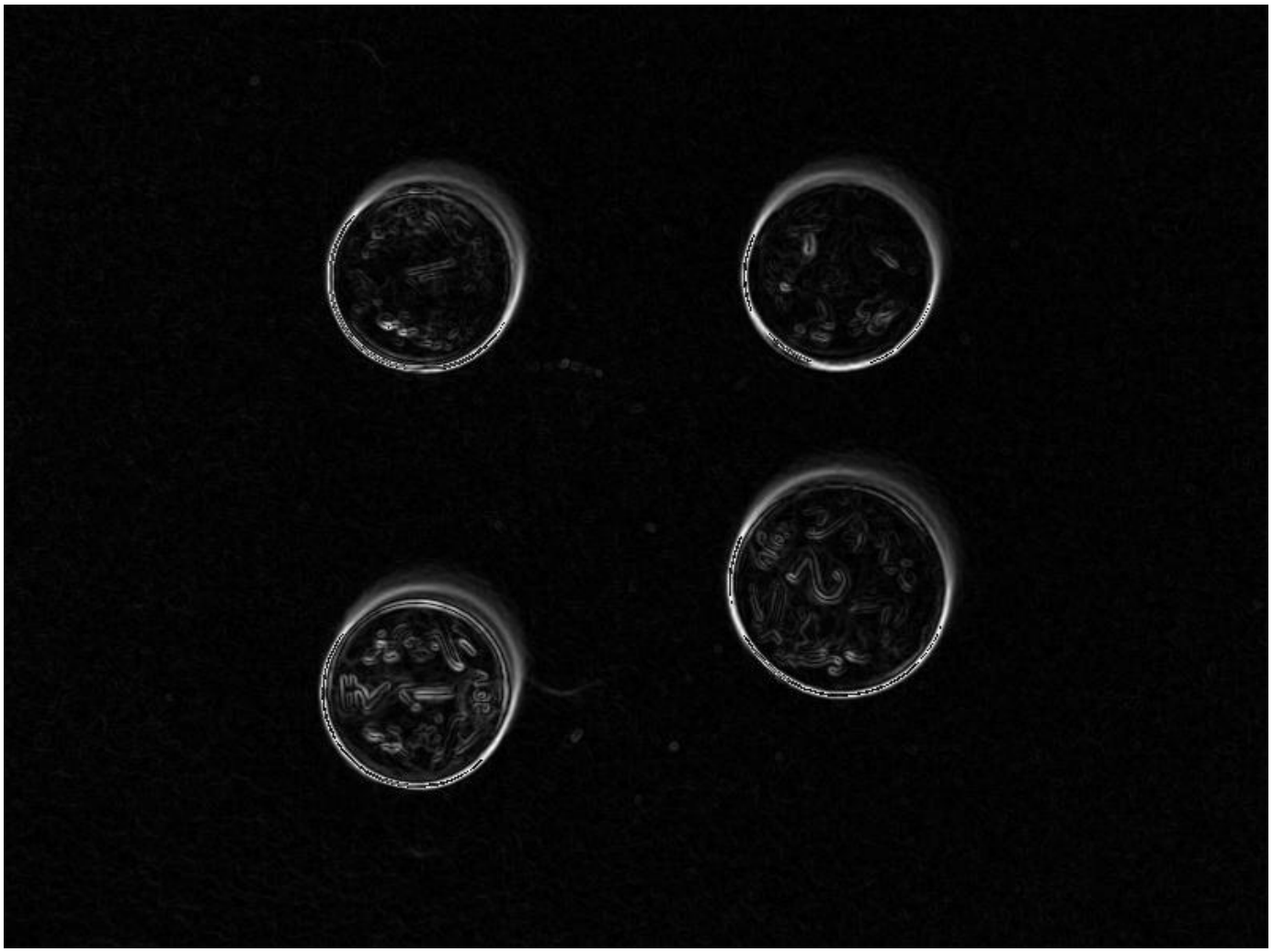
Output Images :

1. coin_detection.py :

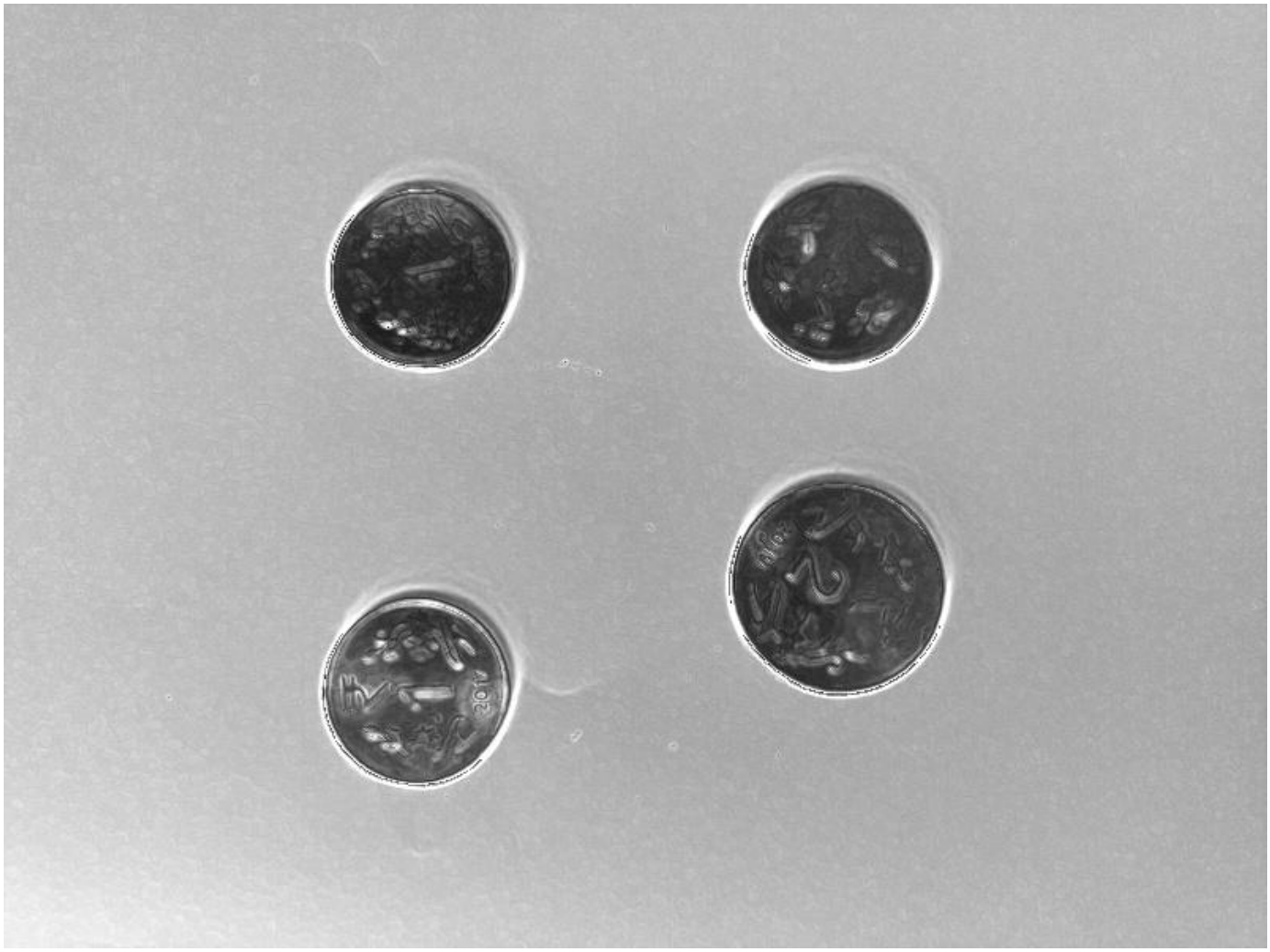
Input image converted to Grayscale :



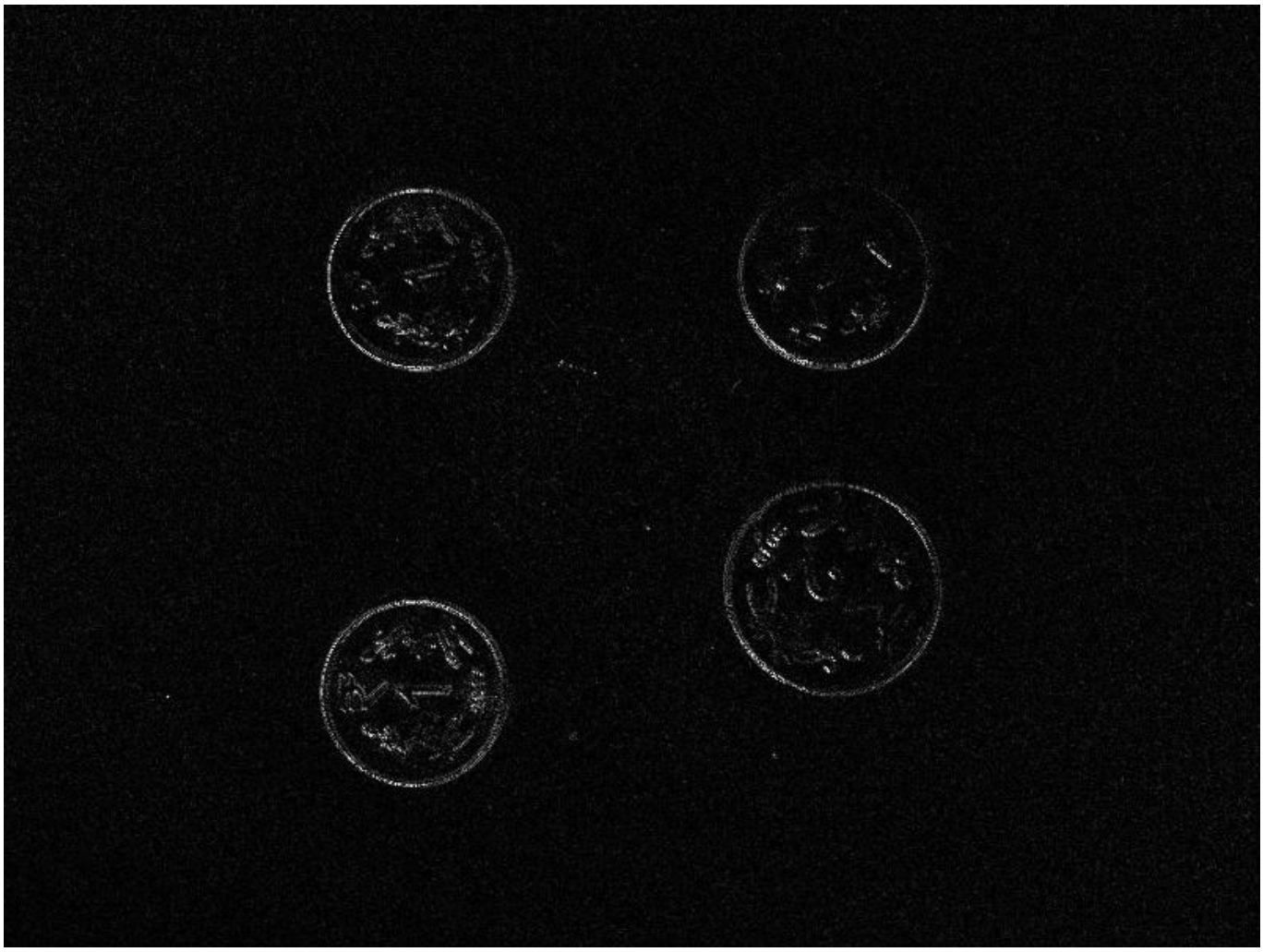
Sobel Edge Map :



Grayscale Input Image with Sobel Edges Highlighted :

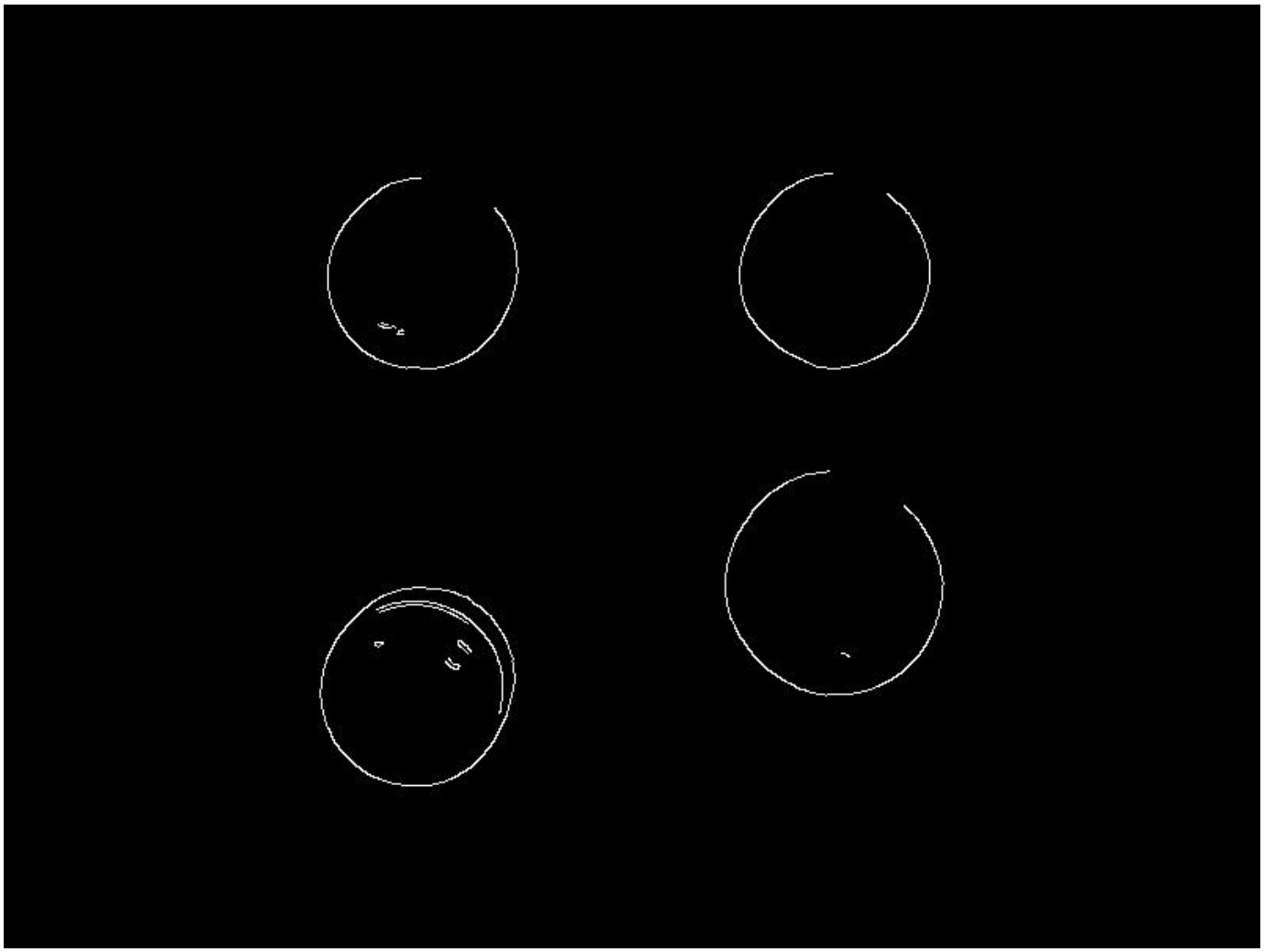


Laplacian Edge Map :

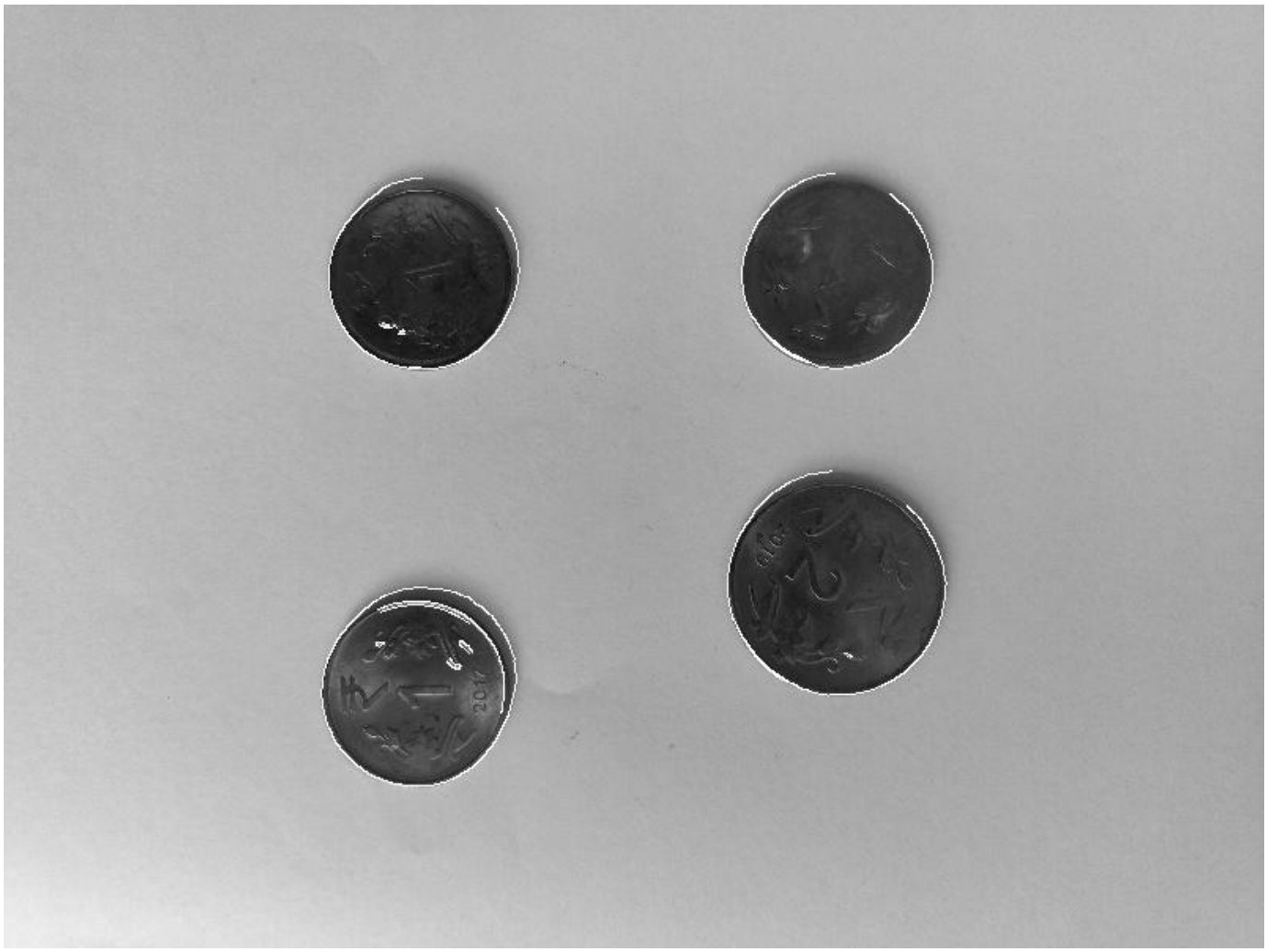


Grayscale Input Image with Laplacian Edges Highlighted :

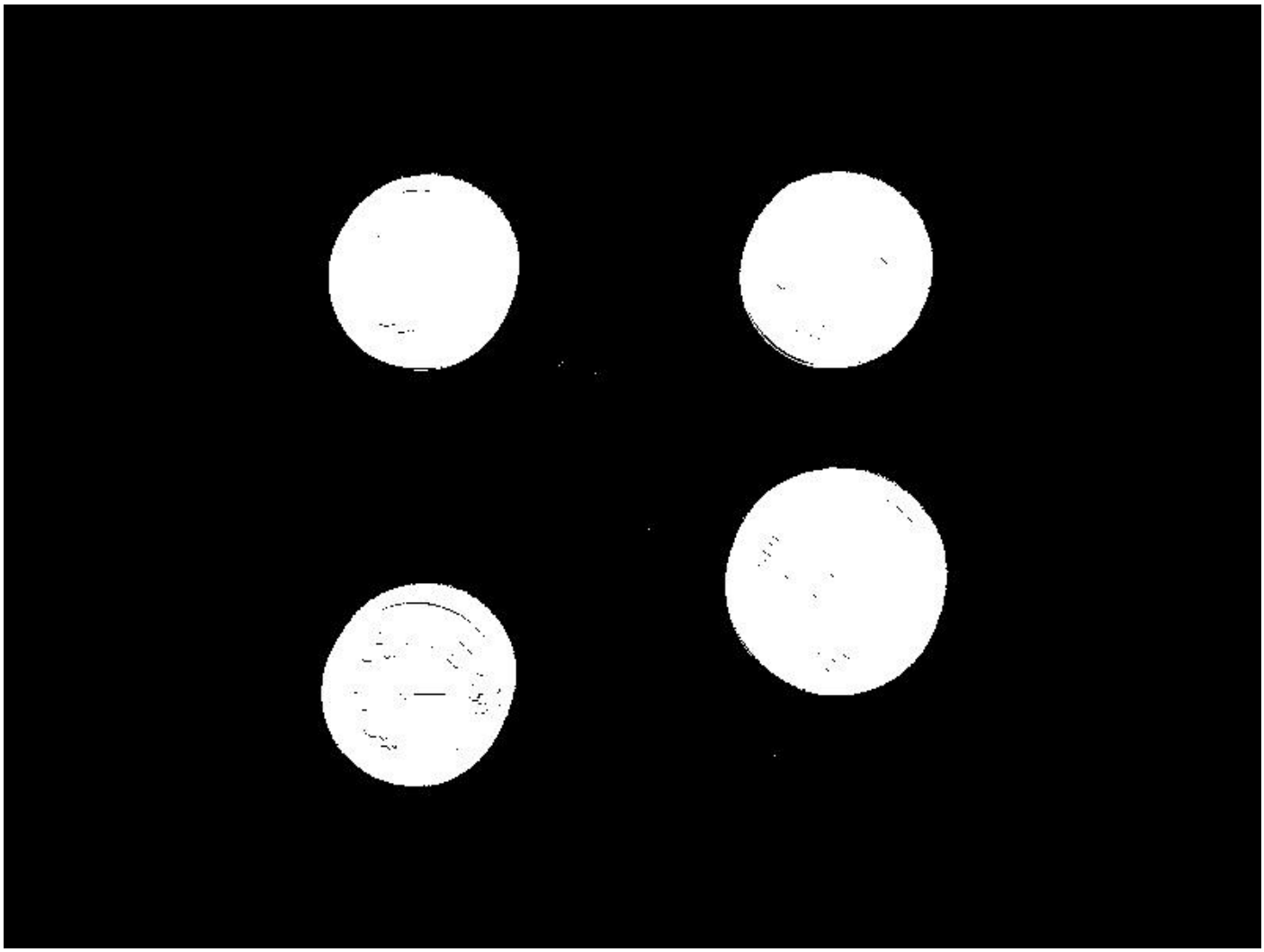
Cammy Edge Map :



Grayscale Input Image with Cammy Edges Highlighted :



Thresholding Output on Grayscale Input Image :

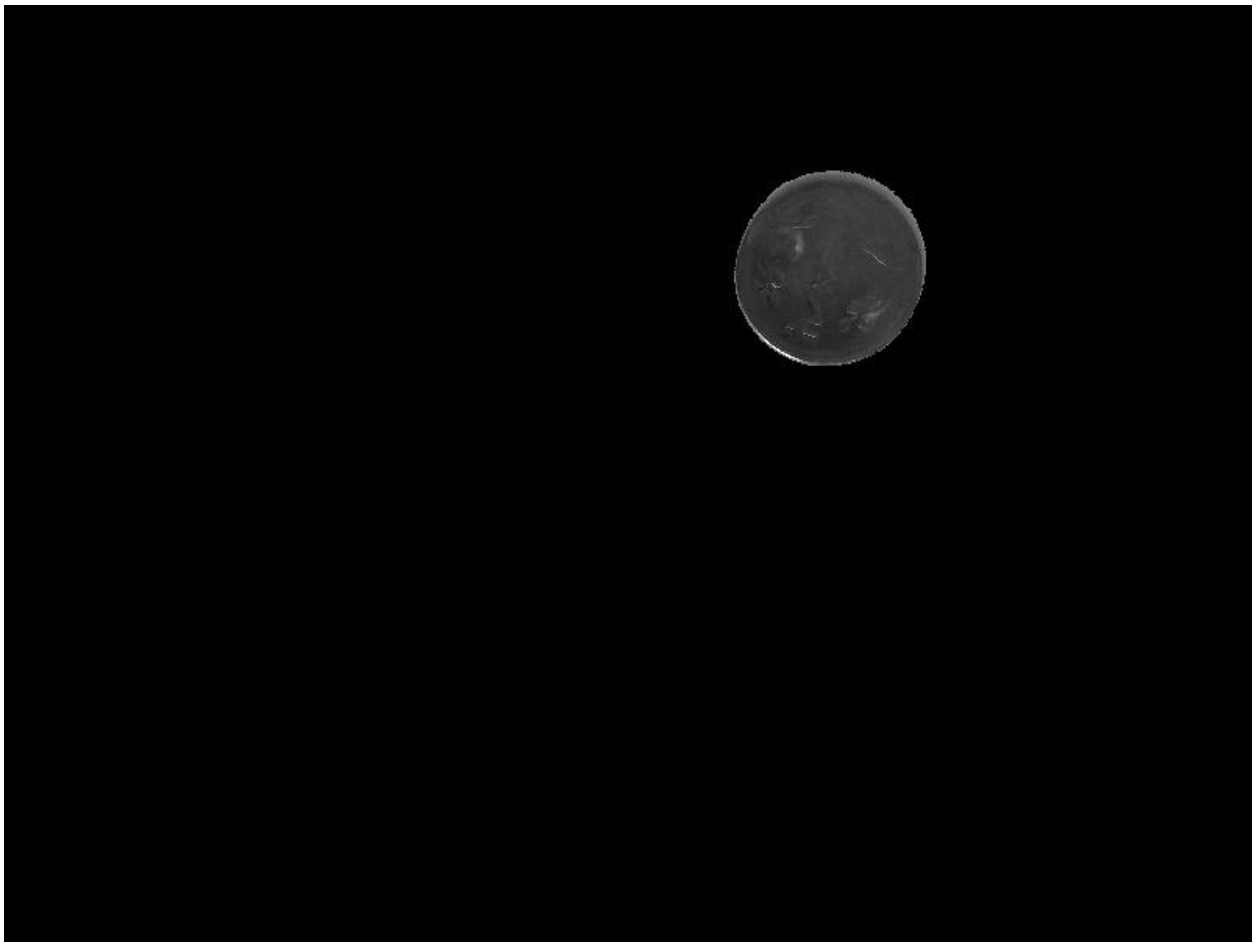


Segmented Coin 1 :

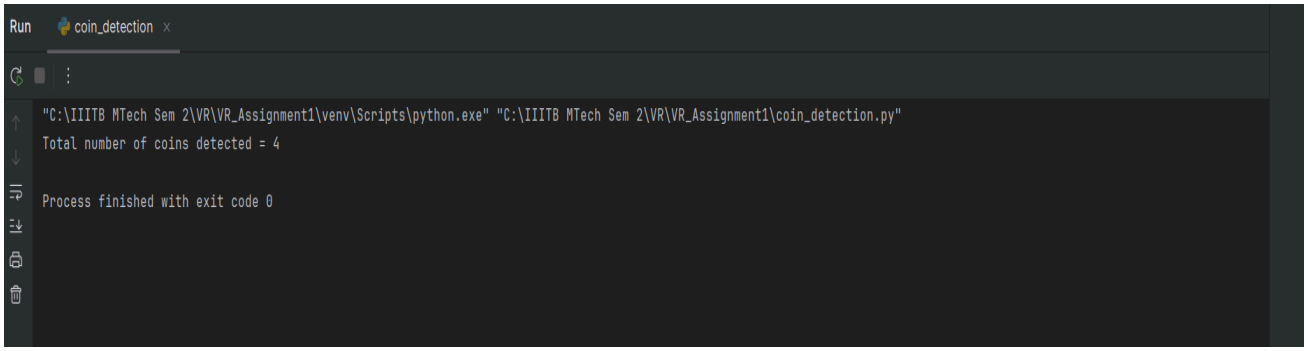
Segmented Coin 2 :

Segmented Coin 3 :

Segmented Coin 4 :



Final coin count returned by the counting function :



2. panorama_stitching.py :

Image 1 with keypoints drawn on it :

Image 2 with keypoints drawn on it :

Image 3 with keypoints drawn on it :

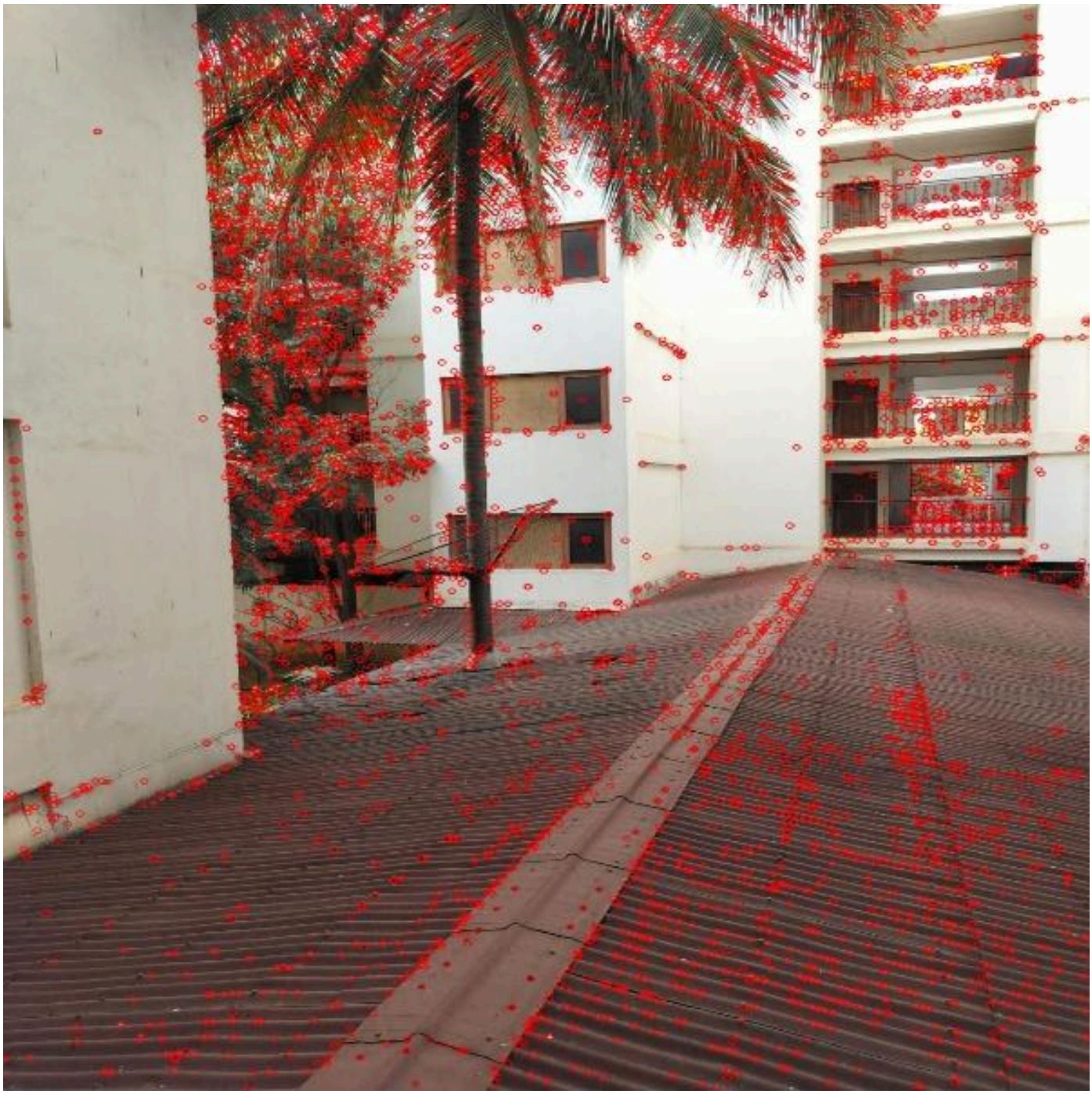


Image 4 with keypoints drawn on it :

Image 5 with keypoints drawn on it :

Image 6 with keypoints drawn on it :

Panorama Output :