

CSC 8830: CV Assignment-3

SAI TEJASWI CHAKRAVARARAM

002762775

1. 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. Pick any image frame from the 10 sec video footage. Pick a region of interest corresponding to an object in the image. Crop this region from the image. Then use this cropped region to compare with randomly picked 10 images in the dataset of 10 sec video frames, to see if there is a match for the object in the scenes from the 10 images. For comparison use sum of squared differences (SSD) or normalized correlation.

A 10 sec video has been taken from a mobile phone camera

Steps Involved:

1. Setup the Camera: Set up your camera on a stable platform or hold it securely by hand.
2. Record Video: Start recording the video with the camera. Ensure that the camera captures the scene with a slight left-right or right-left panning motion during the 10-second duration.
3. Save Video: Once the recording is complete, save the video footage to your device.
4. Pick Image Frame: Select any frame from the recorded video footage that contains the object or region of interest you want to analyze.
5. Crop Region of Interest: Manually select and crop the region of interest from the selected image frame. Ensure that the cropped region contains the object you want to compare.
6. Compare with Dataset: Randomly pick 10 images from the dataset of 10-second video frames. For each selected image, perform the following steps:
 - o • Crop the same region of interest as in step 5 from the selected image.
 - o • Compute the sum of squared differences (SSD) or normalized correlation between the cropped region of interest from the selected image frame and the cropped

region of interest from the dataset image.

- • Store the computed similarity metric (SSD or normalized correlation value) for comparison.



2. Solve the following by hand (on paper or typed: Do not just copy it from literature)

(a) Derive the motion tracking equation from fundamental principles. Select any 2 consecutive frames from the set from problem 1 and compute the motion function estimates.

(b). Derive the procedure for performing Lucas-Kanade algorithm for motion tracking when the motion is known to be affine: $u(x,y) = a_1*x + b_1*y + c_1$; $v(x,y) = a_2*x + b_2*y + c_2$ (the numbers are subscripts, not power)

Assignment 3, Computer vision
csc8830 Saileshwari chakravarthy
002762775

2)

a) Derive the motion tracking equation from fundamental principle. Select 2 consecutive frames from the set from problem 1 and compute the motion function estimates

We have taken 2 equations which will make the problem constrained so that it will be solved; we use 2 equations $I(x+\delta x, y+\delta y, t+\delta t) = I(x, y, t)$, $\delta x, \delta y$ and δt are small.

$$\textcircled{1} \quad I(x+\delta x, y+\delta y, t+\delta t) = I(x, y, t)$$

For the point (x, y) in the image at frame at frame point t , the intensity, brightness/pixel values does not change

when it is moved by $(\delta x, \delta y)$ later at frame $t+\delta t$.

Taylor series explains: when $f(x+\delta x) = f(x) + \frac{\partial f}{\partial x} \delta x +$

$$\frac{\delta^2 f}{\delta x^2} \frac{\delta x^2}{2!} + \dots + \frac{\delta^n f}{\delta x^n} \frac{\delta x^n}{n!} \quad \text{where } \delta x \text{ is small}$$

$$f(x+\delta x) = f(x) + \frac{\delta f}{\delta x} \delta x$$

Here is the next one

\textcircled{11} $\delta x, \delta y$ and δt are small

$$\Rightarrow s(x+\delta x, y+\delta y, t+\delta t)$$

$$= I(x, y, t) + \underbrace{\frac{\delta I}{\delta x} \delta x}_{I_x} + \underbrace{\frac{\delta I}{\delta y} \delta y}_{I_y} + \underbrace{\frac{\delta I}{\delta t} \delta t}_{I_t}$$

Substituting \textcircled{1} from \textcircled{11}

$$I_x \delta x + I_y \delta y + I_t \delta t = 0$$



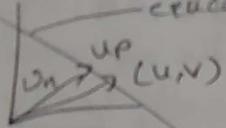
Scanned with OKEN Scanner

dividing by δt and taking the limits $\delta t \rightarrow 0$:

$$I_x \frac{\partial x}{\partial t} + I_y \frac{\partial y}{\partial t} + I_t = 0 \Rightarrow I_x u + I_y v + I_t = 0.$$

$\hookrightarrow \frac{\partial x}{\partial t}, \frac{\partial y}{\partial t}$ = velocity of the point, i.e. optical flow

The equation of points (u, v) is a straight line
we already know that \vec{u}_A, \vec{u}_P can be anything \vec{u}_n is our estimation
equation $I_x u + I_y v + I_t = 0$



u_n , the normal optical flow vector can be computed as magnitude: $\frac{|Pt|}{\sqrt{I_x^2 + I_y^2}}$ direction: $\frac{(R_x, R_y)}{\sqrt{I_x^2 + I_y^2}}$

2b) Derive the procedure for performing Lucas-Kanade algorithm for motion tracking when the motion is known to be affine
 $v(x,y) = a_1 x + b_1 y + c_1$; $v(x,y) = a_2 x + b_2 y + c_2$ (the numbers are subscripts, not power)

When the motion is affine, i.e. representable by translation, rotation, etc. Lucas-Kanade algorithm works effectively for motion tracking. As the optical flow is underconstrained problem, the Lucas-Kanade algorithm assumes that the optical flow, motion field within a small window is the same for all the pixels and further uses this to solve the optical flow estimation problem for all the points (i,j) in the window.

$$I_x(i,j)u + I_y(i,j)v + I_t(i,j) = 0$$

for all window size of $m \times m$

$$\begin{bmatrix} I_x^{(1,1)} & I_y^{(1,1)} \\ I_x^{(m,m)} & I_y^{(m,m)} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t^{(1,1)} \\ I_t^{(m,m)} \end{bmatrix}$$

known ($m^2 \times 2$) unknown (2×1) known ($m^2 + 1$)

general Lucas Kanade equations

As the motion is assumed to be affine we can write the general form above for each pixel (i,j) in w as:

$$\begin{bmatrix} I_x^{(1,1)} & I_y^{(1,1)} & 1 \\ I_x^{(i,i)} & I_y^{(i,i)} & 1 \\ I_x^{(m,m)} & I_y^{(m,m)} & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} I_t^{(1,1)} \\ \vdots \\ I_t^{(m,m)} \end{bmatrix}$$

Similar equation for a_2, b_2, c_2

$$\begin{bmatrix} I_x^{(1,1)} & I_y^{(1,1)} & 1 \\ I_x^{(i,i)} & I_y^{(i,i)} & 1 \\ I_x^{(m,m)} & I_y^{(m,m)} & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} I_t^{(1,1)} \\ \vdots \\ I_t^{(m,m)} \end{bmatrix}$$

These system of equations are solvable via atleast with least squares. Once $a_1, b_1, c_1, a_2, b_2, c_2$ are obtained, $v(x,y), v(x,y)$ are computable for e.g.

3. Fix a marker on a wall or a flat vertical surface. From a distance D , keeping the camera stationary (not handheld and mounted on a tripod or placed on a flat surface), capture an image such that the marker is registered. Then translate the camera by T units along the axis parallel to the ground (horizontal) and then capture another image, with the marker being registered. Compute D using disparity based depth estimation in stereo-vision theory. (Note: you can pick any value for D and T . Keep in mind that T cannot be large as the marker may get out of view. Of course this depends on D)

Steps Involved:

1. **Setup:** Fix a marker on a flat vertical surface and ensure it is well-lit and clearly visible.

Place the camera on a tripod or a stable surface facing the marker.

2. **Calibration:** Calibrate the stereo camera setup to accurately estimate depth. This

involves determining the intrinsic and extrinsic parameters of each camera in the stereo pair. You can use calibration patterns like checkerboards and dedicated calibration software.

3. Capture Images:

- • Capture an image of the marker with the camera stationed at a distance D from the marker.
- • Translate the camera horizontally by T units along the axis parallel to the ground.
- • Capture another image of the marker from the new position.

4. Depth Estimation:

- • Rectify the stereo images to ensure corresponding points in the two images lie on the same scanline.
- • Compute the disparity map between the rectified stereo images. Disparity represents the horizontal shift of image pixels between the left and right images and is inversely proportional to depth.
- • Use the disparity map and stereo calibration parameters to compute the depth map, which represents the distance of each pixel from the camera.

5. Compute Distance:

- • Select corresponding points on the marker in both images. These points should have matching features in both images.
- • Compute the average disparity value for these corresponding points in the disparity map.
- • Use the stereo baseline (distance between the camera centers) and the focal length of the cameras to compute the distance D to the marker using the formula:

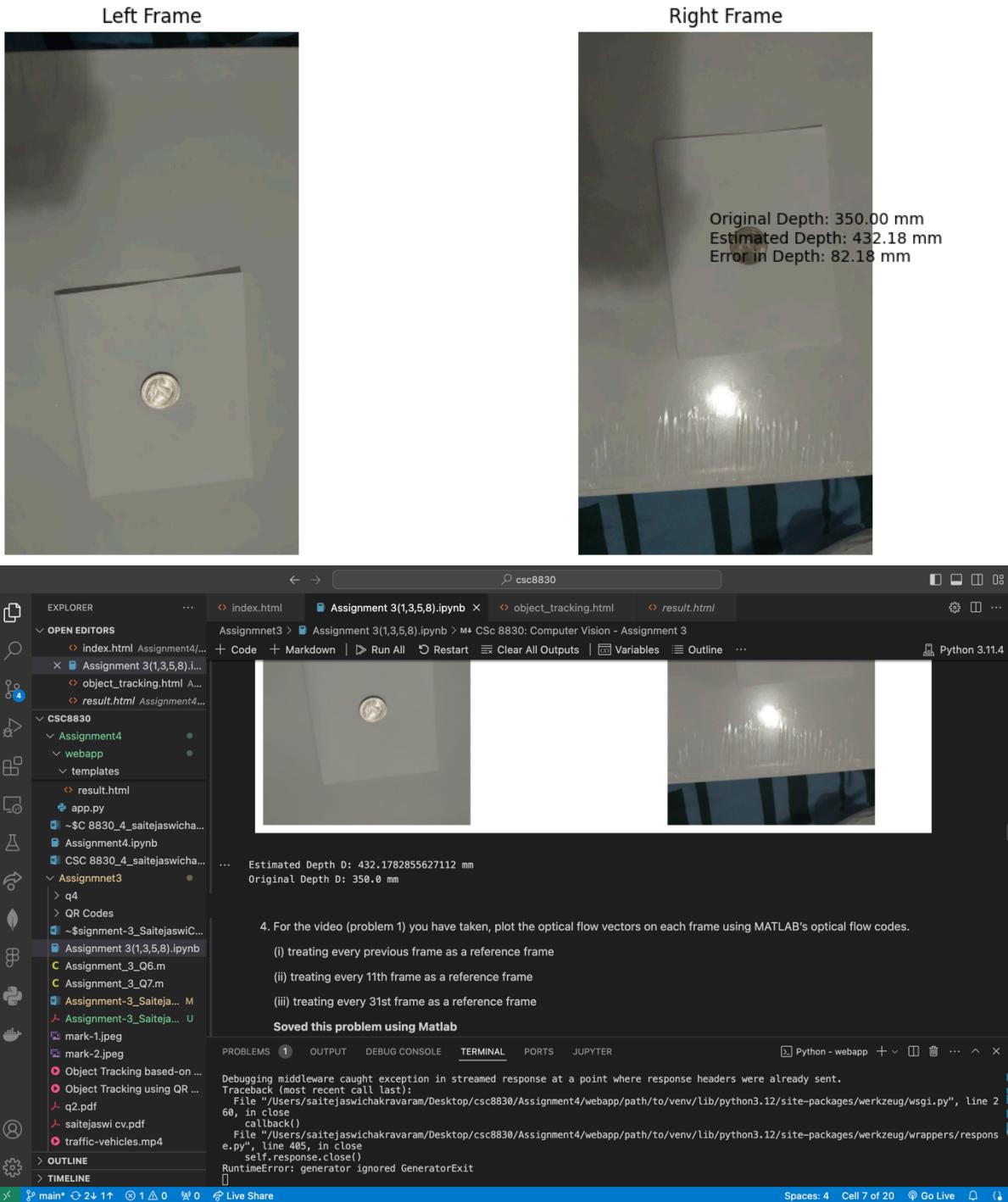
6.

Result: The computed distance D represents the distance between the camera and the marker.

$$D = \frac{f \cdot B}{\text{average disparity}}$$

where:

- f is the focal length of the camera.
- B is the baseline distance between the camera centers.



4. For the video (problem 1) you have taken, plot the optical flow vectors on each frame using MATLAB's optical flow codes. (i) treating every previous frame as a reference

frame (ii) treating every 11th frame as a reference frame (iii) treating every 31st frame as a reference frame

Scenario (i): Treating every previous frame as a reference frame

1. Load the video into MATLAB.
2. Initialize an optical flow object using the opticalFlow function.
3. Loop through each frame of the video.
4. Compute optical flow between the current frame and the previous frame using the estimateFlow method of the optical flow object.
5. Plot the optical flow vectors on the current frame using the plot method of the optical flow object.
6. Repeat steps 4-5 for each frame.

Scenario (ii): Treating every 11th frame as a reference frame

1. Load the video into MATLAB.
2. Initialize an optical flow object using the opticalFlow function.
3. Loop through every 11th frame of the video.
4. Compute optical flow between the current frame and the 10th previous frame (as a reference frame) using the estimateFlow method of the optical flow object.
5. Plot the optical flow vectors on the current frame using the plot method of the optical flow object.
6. Repeat steps 3-5 for each 11th frame.

Scenario (iii): Treating every 31st frame as a reference frame

1. Load the video into MATLAB.
2. Initialize an optical flow object using the opticalFlow function.
3. Loop through every 31st frame of the video.
4. Compute optical flow between the current frame and the 30th previous frame (as a reference frame) using the estimateFlow method of the optical flow object.
5. Plot the optical flow vectors on the current frame using the plot method of the optical flow object.
6. Repeat steps 3-5 for each 31st frame.

This question is solved using Matlab

The screenshot shows the MATLAB Live Editor interface. The code in the editor is:

```
Assignment 3 - Question 4 - saitejaswi  
chakravaram - 002762775  
Reading the video:  
vidReader = VideoReader('./video.mp4','CurrentTime',  
1  
Specifying the optical flow estimation method as  
opticalFlowHS.  
opticFlow = opticalFlowHS;  
1. Plotting the optical flow vectors when treating every previous  
frame as a reference frame.  
h = figure;  
movegui(h);  
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title',  
hPlot = axes(hViewPanel);  
  
while hasFrame(vidReader)  
    frameRGB = readFrame(vidReader);  
    frameGray = im2gray(frameRGB);  
    flow = estimateFlow(opticFlow,frameGray);  
    imshow(frameRGB)  
    plot(flow,'DecimationFactor',[5 5],'ScaleFactor',  
    hold on  
    plot(flow,'DecimationFactor',[5 5],'ScaleFactor',  
    hold off  
    pause(10^-3)  
end  
opticFlow.reset();  
vidReader = VideoReader('./video.mp4','CurrentTime',  
2  
Plot of Optical Flow Vectors  
Plot of Optical Flow Vectors
```

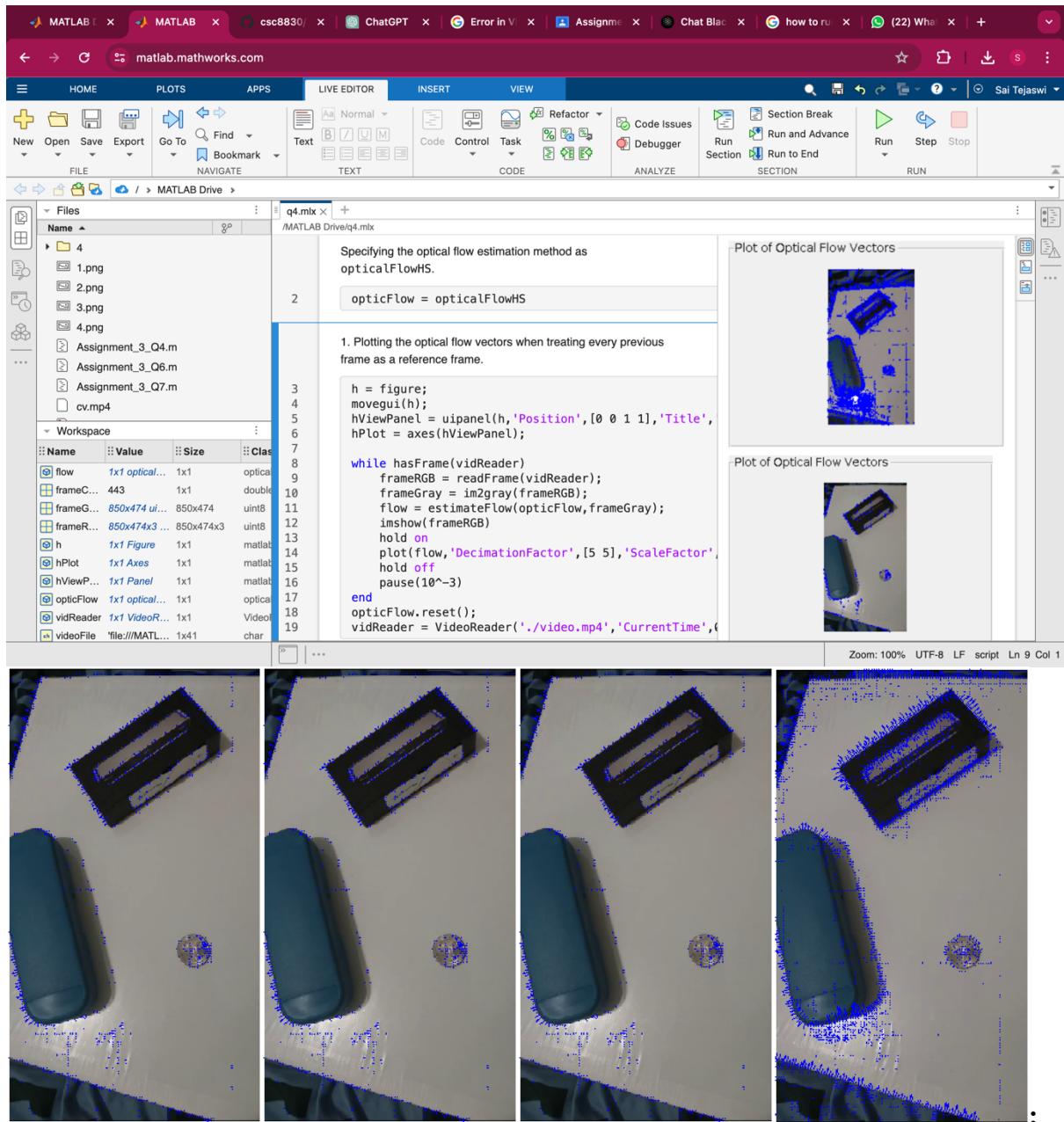
Two plots titled "Plot of Optical Flow Vectors" are displayed on the right, showing blue vector fields over grayscale images of objects.

The screenshot shows the MATLAB Live Editor interface. The code in the editor is identical to the one in the first screenshot, but the workspace table now includes the variable "flow". The plots on the right remain the same.

Specifying the optical flow estimation method as
opticalFlowHS.
opticFlow = opticalFlowHS;
1. Plotting the optical flow vectors when treating every previous
frame as a reference frame.
h = figure;
movegui(h);
hViewPanel = uipanel(h,'Position',[0 0 1 1],'Title',
hPlot = axes(hViewPanel);

while hasFrame(vidReader)
 frameRGB = readFrame(vidReader);
 frameGray = im2gray(frameRGB);
 flow = estimateFlow(opticFlow,frameGray);
 imshow(frameRGB)
 plot(flow,'DecimationFactor',[5 5],'ScaleFactor',
 hold on
 plot(flow,'DecimationFactor',[5 5],'ScaleFactor',
 hold off
 pause(10^-3)
end
opticFlow.reset();
vidReader = VideoReader('./video.mp4','CurrentTime',
2
Plot of Optical Flow Vectors
Plot of Optical Flow Vectors

11 usages of "vidReader" found



Matlab:

5. Run the feature-based matching object detection on the images from problem (1).

MATLAB (not mandatory for this problem) Tutorial for feature-based matching object detection is available here: <https://www.mathworks.com/help/vision/ug/object-detection-in-a-cluttered-scene-using-point-feature-matching.html>

1. **Load Images:** Load the images from problem (1) into Python using OpenCV's cv2.imread() function.

2. **Feature Detection:** Detect keypoint features in the images using a feature detection algorithm such as ORB, SIFT, or SURF. For example, you can use the ORB (Oriented FAST and Rotated BRIEF) algorithm, which is efficient and free to use.
3. **Feature Description:** Compute descriptors for the detected keypoints to describe their local image regions. This step is essential for matching keypoints between images.
4. **Feature Matching:** Match keypoints between pairs of images using a feature matching algorithm. OpenCV provides functions like cv2.BFM Matcher() for brute-force matching or cv2.FlannBasedMatcher() for FLANN-based matching.
5. **Filter Matches:** Apply a matching ratio test or other filtering techniques to eliminate incorrect matches and retain only reliable matches.
6. **Draw Matches:** Draw the matched keypoints on the images to visualize the detected objects and their corresponding matches.
7. **Object Detection:** Determine the presence and location of the object of interest in the scene based on the matched keypoints.

This is Solved using Python



6. Refer to the Bag of Features example MATLAB source code provided in the classroom's classwork page. In your homework, pick an object category that would be commonly seen in any household (e.g. cutlery) and pick 5 object types (e.g. for cutlery pick spoon, fork, butter knife, cutting knife, ladle). Present your performance evaluation.

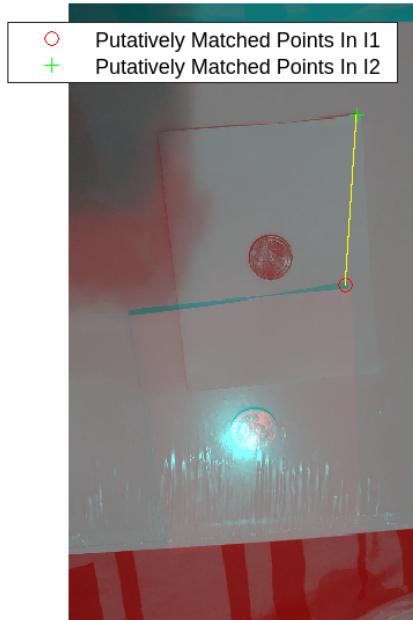
Followed the steps directly provided in the tutorial.

7. Repeat the image capture experiment from problem (3), however, now also rotate (along the ground plane) the camera 2 (right camera) towards camera 1 position, after translation by T . Make sure the marker is within view. Note down the rotation angle. Run the tutorial provided for uncalibrated stereo rectification in here:

<https://www.mathworks.com/help/vision/ug/uncalibrated-stereo-image-rectification.html> (MATLAB is mandatory for this exercise). Exercise this tutorial for the image pairs you have captured. You can make assumptions as necessary, however, justify them in your answers/description. (Note: you can print out protractors from any

online source and place your cameras on that when running experiments:
<http://www.ossmann.com/protractor/conventional-protractor.pdf>.

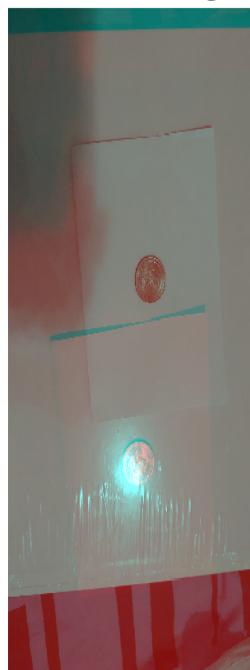
Followed the steps directly provided in the tutorial. Rotation Angle: 15 degrees



Thirty Strongest SURF Features In I2



Composite Image (Red - Left Image, Cyan - Right Image)



Thirty Strongest SURF Features In I2



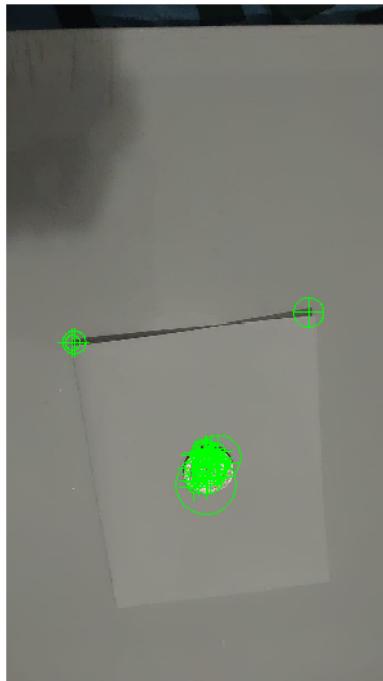
I1 (left); I2 (right)



Thirty Strongest SURF Features In I2



Thirty Strongest SURF Features In I1



```
q7.mlx qq.mlx
Command Window
Encoding images using Bag-Of-Features.
* Encoding an image...done.
Accuracy: 1
>>
```

8. Implement a real-time object tracker (two versions) that (i) uses a marker (e.g. QR code or April tags), and (ii) does not use any marker and only relies on the object

1. With QR Marker:

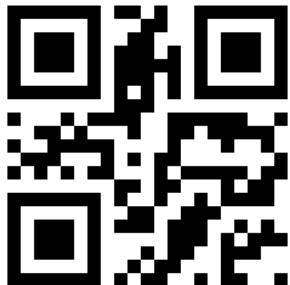
- This version makes use of the cv2.QRCodeDetector() class to identify and decode QR codes from the camera feed. When a QR code is recognised, it shows the decoded data and surrounds it with rectangles.

2. No Marker:

This version tracks objects by updating bounding box centre points and giving each one a unique ID. It employs background subtraction for object recognition.

- Every identified item is symbolised by a bounding box bearing an ID, exhibited on

the frame. • Items are monitored according to their movements over successive frames.



Repo: <https://github.com/05saitejaswi/csc8830/tree/main/Assignment3>

Webex meeting recording: schakravaram1@student.gsu.edu's meeting-20240423 0040-3

Recording link:

<https://gsumeetings.webex.com/gsumeetings/ldr.php?RCID=31de2d12c2f05ea9d388fa0dcf6c2dce>