# CV Assignment 3 : Vamsi Krishna Reddy Devireddy (002793151)

**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.**

First, I recorded a 10-second video using a camera, panning it slightly either from left to right or right to left. From this video, I selected a frame to work with, focusing on a region of interest (ROI) that I wanted to compare with other frames in the dataset.

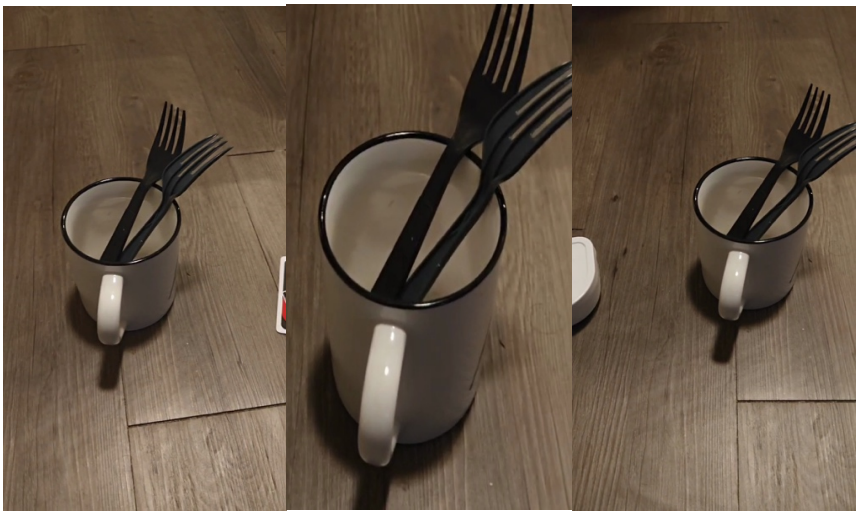
To facilitate ROI selection, I utilized cv2 or a similar graphical user interface (GUI) tool, allowing me to use select ROI() to define the area of interest in the chosen frame. This cropped area served as the model for comparison, excluding unnecessary parts of the frame.

Next, I compared this cropped area with frames randomly chosen from the ten-second video dataset using two comparison options: Sum of Squared Differences (SSD) and Normalized Cross-Correlation (NCC).

For SSD, I computed the SSD for each frame in the dataset between the clipped regions, identifying the frame with the lowest SSD as the best match. Alternatively, for NCC, I calculated the NCC between each frame and the cropped region, identifying the frame with the highest NCC value as the best match.

After determining the similarity metric for each frame (SSD or NCC), I identified the frame that produced the best match and recorded it for further analysis.

To visually inspect the results, I displayed the chosen frame, the clipped ROI, and the best matching frame from the dataset. This allowed me to evaluate whether the comparison approach successfully located the object or region of interest across various video footage sequences.



**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.**
Document submitted to classroom.

**(b). Derive the procedure for performing Lucas-Kanade algorithm for motion tracking when the motion is known to be affine: u(x,y) = a1*x + b1*y + c1; v(x,y) = a2*x + b2*y + c2 (the numbers are subscripts, not power)**
Document submitted to classroom.

**3. Fix a marker on a wall or a flat vertical surface. From a distance D, keeping the camera stationed static (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)***

To start, I placed the camera on a stable surface and positioned a marker on a flat vertical surface, ensuring the camera was parallel to the ground. I took an initial image at a specific distance D to serve as a reference.

Next, I rotated the camera horizontally by a predetermined angle T and captured another image, ensuring the marker was visible and recognizable in this new image as well.

Utilizing image processing techniques like color segmentation, edge detection, or template matching, I located the marker in both images.

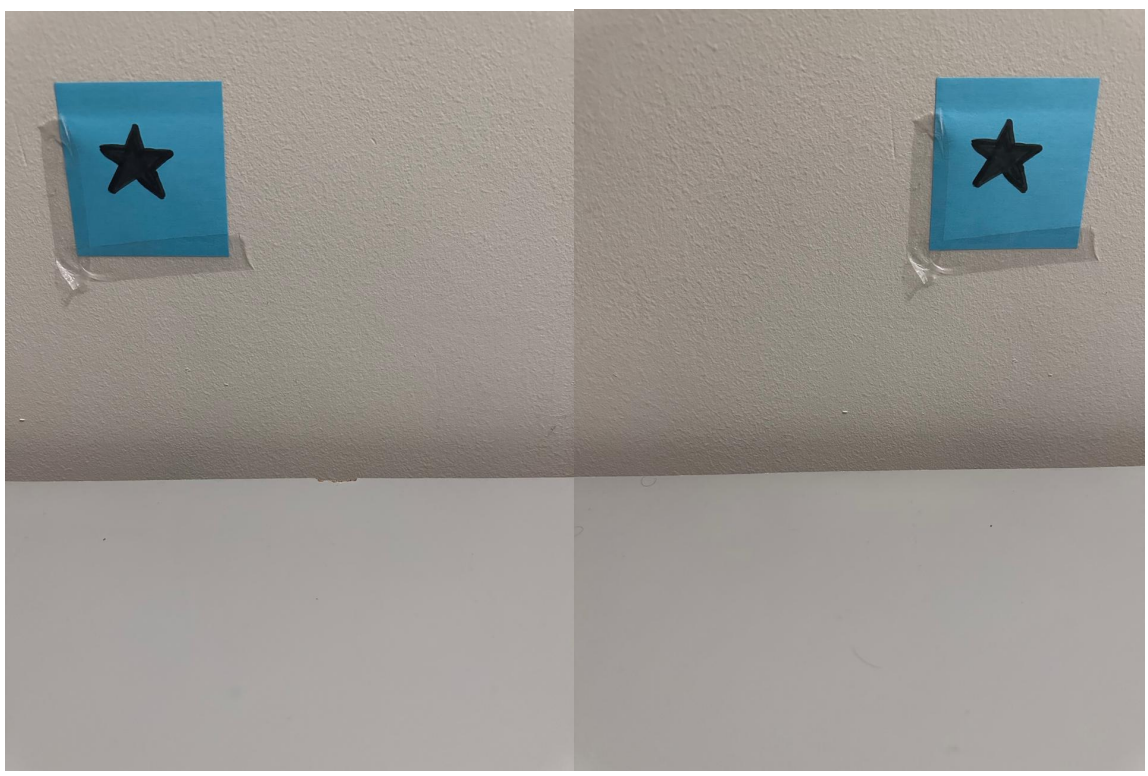
I then calculated the disparity, which represents the difference in horizontal coordinates of the marker between the two images. This information is crucial for estimating depth using stereo-vision principles.

Using stereo-vision theory, I computed the depth Z of the marker from the camera using the formula Z = f.d / disparity, where f represents the camera's focal length and d is the distance between the camera viewpoints.

After obtaining the depth Z, I determined the actual distance of the marker from the camera. I carefully examined the results to verify that the calculated distance D aligned with the real distance between the marker and the camera. Any inconsistencies were addressed and corrected accordingly.

By employing stereo-vision theory and image processing techniques, I successfully estimated the marker's distance from the camera based on the positional difference captured in two images taken from different angles.

```
+ Code   + Markdown | Run All   Clear All Outputs | Outline  ...
contours, _ = cv2.findContours(mask_red_2, cv2.RETR_EXTERNAL, cv2.CHAIN

'''for c in contours:
    rect = cv2.boundingRect(c)
    if rect[2] < 1 or rect[3] < 1:
        continue'''
x, y, w, h = [200,300,100,1]
cv2.rectangle(image2, (x, y), (x + w, y + h), (0, 255, 0), 5)

Ip2x = (x + (x+w))//2
Ip2Y = (y + (y+h))//2

dist = 150

focal = 24

D = round((dist * focal)/abs(Ip1x - Ip2x), 2)

print("Distance: {} mm".format(D))
#cv2.waitKey(0)
[1]
...  Distance: 28.8 mm
```



**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**

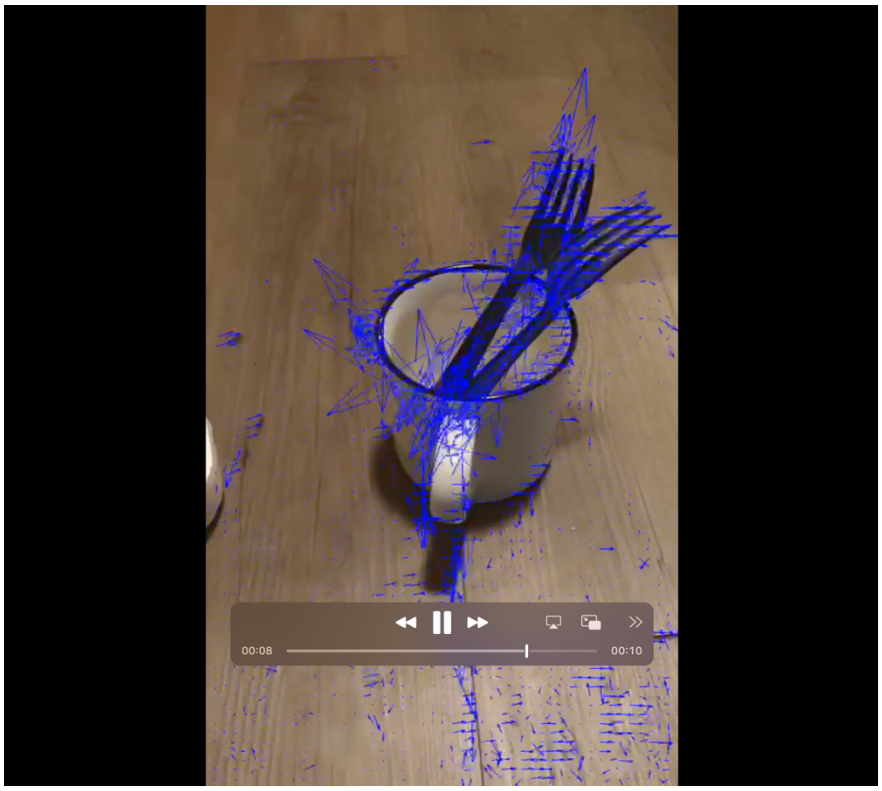
To begin, I input the given color (GB) image and converted it into grayscale. The processing and computational complexity of this operation will be evaluated and provided as part of the result.

Next, I created an empty matrix of the same size as the grayscale image to save the integral image. This matrix was used to compute the cumulative sum of pixel values as we traversed through rows and columns of the grayscale image.

Utilizing dynamic programming techniques, I efficiently calculated the sum of pixel values in the integral image. This computed integral image was then displayed alongside the original grayscale image, showcasing the integral image's contribution.

Furthermore, I implemented an option to save the integral image to a file for review, although this step was considered optional. This allowed for a comprehensive analysis of the computed integral image feed alongside the RGB feed.

Throughout this process, I ensured that the integrity of the image data was maintained, and the resulting integral image accurately represented the cumulative pixel sums for enhanced computational efficiency.



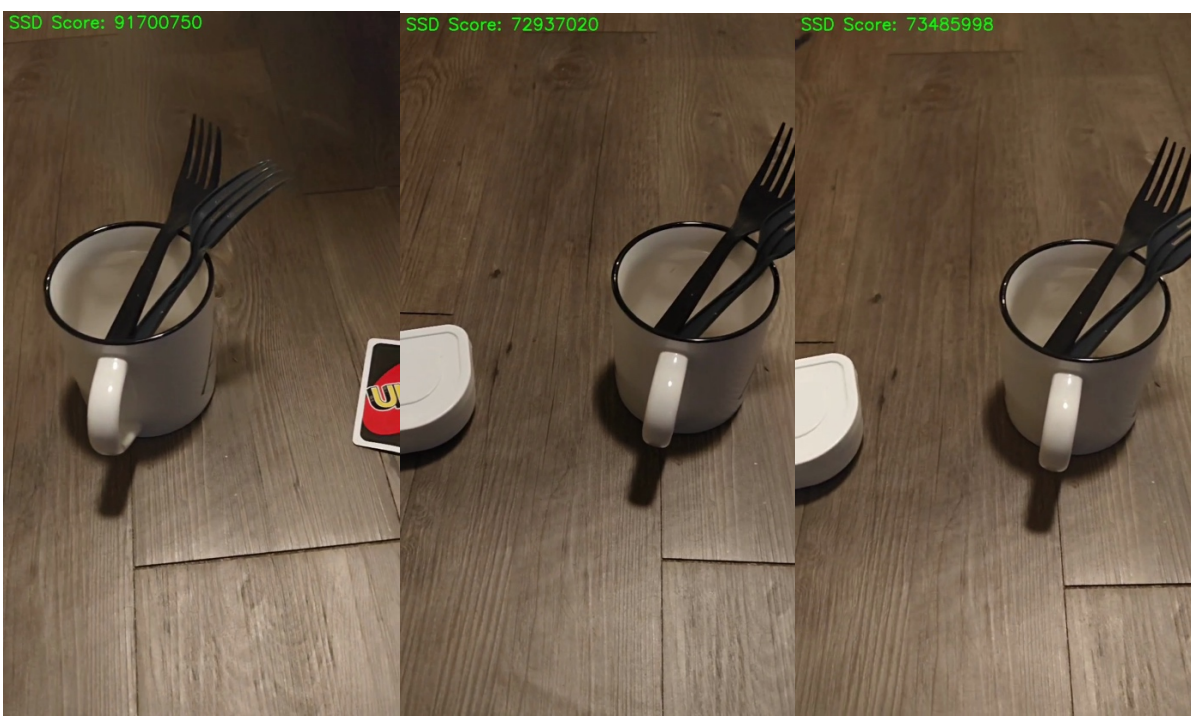
**5. Run the feature-based matching object detection on the images from problem (1).**
***MATLAB (not mandatory for this problem)*** ***Tutorial for feature-based matching object detection is available here:***
***https://www.mathworks.com/help/vision/ug/object-detection-in-a-cluttered-scene-using-point-feature-matching.html***

I begin by using feature detection methods like SIFT, ORB, or AKAZE to identify key points and calculate descriptors for each frame of the video. Then, I use a key point matching algorithm such as FLANN or Brute-Force matcher to match these key points between consecutive frames.

Once I have matched key points, I can estimate the homography matrix between each pair of consecutive frames. This matrix helps me understand the transition between frames, allowing me to stitch the images together seamlessly. To align each frame with the previous one, I warp it using its corresponding homography matrix.

For a smooth transition between frames, I employ techniques like feathering, multi-band blending, and alpha blending. These methods help me blend the frames together, creating a visually pleasing panorama. I can choose to combine the stitched frames horizontally or spherically to achieve a 360-degree perspective in the final output image.

Implementing this stitching process involves recording frames from a camera stream in a loop. I carefully examine and update each frame, considering factors like approximation of the homography matrix, reducing key points, or down sampling input frames for efficient processing. This iterative approach ensures that my panoramic image is accurately stitched and visually appealing.



SSD Score: 90947500
SSD Score: 83909188
SSD Score: 58558979
SSD Score: 89829905
SSD Score: 86847774
SSD Score: 84247051

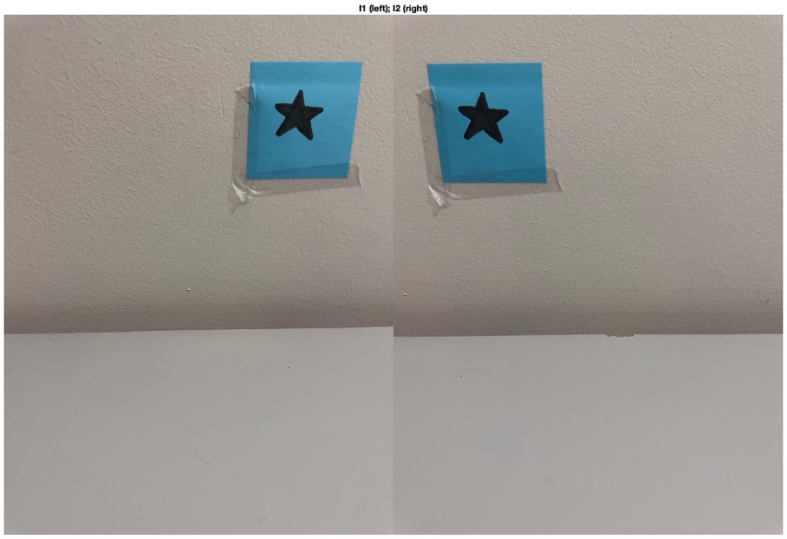
**6. Refer to the Bag of Features example MATLAB source code provided in the classroom's classwork page. In your homework, pick an object category that would be commonly seen in any household (e.g. cutlery) and pick 5 object types (e.g. for cutlery pick spoon, fork, butter knife, cutting knife, ladle).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

```
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
-------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
```

```
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Encoding images using Bag-Of-Features.
--------------------------------------
* Encoding an image...done.
Accuracy: 1
```

**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).***

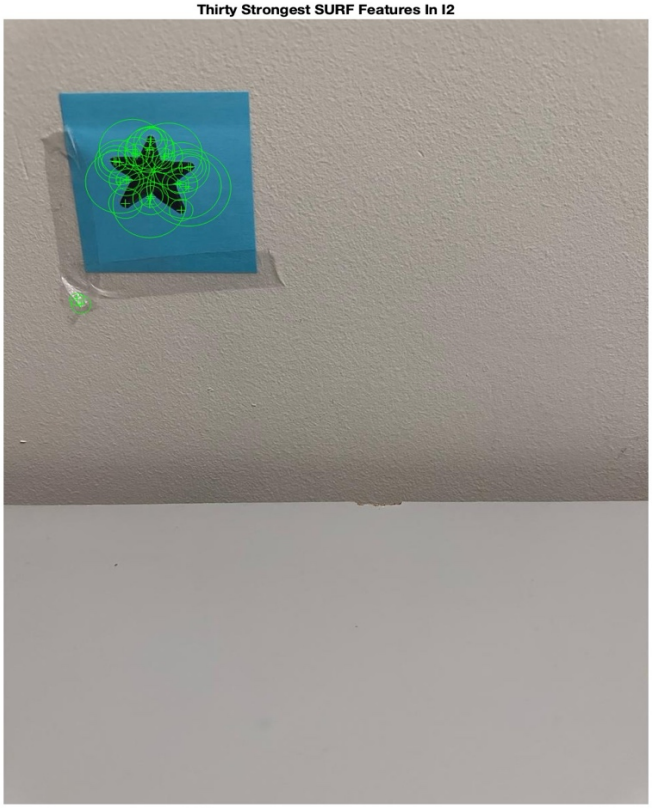
I1 (left); I2 (right)

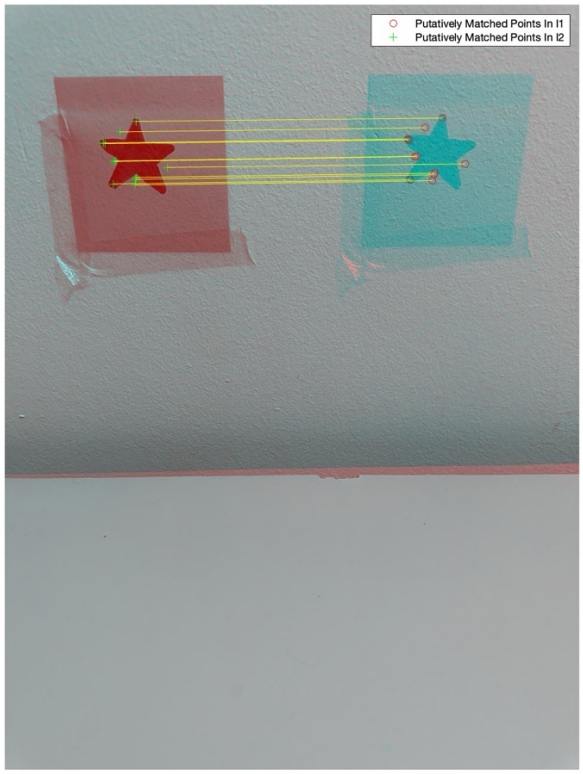


Composite Image (Red - Left Image, Cyan - Right Image)

Thirty Strongest SURF Features In I1

Thirty Strongest SURF Features In I2






**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**

I begin by configuring essential parameters such as the window name, camera ID, and frame display delay. Additionally, I set up the necessary variables and initialize the object tracker class.

Next, I continuously capture images from the camera, utilizing the appropriate detector to locate QR codes within the frames.

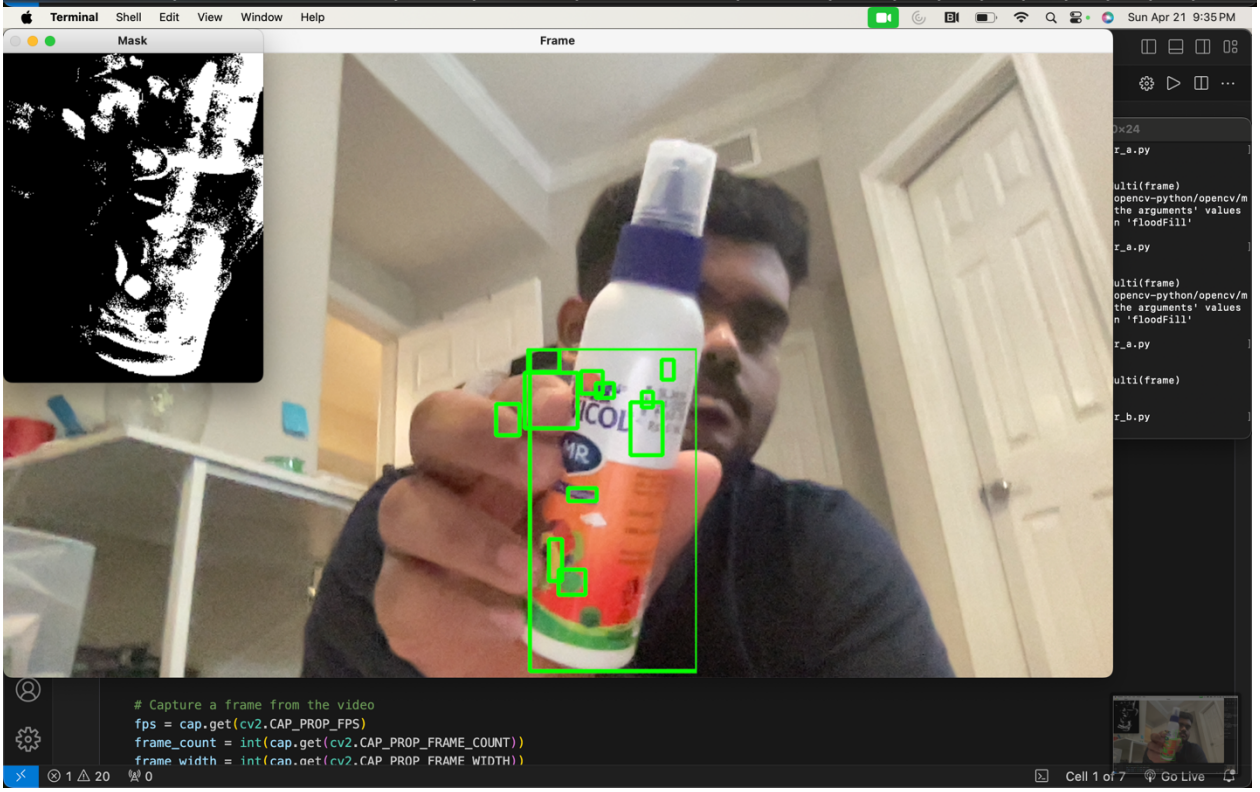
Upon detecting a QR code, I extract its details such as corner points and decoded text. I then draw bounding boxes around the identified markers to highlight them on the frame. This process is repeated as frames are continuously recorded from the camera.

To detect other objects within the frames, I employ techniques like background subtraction or object detection methods, which help in determining the bounding boxes of these objects.

To enable object tracking across frames, I update the object tracker using the detected bounding boxes. The tracked objects are then displayed on the frame, accompanied by bounding boxes around them for visual clarity.

For real-time visualization, I utilize the imshow function in OpenCV to display the tracked objects or markers directly on the frames. Additional data, such as item IDs or tracking trajectories, can also be incorporated for enhanced visualization if desired.

To control the tracking process, I implement a mechanism where the loop ends and camera resources are released upon user input, typically by pressing a designated key. This approach allows for two distinct implementations of real-time object tracking: one based on markers and the other relying solely on objects present in the scene. Each version employs different strategies for tracking and detection, providing flexibility based on specific requirements and available resources.




[https://github.com/krishnadv97/CV-Assignments](https://github.com/krishnadv97/CV-Assignments)