ASSIGNMENT-1 REPORT

-ASHIRWAD MISHRA

IMT2022108

INTRODUCTION: This assignment consists of two computer vision tasks: coin detection and segmentation, and panorama stitching. The first part detects and segments coins in an image, while the second part stitches multiple images into a single panoramic view.

Part -1 : Coin Detection

INPUT IMAGE:

We use certain methods to carry out successful edge detection and coin counting in this Computer Vision task. All of them are mentioned next.

1. CLAHE (Contrast Limited Adaptive Histogram Equalization)

CLAHE enhances the contrast of the grayscale image by limiting the contrast amplification in regions where noise could be a problem. This helps in making the coin edges more distinct, improving their detection. It works by dividing the image into small tiles and adjusting the contrast locally rather than globally. This prevents overly bright or dark areas from dominating the entire image.

2. Gaussian Blur

Blurring the image using a Gaussian kernel helps reduce noise and smooth the image, making it easier for the edge detection algorithms to focus on significant edges rather than minor details. It works by averaging pixel values around a selected region, which helps in suppressing unwanted small details. This ensures that only the most important features, like coin edges, remain visible.

3. Hough Circle Detection

This method detects circular shapes in the image using the Hough Transform. It works by identifying potential circle candidates based on gradient changes and accumulating votes in a parameter space. The algorithm looks at different edge patterns in the image and decides which ones are likely to form a circle. It is useful for

detecting coins since they have a well-defined circular shape.

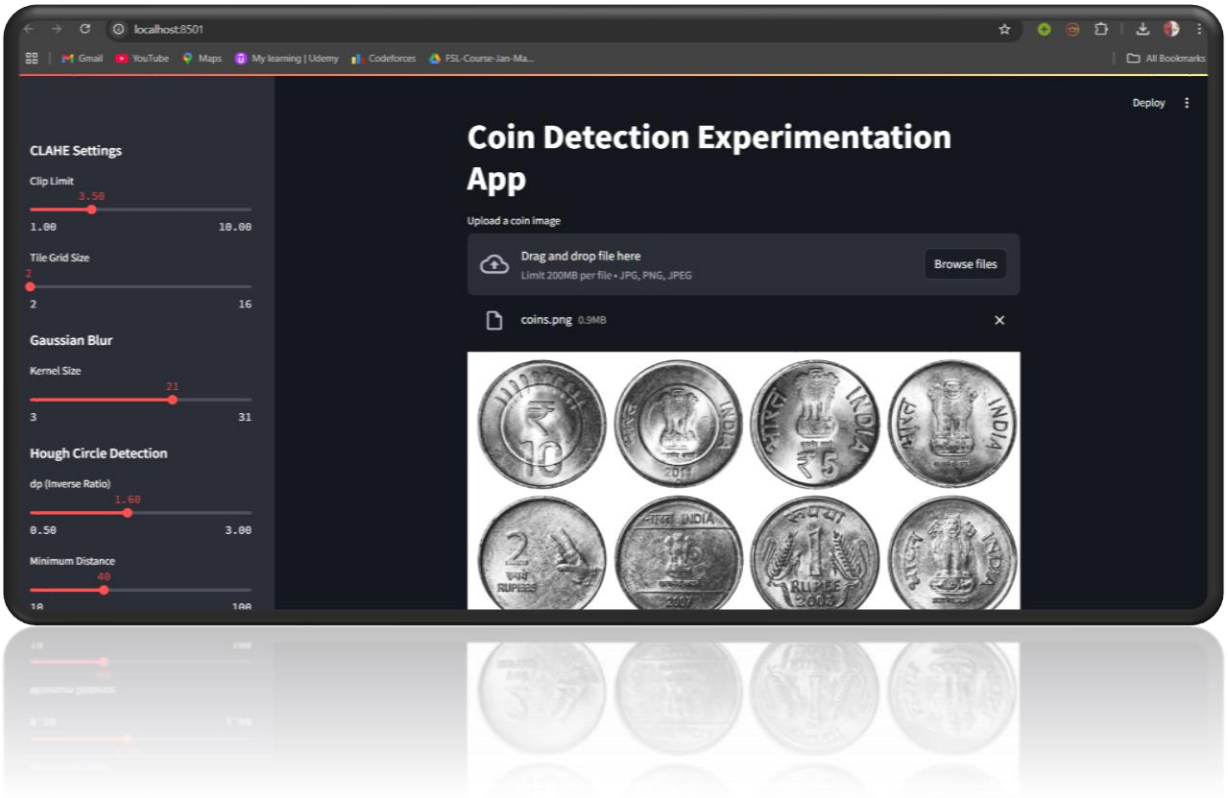
4. Canny Edge Detection

Canny Edge Detection is used to extract significant edges from the image. It involves applying gradient detection and suppression steps to highlight coin boundaries effectively. The process works in multiple stages, including noise reduction, finding the intensity gradient, and edge thinning. By adjusting its threshold values, we can control how strong or weak the detected edges appear.

Next, we go through a parameter selection process which is carried out using a streamlit dashboard. This was done to make it easier to tweak parameters and reach an optimal value for them using sliders and real time update.

Parameter Selection Process

The parameters were selected through an iterative process by adjusting the sliders in the Streamlit interface.



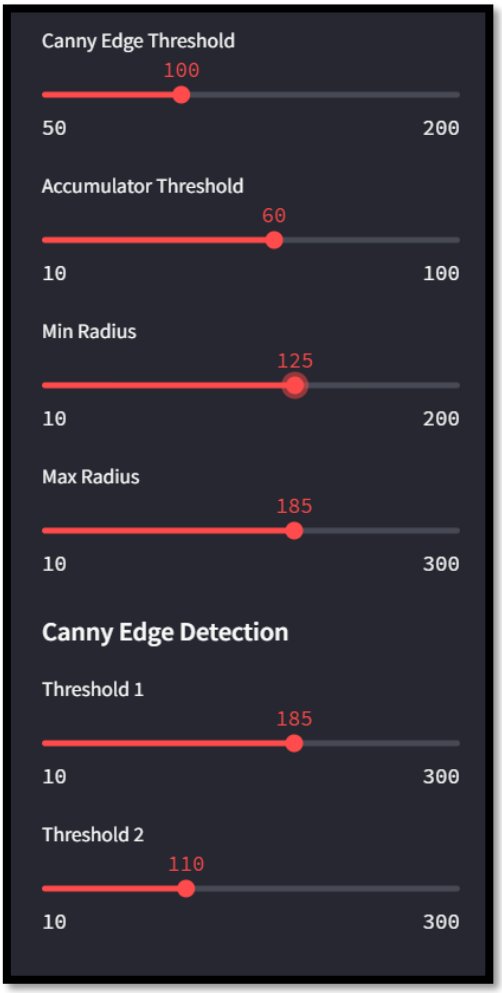
The following approach was used:

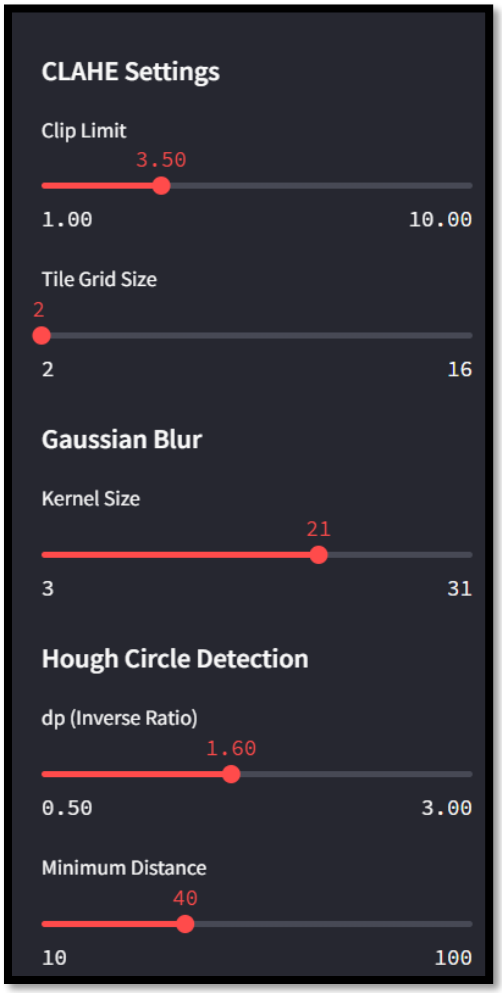
- **CLAHE Settings**: The clip limit and tile grid size were fine-tuned to ensure that the contrast enhancement does not introduce excessive noise while still making the coin edges distinct.
- **Gaussian Blur**: The kernel size was adjusted to smooth out noise without over-blurring the coin edges.
- **Hough Circle Detection**: Several parameters such as dp, minDist, param1, param2, minRadius, and maxRadius were modified to ensure that circles representing coins were detected correctly without false positives.

- **Canny Edge Detection**: The two threshold values were tuned to balance detecting edges clearly while avoiding unnecessary details from noise.

Parameter selection result

After multiple iterations, the optimal parameters were determined (as shown in the attached images).





These parameters successfully detect the correct number of coins in the image. Based on this analysis, the selected values will now be hard-

coded into the final version of the coin detection program to ensure consistent result.

OUTPUT IMAGES(COIN DETECTION):

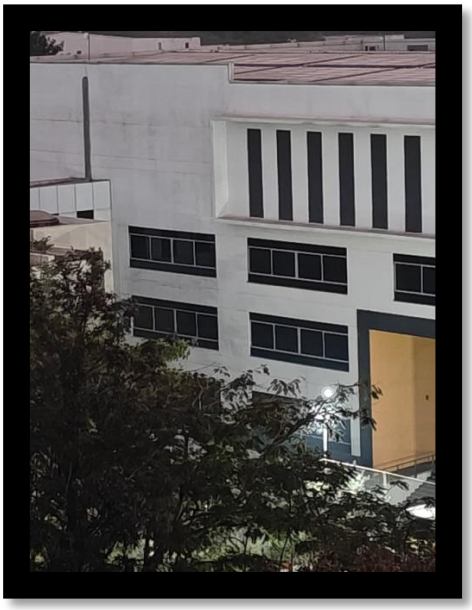
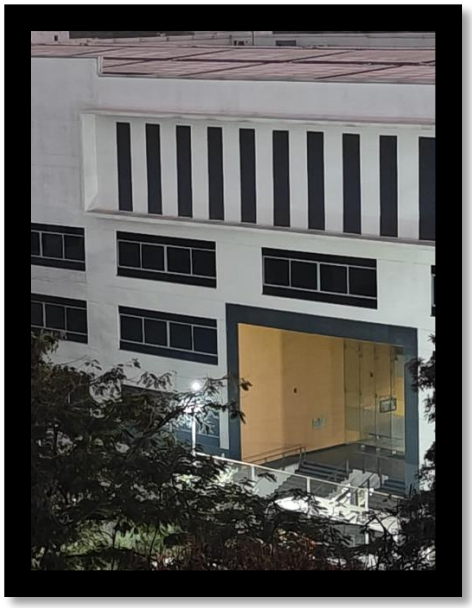
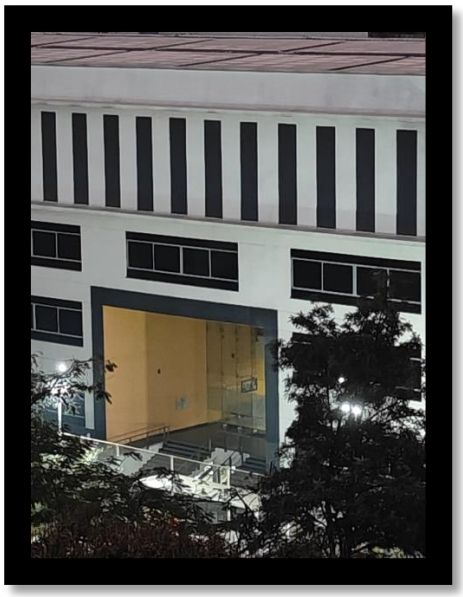
1 – Canny edge detector output, 2-segmentation output

3 – Hough circle detection output

Once the python script – 'coin_detection.py' is run successfully, a folder is created named – 'coin_images'. It contains all these images shown above along with each coin's image separated from the raw input image.

Part -1 : Image Stitching

INPUT IMAGES:

An important note – I experimented with a few types of set of images to carry out this stitching and the best output I got was for the given image, wherein, the zoom setting was 6X and the orientation of my device hardly changed while clicking these pictures. This is quite evident that the orientation wouldn't change much due to such high level of zoom. If you take an image of nearby objects rather than further objects, then this program may not give desired output because the heavy change in orientation of the images captured makes it really difficult to match the feature descriptors of the points on the edge. Any far scenery without any zoom, which in turn make you move the camera a lot, will result in a good stitching, but the output would be a highly warped perspective with a lot of black regions on vertices of the output image. So the best way to click, for this program would be – a distant scenery (with sufficient amount of

objects, not a horizon or a field, so that one can make out that the stitching is done properly), with substantial amount of Zoom in and very less lateral camera movement.

1. Feature Detection (SIFT)

SIFT (Scale-Invariant Feature Transform) detects key points in images that remain consistent across different views. These key points help in identifying overlapping areas between images. By extracting unique features, SIFT ensures that images can be aligned correctly for stitching. This method is useful because it finds stable points that do not change even if the image is rotated or scaled. It helps in ensuring accurate alignment when combining multiple images.

2. Feature Matching (Brute-Force Matcher)

To align images, we need to find common points between them. The Brute-Force Matcher compares key points from two images and finds the best matches. A filtering step ensures that only the best matches are kept, removing poor-quality ones. This is done by comparing the descriptors of key points and keeping the ones that are most similar. A ratio test is also used to remove incorrect matches and improve accuracy.

3. Homography Transformation

Homography is a mathematical transformation that aligns one image with another using the matched key points. It helps in warping an image

so that it fits onto another image correctly. This step ensures that images are aligned properly before blending them. Homography uses perspective transformation, meaning it corrects for camera angles and distortions, making sure the images look seamless when stitched together.

4. Image Blending

Once images are aligned, they need to be blended smoothly to remove visible seams. A weighted gradient blending approach ensures that the transition between images is smooth. This helps in making the final stitched image look natural without harsh edges. Blending techniques prevent sharp boundaries from being visible, and ensure that lighting differences between images do not create noticeable changes.

5. Removing Black Borders

After stitching, there may be unwanted black regions in the image. These are cropped out by detecting the valid image region, ensuring that only the useful parts remain in the final panorama. The algorithm finds the boundary where the valid image content starts and removes any extra black areas. This makes the final output look clean and professional.

OUTPUT IMAGES(IMAGE STITCHING):

FEATURE DESCRIPTOR OUTPUTS

(img-1,2 from L to R in row-1, img-3 in row-2)

FINAL STITCHED IMAGE OUTPUT:

Once the python script – ‘image_segmentation.py’ is run successfully, a folder is created named – ‘stitched_images’. It contains all these images shown above.

Sources:

1. **CLAHE -** https://www.geeksforgeeks.org/clahe-histogram-eqalization-opencv/
2. **Hough Circle detection -** https://www.geeksforgeeks.org/circle-detection-using-opencv-python/
3. **Watershed algorithm for segmentation -** https://www.geeksforgeeks.org/image-segmentation-with-watershed-algorithm-opencv-python/
4. **Image stitching -** https://www.geeksforgeeks.org/image-stitching-with-opencv/