CSc 8830: Computer Vision

Assignment-2 Report

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.

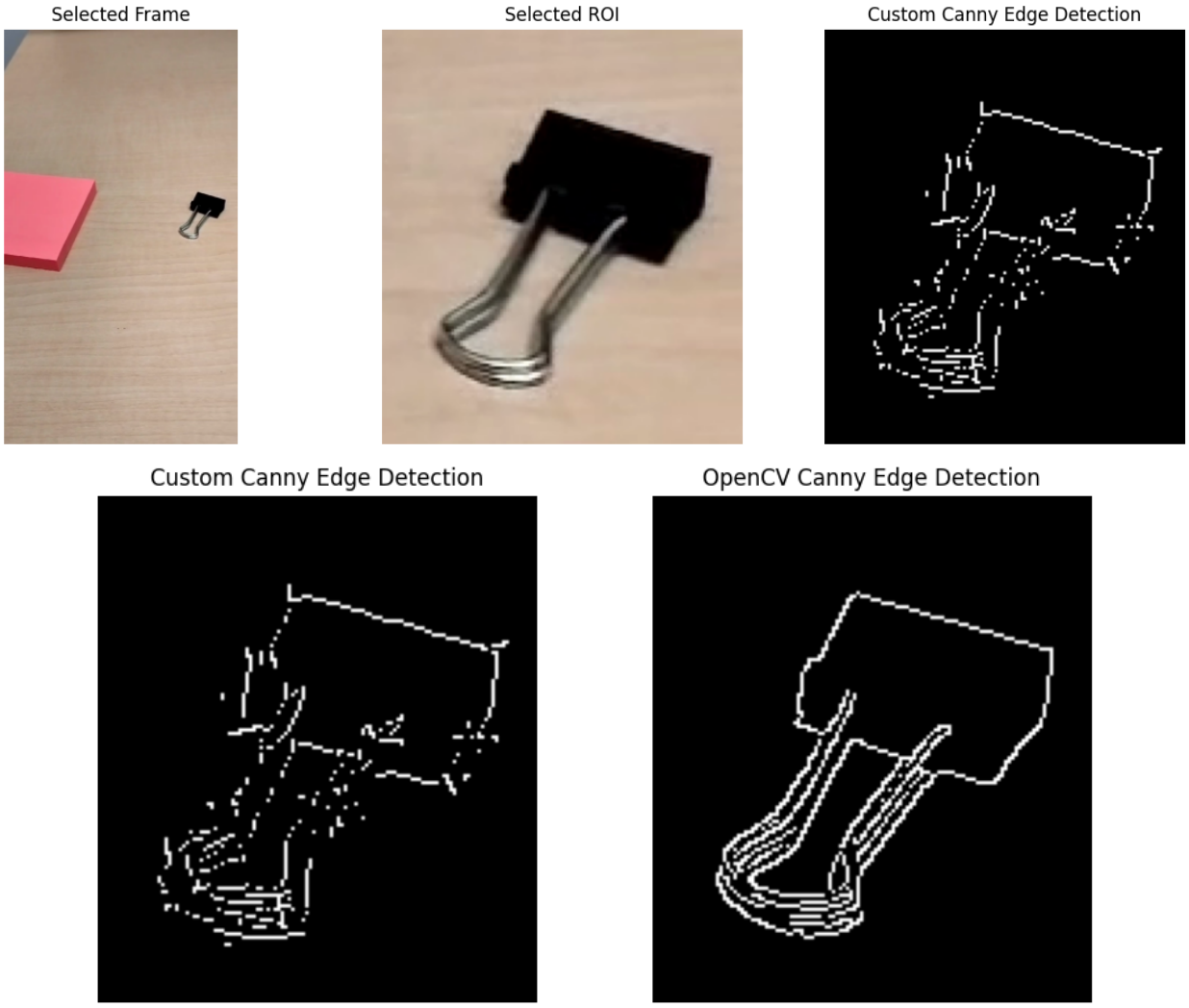
Approach:

Manual Canny Edge Detection Steps:

1. Image Patch Selection:
    - Select a 5x5 image patch containing an edge region from the frame. Ensure that the patch contains clear edges for accurate detection.
2. Grayscale Conversion:
    - Convert the selected image patch to grayscale. This simplifies the edge detection process.
3. Gaussian Smoothing:
    - Apply a Gaussian blur to the grayscale image to reduce noise and unwanted details.
4. Gradient Calculation:
    - Calculate the gradient magnitude and direction using Sobel operators. Compute the gradients in both the x and y directions.
5. Non-maximum Suppression:
    - Suppress non-maximum gradient values to keep only the local maxima as potential edges. Compare the gradient magnitude at each pixel with the magnitudes of its neighbors along the gradient direction.
6. Double Thresholding:
    - Apply double thresholding to classify pixels into strong, weak, and non-edge pixels based on their gradient magnitudes.
    - Pixels with gradient magnitudes above a high threshold are classified as strong edges.
    - Pixels with gradient magnitudes between low and high thresholds are classified as weak edges.
    - Pixels with gradient magnitudes below a low threshold are classified as non-edge pixels and are discarded.
7. Edge Tracking by Hysteresis:
    - Perform edge tracking by hysteresis to link weak edge pixels to strong edge pixels and form continuous edges.

- Start from strong edge pixels and follow the edges by connecting adjacent weak edge pixels that are connected to strong edges.

8. Edge Pixel Identification:
    - Identify the pixels that correspond to the detected edges after edge tracking.
9. OpenCV:
    - Utilize OpenCV's cv2.Canny() function to perform Canny edge detection on the same image patch.
    - Compare the outcome of manual edge detection with OpenCV's result.



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.

Approach:

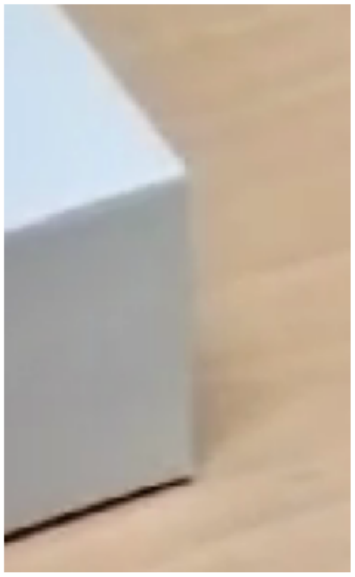
Manual Harris Corner Detection Steps:

1. Image Patch Selection:
    - Select a 5x5 image patch containing a corner region from the frame. Ensure that the patch contains a clear corner for accurate detection.
2. Grayscale Conversion:
    - Convert the selected image patch to grayscale. This simplifies the corner detection process.
3. Calculate Gradients:
    - Compute the gradients in both the x and y directions using Sobel operators or other gradient calculation methods.
4. Structure Tensor Calculation:
    - Compute the structure tensor for each pixel in the image patch. The structure tensor is a matrix that summarizes the local image structure around each pixel.
5. Corner Response Function:
    - Calculate the corner response function for each pixel using the structure tensor. Harris corner detector typically uses the following formula:
        6. R = det(M) - k * (trace(M))^2

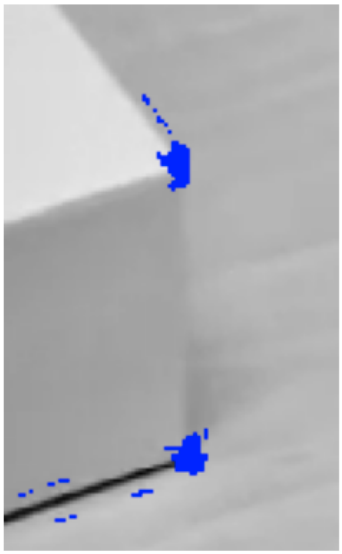
      Where M is the structure tensor, det(M) is the determinant of M, trace(M) is the trace of M, and k is an empirically determined constant (typically 0.04).
7. Thresholding:
    - Apply a threshold to the corner response function to identify potential corner points. Pixels with corner response values above a certain threshold are considered corner candidates.
8. Non-maximum Suppression:
    - Perform non-maximum suppression to retain only the local maxima in the corner response function. This helps to ensure that only the strongest corner points are detected.
9. Corner Point Identification:
    - Identify the pixels that correspond to the detected corners after non-maximum suppression.
10. OpenCV:
    - Utilize OpenCV's cv2.cornerHarris() function to perform Harris corner detection on the same image patch. Compare the outcome of manual corner detection.

Selected Frame

Selected ROI



Manual Harris Corner Detection



CV2 Harris Corner Detection

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.

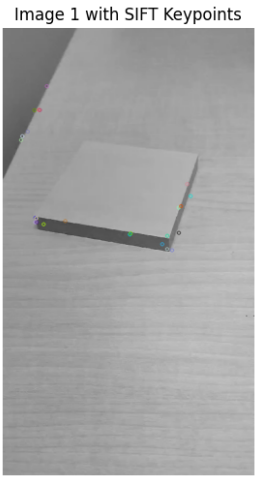
a. Pick a pixel (super-pixel patch as discussed in class) on image 1 and a corresponding pixel ((super-pixel patch as discussed in class)) on image 2 (the pixel on image 2 that corresponds to the same object area on image 1). Compute the SIFT feature for each of these 2 patches. Compute the sum of squared difference (SSD) value between the SIFT vector for these two pixels. Use MATLAB or Python or C++ implementation -- The MATLAB code for SIFT feature extraction and matching can be downloaded from here: https://www.cs.ubc.ca/~lowe/keypoints/ (Please first read the ReadMe document in the folder to find instructions to execute the code).
b. Compute the Homography matrix between these two images using MATLAB or Python or C++ implementation. Compute its inverse.

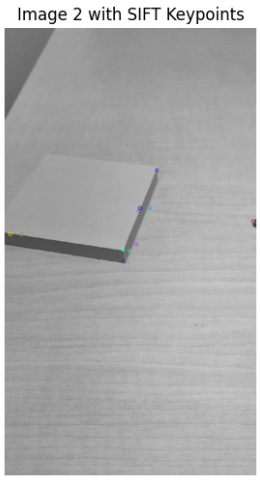
Computing SSD between SIFT Vectors:

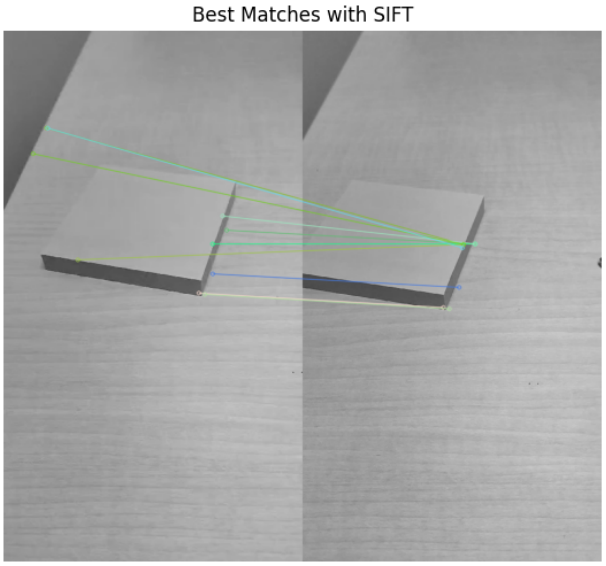
1. Select Image Patches:
    - Choose a pixel or a super-pixel patch in image 1 and its corresponding patch in image 2. Ensure that there is some overlap between the scenes captured in the two images.
2. Extract SIFT Features:
    - Use a SIFT feature extraction algorithm (such as OpenCV's cv2.SIFT or cv2.SURF functions) to extract SIFT descriptors for the selected patches in both images.
3. Compute SIFT Descriptors:
    - Calculate the SIFT descriptors for the selected patches in both images. These descriptors represent distinctive features of the patches.
4. Compute SSD:
    - Compute the sum of squared differences (SSD) between the SIFT descriptors of the corresponding patches in the two images. This involves computing the squared difference between each pair of corresponding elements in the two SIFT descriptors and summing up the squared differences.

Computing Homography Matrix:

1. Feature Matching:
    - Use a feature matching algorithm (such as OpenCV's cv2.FlannBasedMatcher or cv2.BFMatcher) to find matches between the SIFT keypoints detected in both images.
2. Filter Matches:
    - Apply a filter to remove unreliable matches, such as using a distance threshold or ratio test.
3. Estimate Homography:
    - Use a robust estimation method (such as RANSAC) to estimate the homography matrix that transforms points from the perspective of one image to the other.
4. Compute Inverse Homography:
    - Calculate the inverse of the estimated homography matrix. This inverse matrix can be used to transform points from the perspective of image 2 to the perspective of image 1.







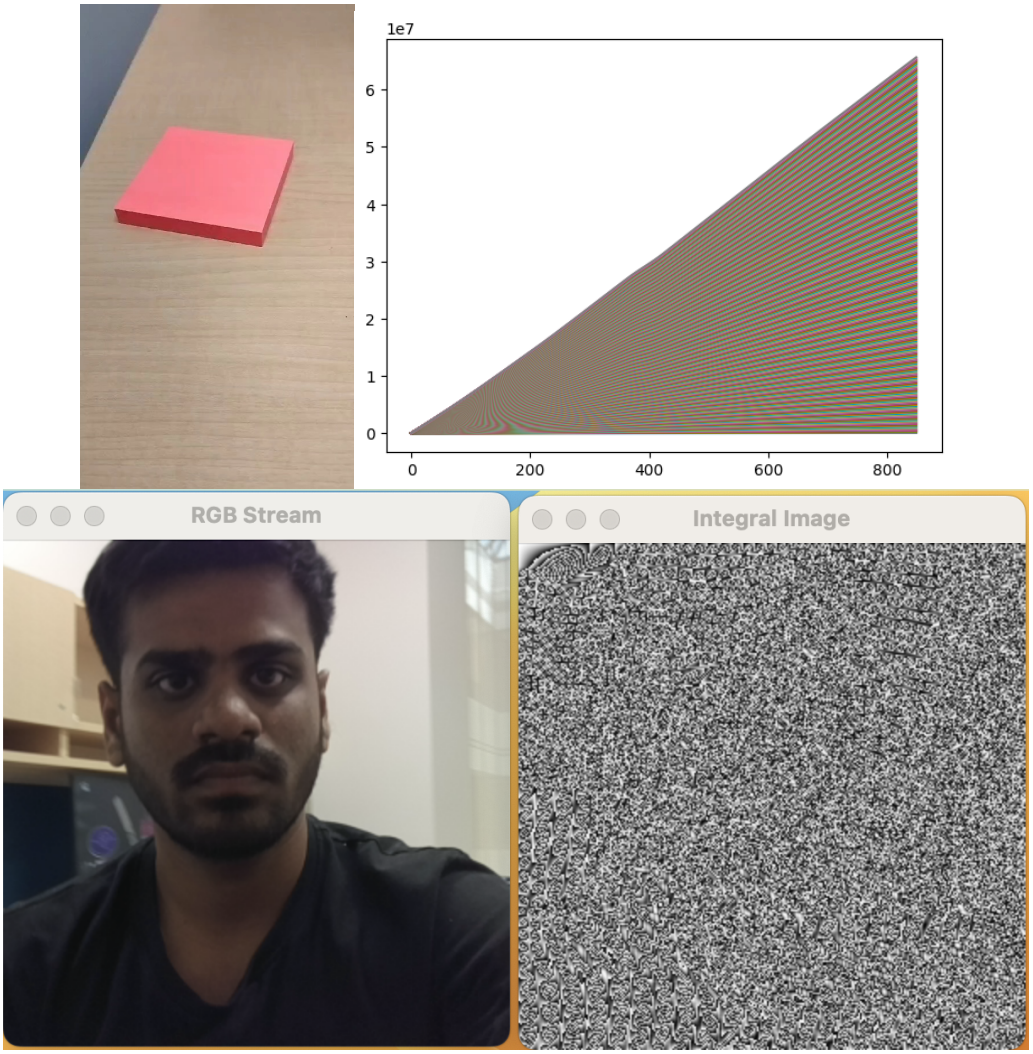

```
Sum of Squared Difference (SSD) between SIFT vectors: 4455.0
Two frames with at least 2-second difference were selected.
Homography Matrix:
[[-5.15969970e-01 -2.28519209e-01  2.56823547e+02]
 [-6.86768868e-01 -3.04164752e-01  3.41838531e+02]
 [-2.00904464e-03 -8.89790718e-04  1.00000000e+00]]

Inverse of Homography Matrix:
[[-1.36669475e+16  1.28836071e+16 -8.94119400e+17]
 [-2.54893927e+16  3.27274663e+16 -4.64123278e+18]
 [-5.01377325e+13  5.50043375e+13 -5.92605163e+15]]
```

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)”

Approach

1. Read RGB Frames: Continuously capture frames from the RGB camera.
2. Convert to Grayscale: Convert each RGB frame to grayscale.
3. Compute Integral Image: Iterate over each pixel in the grayscale frame and compute the integral image pixel value based on the sum of pixels above and to the left of the current pixel.
4. Store Integral Image: Store the computed integral image.



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*

127.0.0.1

CSc 8830: Computer Vision

Teja Krishna Cherukuri (002763290)

Assignment 1 (5): Find Real-world Dimensions

Choose a file

Compute Dimensions

Assignment 2 (4): Compute Intergral Image

Choose a file

Integral Image

Assignment 2 (5): Create Panorama

Choose images

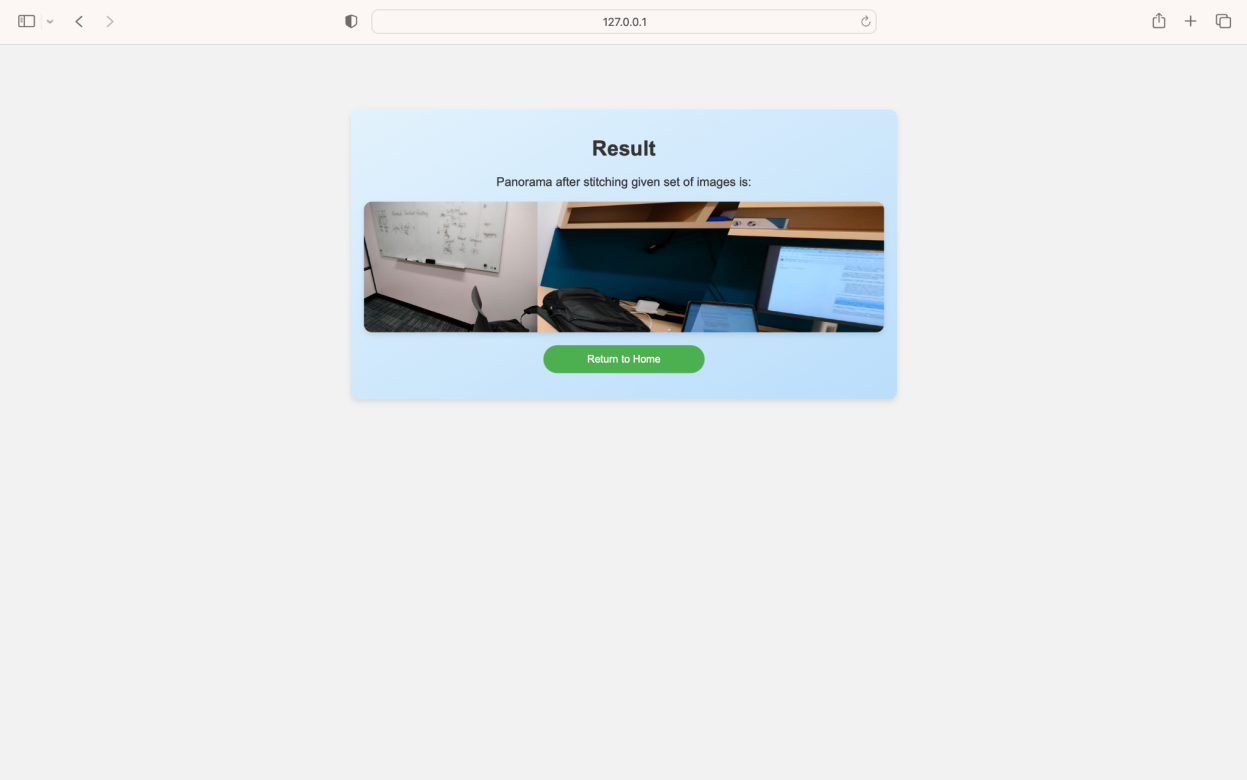
Create Panorama

127.0.0.1

Result

Integral of given RGB image is:

Return to Home



Repo: https://github.com/TejaCherukuri/CSC8830_ComputerVision/tree/main/Assignment-2