CSc 8830: Computer vision Assignment-2 Report

SAI TEJASWI CHAKRAVARAM

002762775

Capture a 10 sec video footage using a camera of your choice. The footage should be taken with the camera in hand and you need to pan the camera slightly from left-right or right-left during the 10 sec duration. For all the images, operate at grayscale unless otherwise specified:

A video of 10 sec was recorded from a mobile camera with 1030p resolution and 30FPS frame rate.

1.Pick any image frame from the 10 sec video footage. Pick a region of interest in the image making sure there is an EDGE in that region. Pick a 5 x 5 image patch in that region that constitutes the edge. Perform the steps of CANNY EDGE DETECTION manually and note the pixels that correspond to the EDGE. Compare the outcome with MATLAB or OpenCV or DepthAI's Canny edge detection function.

Method:

Handheld Canny Edge Identification Procedures: 1. Image Patch Selection:
● Choose a 5x5 image patch that has a frame edge region in it. Make sure the patch has distinct boundaries so that it can be detected correctly.
2. Converting picture Patch to Grayscale:
● Make the chosen picture patch grayscale. As a result, edge detection is made simpler.
3. Gaussian Smoothing:
● To lessen noise and extraneous details, apply a Gaussian blur to the grayscale image.
4. Gradient Calculation:
● Use Sobel operators to determine the gradient's direction and amplitude. Determine the gradients in the directions of x and y.

5. Non-maximum Suppression:
● Only the local maxima remain as potential edges by suppressing non-maximum gradient values. Evaluate each pixel's gradient magnitude by comparing it to the gradient direction's neighbouring magnitudes.

6. Double Thresholding:

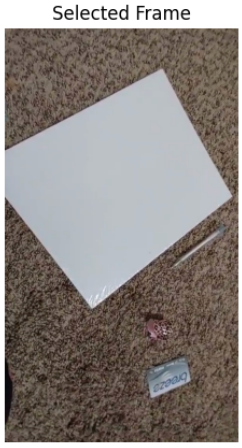
● Utilise double thresholding to categorise pixels into strong, weak, and non-edge categories according to the gradient magnitudes of those pixels.
Pixels that possess gradient magnitudes greater than a certain threshold are categorised as strong edges.
Pixels defined as weak edges are those whose gradient magnitude falls between low and high criteria; those classed as non-edge pixels are those whose gradient

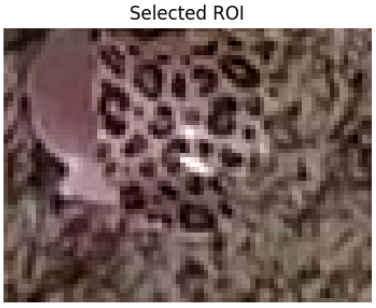
magnitude falls below a low threshold and are eliminated.
7. Hysteresis-Based Edge Tracking:

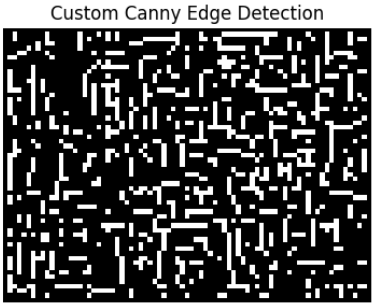
● Use hysteresis-based edge tracking to connect weak edge pixels to strong edge pixels and create continuous edges.

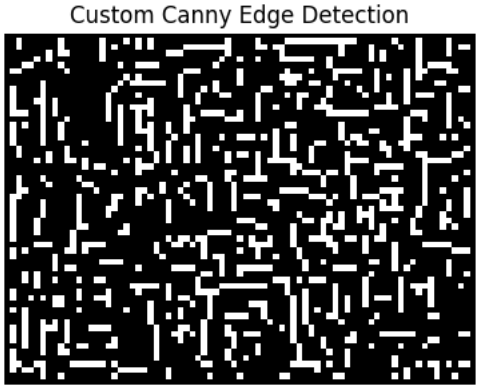
● Begin with pixels that have strong edges and proceed along the edges by joining neighbouring weak edges that are related to strong edges.
8. Edge Pixel Identification:

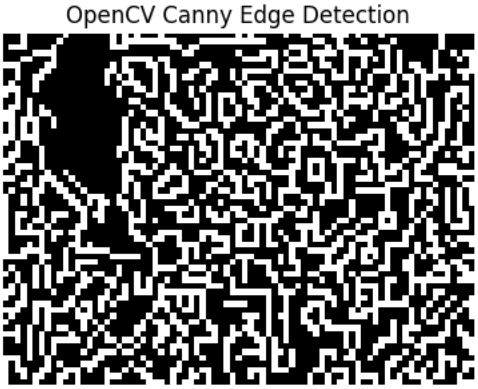
● Following edge tracking, determine which pixels match the edges that were discovered.









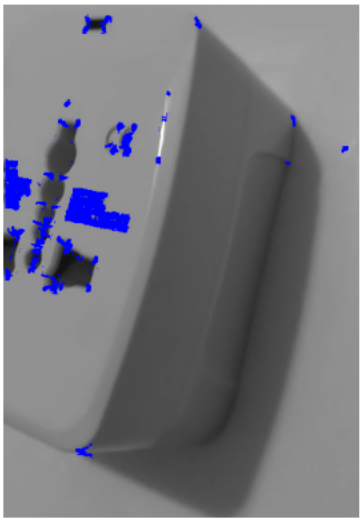


2. Pick any image frame from the 10 sec video footage. Pick a region of interest in the image making sure there is a CORNER in that region. Pick a 5 x 5 image patch in that region that constitutes the edge. Perform the steps of HARRIS CORNER DETECTION manually and note the pixels that correspond to the CORNER. Compare the outcome with MATLAB or OpenCV or DepthAI's Harris corner detection function.

Selected Frame

Selected ROI

Manual Harris Corner Detection



CV2 Harris Corner Detection

Method: Manual Steps for Detecting a Harris Corner:

1. Image Patch Selection:

● Choose a 5 by 5 image patch that includes a frame corner area. Make sure the patch has a visible corner so that it can be detected correctly.

2. Converting picture Patch to Grayscale:
● Make the chosen picture patch grayscale. This makes the process of detecting corners simpler.
3. Calculate Gradients:
● Use Sobel operators or alternative gradient computation techniques to compute the gradients in both the x and y directions.
4. Compute the structure tensor for every pixel in the image patch using the structure tensor calculation method. A matrix that condenses the local image structure surrounding each pixel is called the structure tensor.
5. Corner reaction Function: Using the structure tensor, determine each pixel's corner reaction function.
Typically, the Harris corner detector applies the following formula:

6. R = det(M) - k * (trace(M))^2

Where M is the structure tensor, det(M) is the determinant of M, trace(M) is the

trace of M, and k is an empirically determined constant (typically 0.04).

7. Thresholding: ● Determine possible corner points by setting a threshold for the corner response function. Pixels that exhibit corner response values greater than a specific threshold are classified as corner candidates.
8. Non-maximum Suppression: Use non-maximum suppression to keep the corner response function's local maxima alone. This makes it possible to guarantee that only the strongest corner points are identified.
9. Corner Point Identification: ● Find the pixels that, following non-maximum suppression, correspond to the corners that were detected.
10. OpenCV: ● Construct an image patch and apply Harris corner detection using OpenCV's cv2.cornerHarris() method. Compare the manual corner detection results.

3. Consider an image pair from your footage where the images are separated by at least 2 seconds. Also ensure there is at least some overlap of scenes in the two images.

Image 1 with SIFT Keypoints

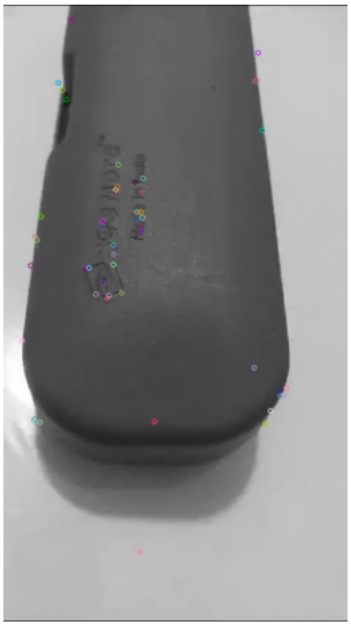
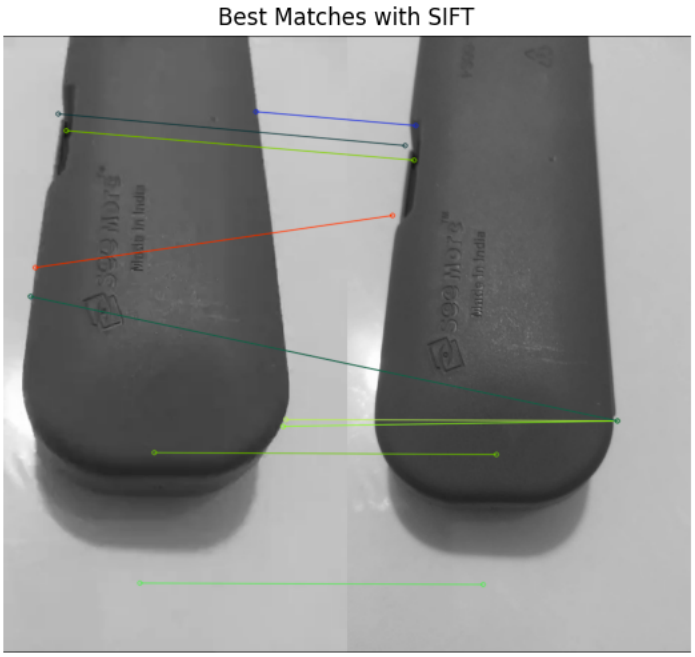
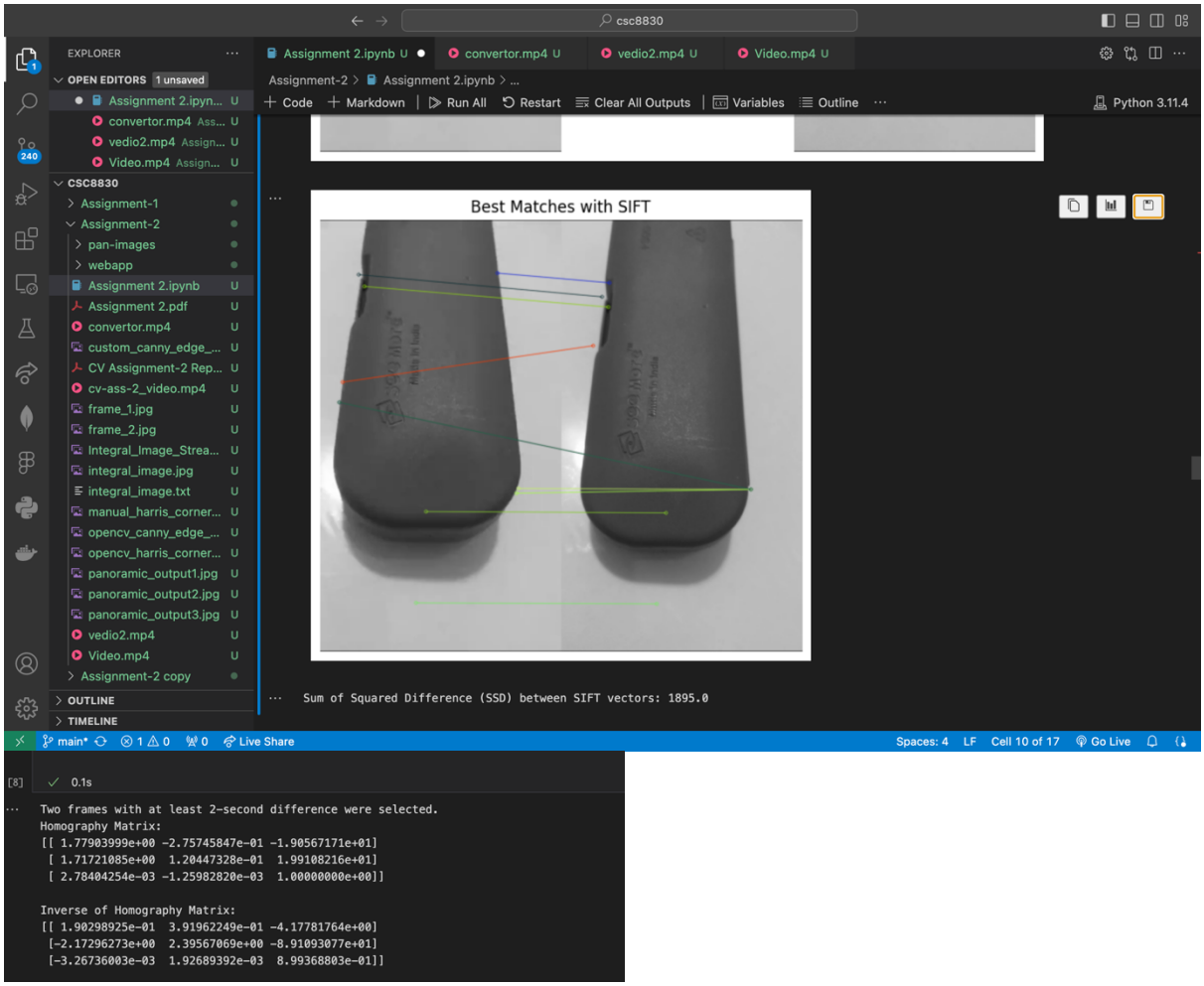


Image 2 with SIFT Keypoints





a. Select a pixel from image 1 (the super-pixel patch that was covered in class) and a corresponding pixel from image 2 (the pixel on image 2 that corresponds to the same object area on image 1). For each of these two patches, calculate the SIFT feature. Determine the SIFT vector's sum of

squared differences (SSD) value for these two pixels. Make use of C++, Python, or MATLAB implementation The following link will download the MATLAB code for SIFT feature extraction and matching: https://www.cs.ubc.ca/~lowe/keypoints/ To find out how to run the code, please read the ReadMe document inside the folder.
b. Use MATLAB, Python, or C++ to compute the homography matrix between these two images. Determine its inverse.

Calculating SSD between vectors in SIFT:

1. Select Image Patches:
● Pick a patch in image 1 (or a super-pixel patch) and the patch in image 2 that corresponds to it. Make sure that the scenes that were photographed in the two pictures somewhat overlap.
2. Extract SIFT Features:
● Extract SIFT descriptors for the chosen patches in both pictures using a SIFT feature extraction approach (e.g., OpenCV's cv2.SIFT or cv2.SURF functions).
3. Compute SIFT Descriptors:
* Determine the SIFT descriptors for each of the two images' chosen patches. These characteristics show what makes the patches unique.
4. Determine SSD:
● Determine the sum of squared differences (SSD) between the respective patches' SIFT descriptors in the two images. This entails calculating and adding the squared differences between every pair of corresponding elements in the two SIFT descriptors.

Computing Homography Matrix:

1. Feature Matching:
● Find matches between the SIFT keypoints identified in both images using a feature matching technique (e.g., OpenCV's cv2.FlannBasedMatcher or cv2.BFMatcher).
2. Filter Matches:
● Use a filter, such as a distance cutoff or ratio test, to exclude untrustworthy matches.
3. Estimate Homography:
● To estimate the homography matrix, which converts points from one image's perspective to another, use a reliable estimation technique (like RANSAC).
4. Compute Inverse Homography: Compute the estimated homography matrix's inverse. Points can be converted from the perspective of picture 2 to that of image 1 using this inverse matrix.

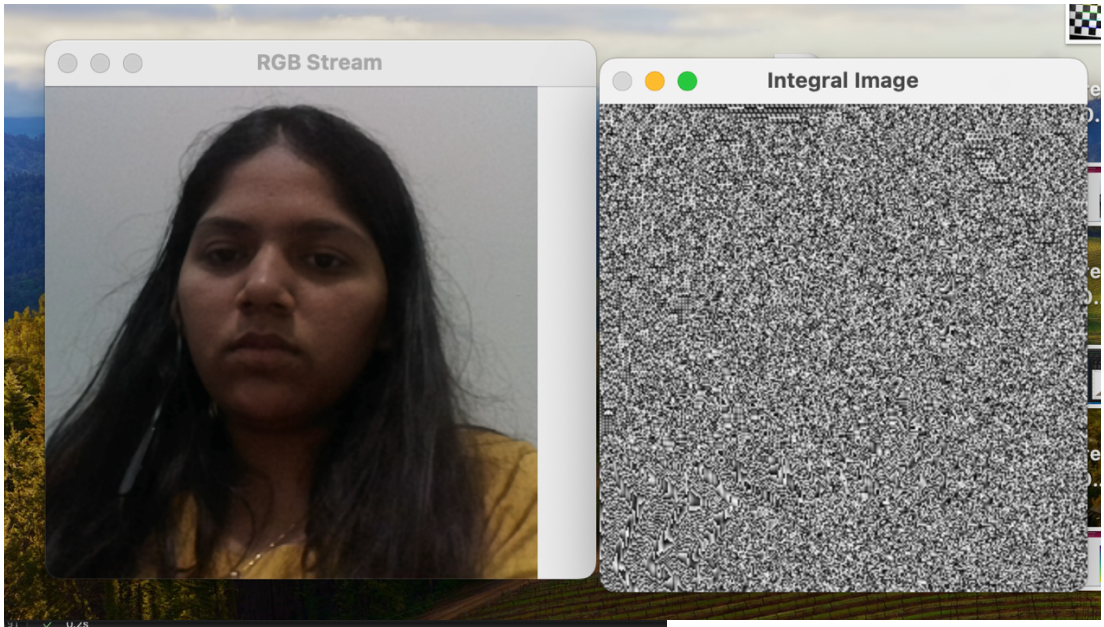
4. Implement an application that will compute and display the INTEGRAL image feed along with the RGB feed. You cannot use a built-in function such as “output = integral_image(input)”

Method:

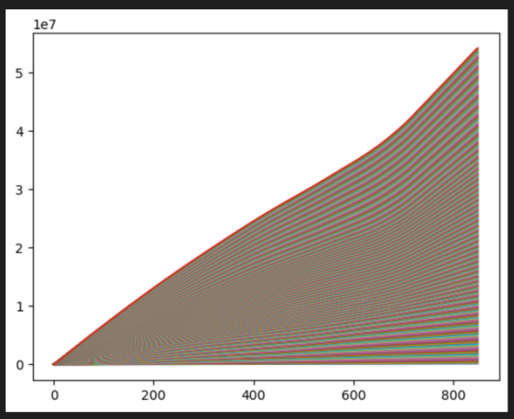
1. Read RGB Frames: Constantly take pictures using the RGB camera.
2. Convert to Grayscale: Make every RGB frame a grayscale conversion.
3. Compute Integral Image: The integral image pixel value is calculated by iterating over each pixel in the grayscale frame and adding the pixels to the left and above the current pixel.

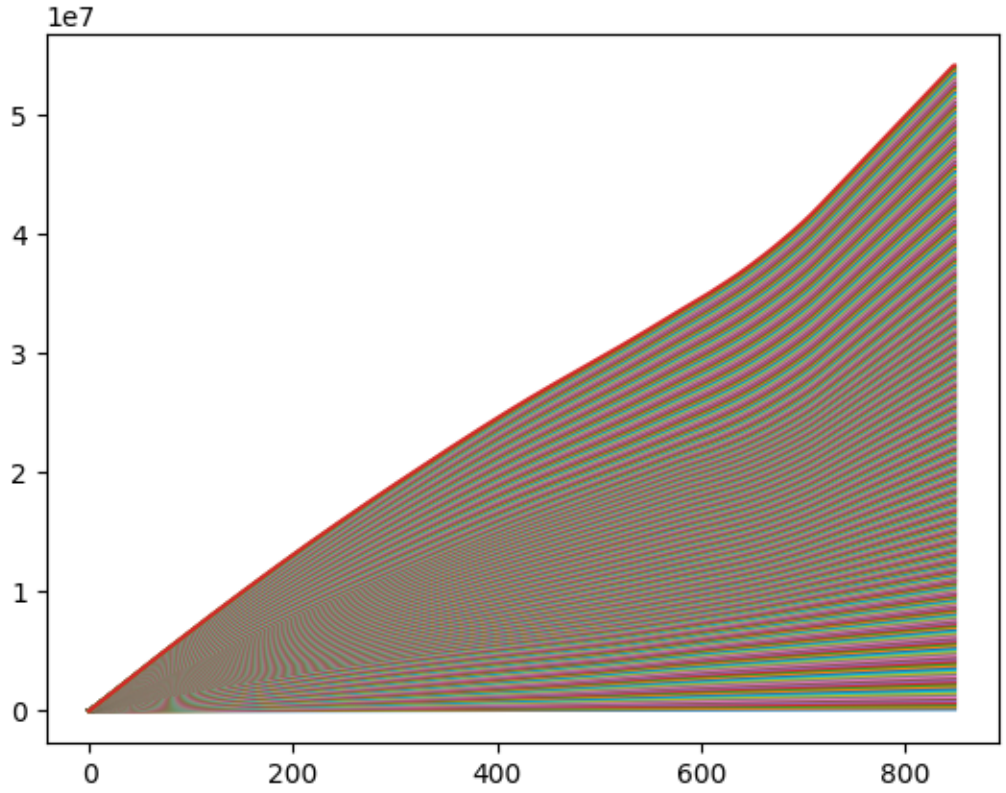
The fourth step is to store the integral image that has been computed.


Sag Mart
Made In India




```
[[      0       0       0 ...    1981    1982    1983]
 [    202     404     606 ...   69564   69758   69952]
 [    402     804    1206 ...  136354  136733  137112]
 ...
 [ 172972  345981  519090 ... 53623062 53788124 53953186]
 [ 173150  346337  519624 ... 53707316 53872565 54037814]
 [ 173151  346339  519627 ... 53707878 53873130 54038382]]
```





5. Implement the image stitching for a 360 degree panoramic output. This should function in real-time. You can use any type of features. You can use built-in libraries/tools provided by OpenCV or DepthAI API. You cannot use any built-in function that does output = image_stitch(image1, image2). You are supposed to implement the image_stitch() function

Step 1: Feature Detection and Matching

● Feature Detection: Detect keypoint features in each frame using a feature detection algorithm such as ORB, SIFT, or SURF.

● Feature Description: Compute descriptors for the detected keypoints to represent their local neighborhood.

● Feature Matching: Match keypoints between consecutive frames to find corresponding points.

Step 2: Estimate Homography

● Find Homography: Use a robust estimation method such as RANSAC to estimate the homography matrix between pairs of consecutive frames.

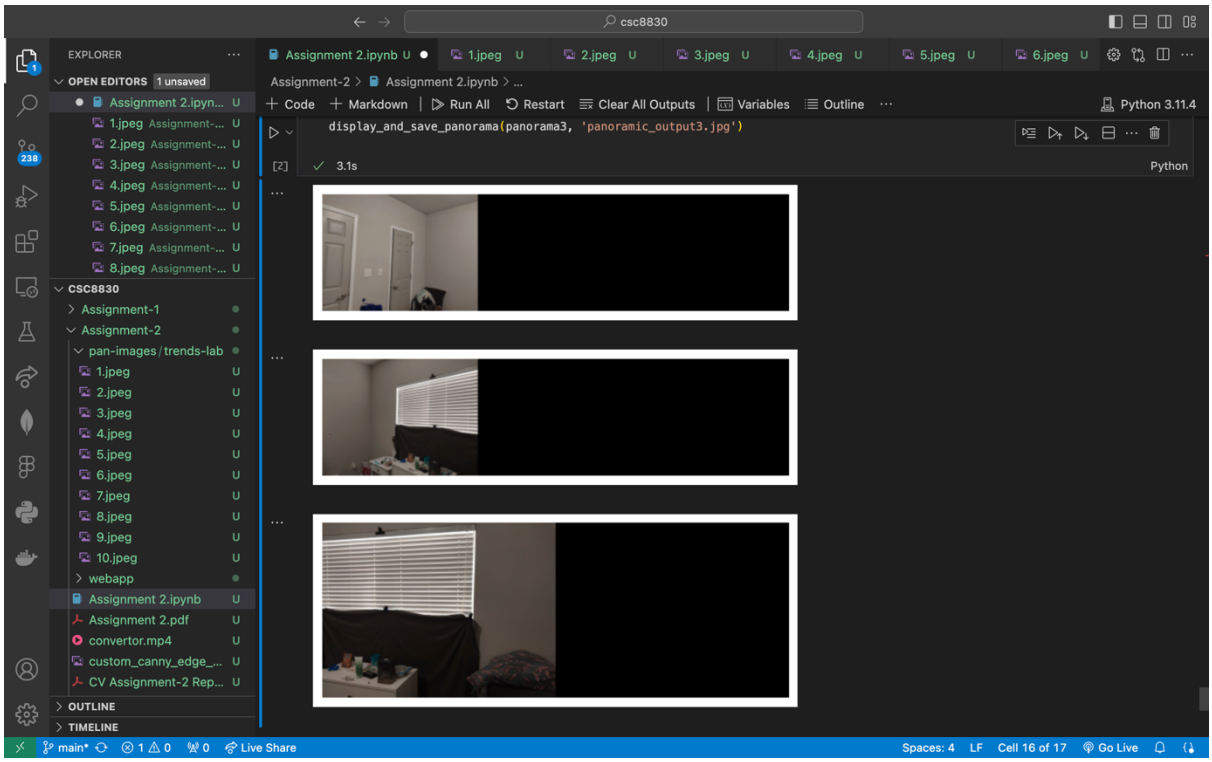
● Warp Images: Warp one of the frames using the estimated homography matrix to align it with the other frame.

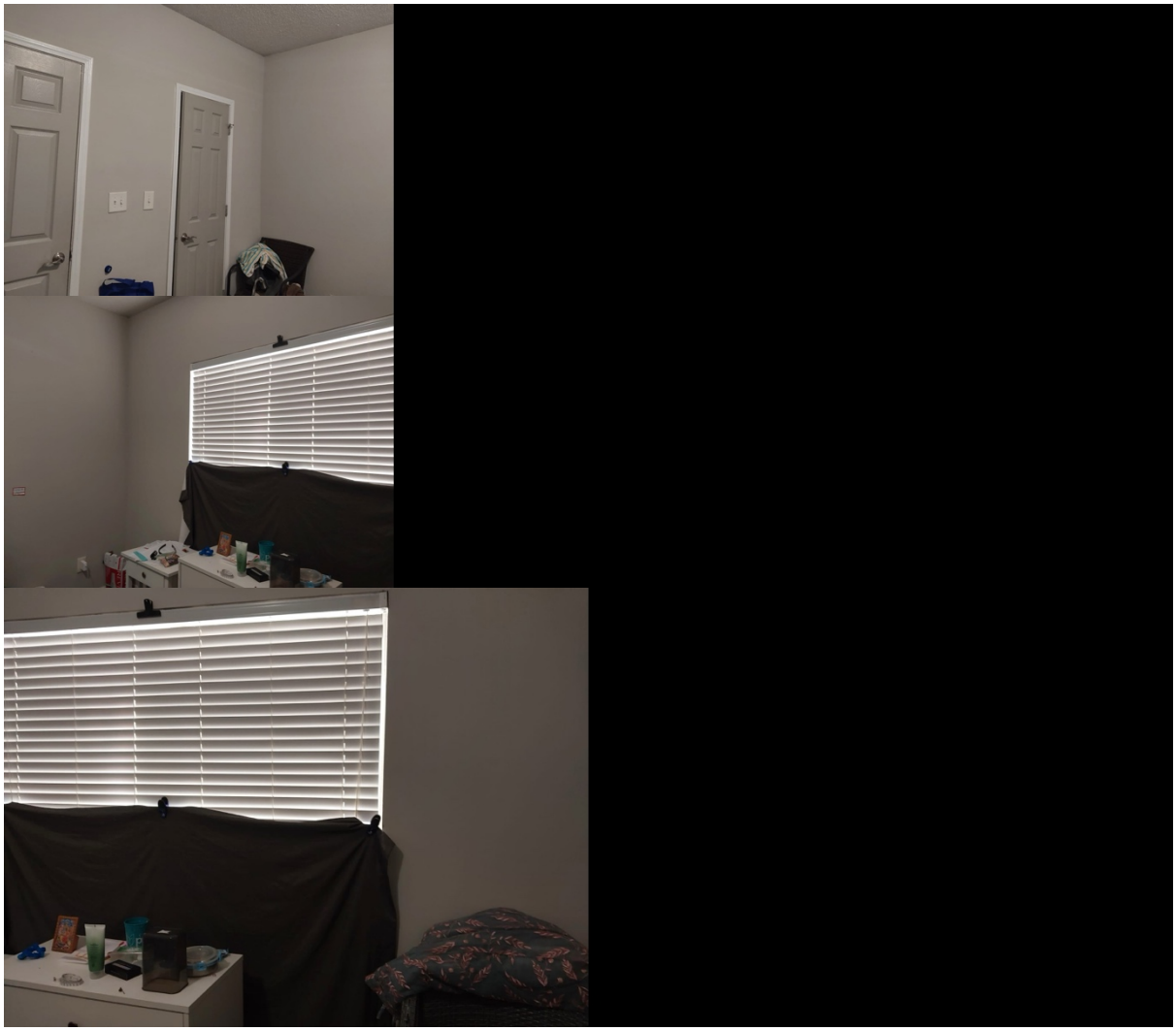
Step 3: Blending

● Blend Images: Blend the warped frame with the reference frame to seamlessly merge the overlapping regions.

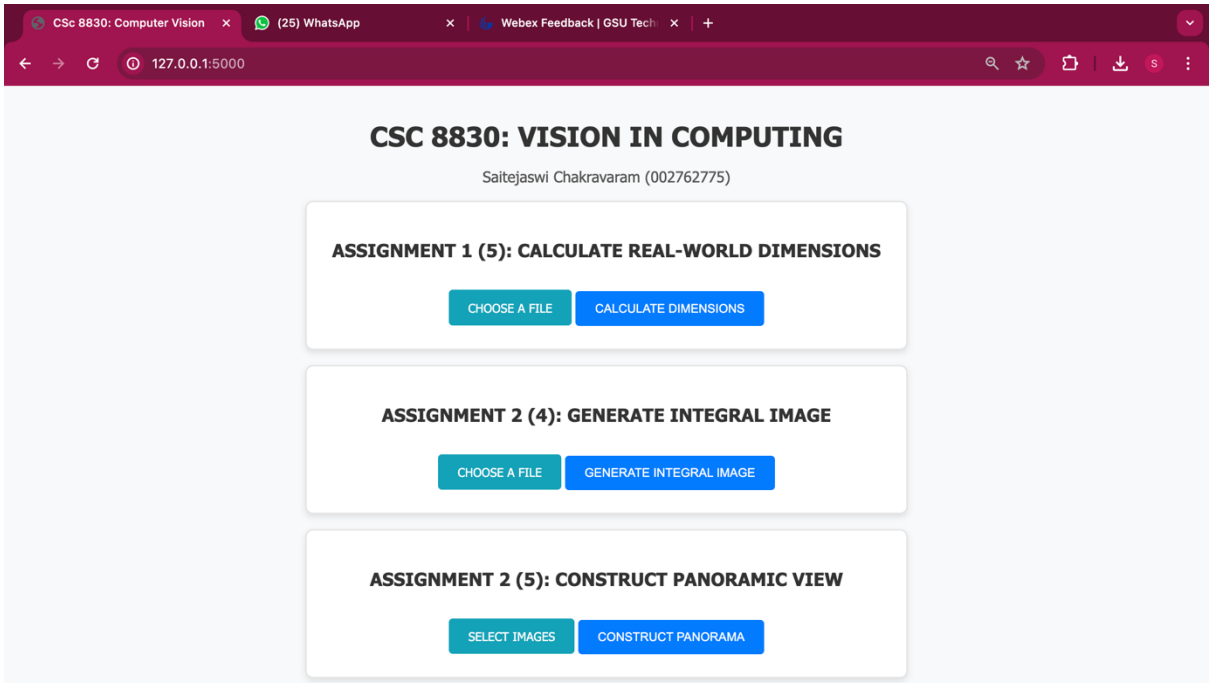
Step 4: Repeat for Multiple Frames

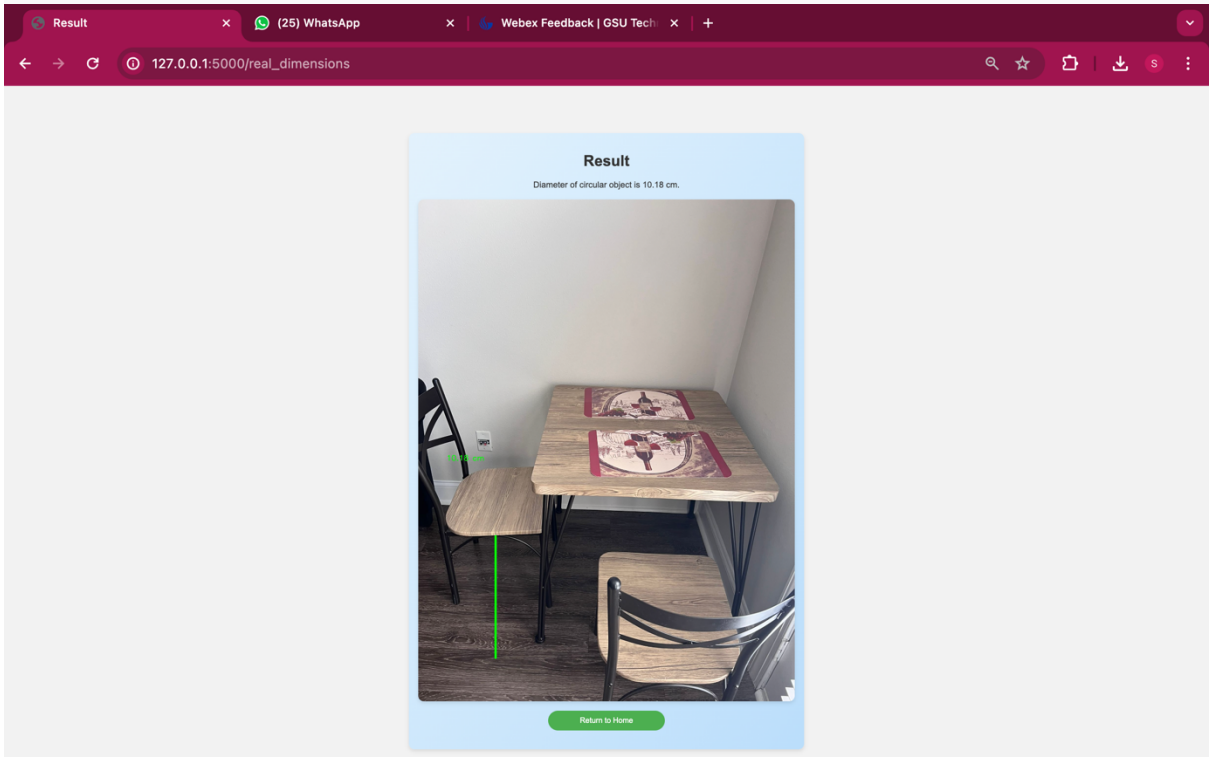
● Iterative Process: Repeat the feature detection, matching, homography estimation, and blending steps for multiple consecutive frames to stitch them together into a panoramic output.

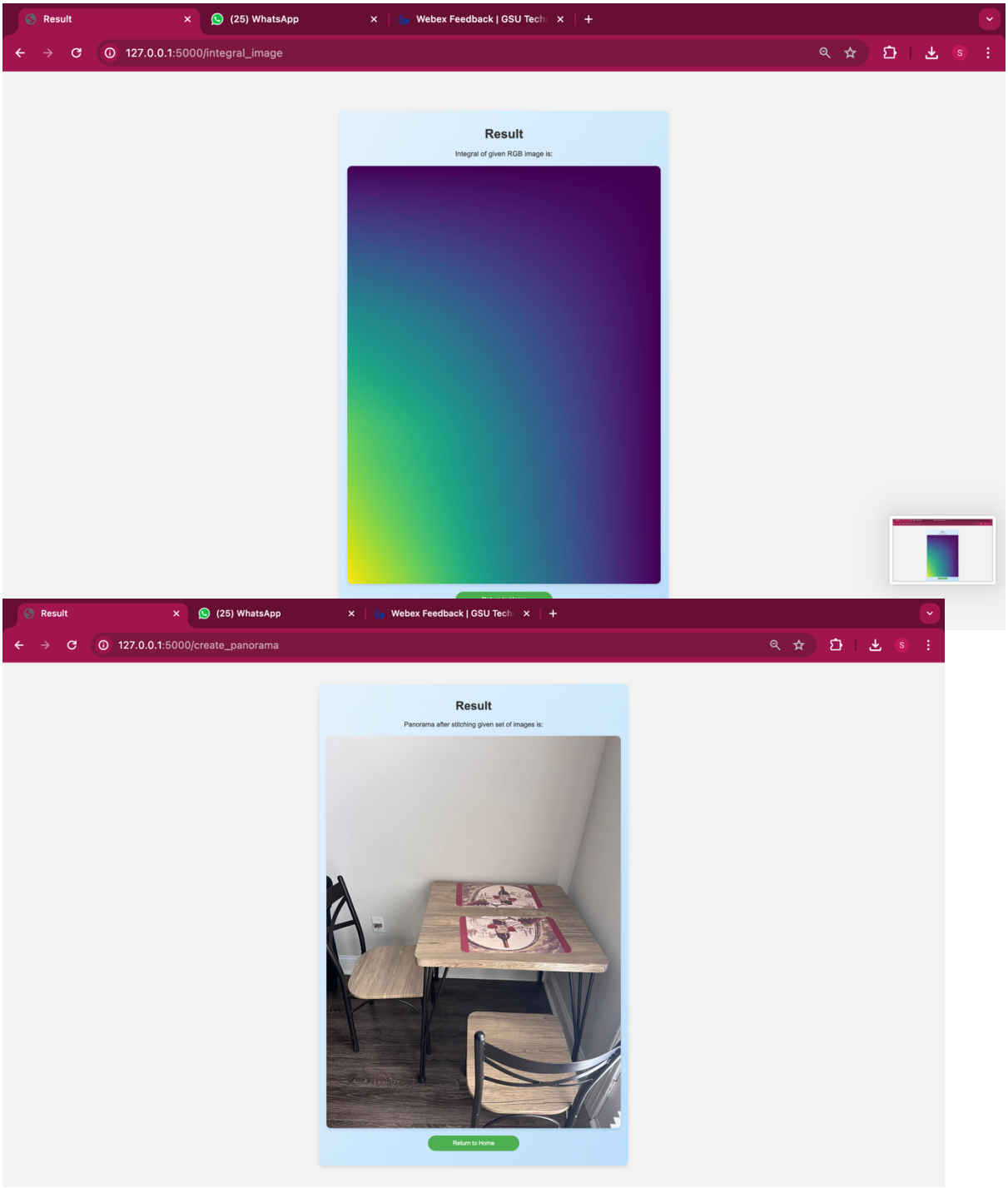




6. Integrate the applications developed for problems 4 and 5 with the web application developed in Assignment 1 problem 4*


CSc 8830: Computer Vision
(25) WhatsApp
Webex Feedback | GSU Tech
127.0.0.1:5000
CSC 8830: VISION IN COMPUTING
Saitejaswi Chakravaram (002762775)
ASSIGNMENT 1 (5): CALCULATE REAL-WORLD DIMENSIONS
CHOOSE A FILE
CALCULATE DIMENSIONS
ASSIGNMENT 2 (4): GENERATE INTEGRAL IMAGE
CHOOSE A FILE
GENERATE INTEGRAL IMAGE
ASSIGNMENT 2 (5): CONSTRUCT PANORAMIC VIEW
SELECT IMAGES
CONSTRUCT PANORAMA


Result
(25) WhatsApp
Webex Feedback | GSU Tech
127.0.0.1:5000/real_dimensions
Result
Diameter of circular object is 10.18 cm.
Return to Home



Github:

https://github.com/05saitejaswi/csc8830/tree/main/Assignment2

Recording link:
https://gsumeetings.webex.com/gsumeetings/ldr.php?RCID=c8f7cdf7e8e4eb7b044470c956b8329f