Project 4 : Calibration and Augmented Reality

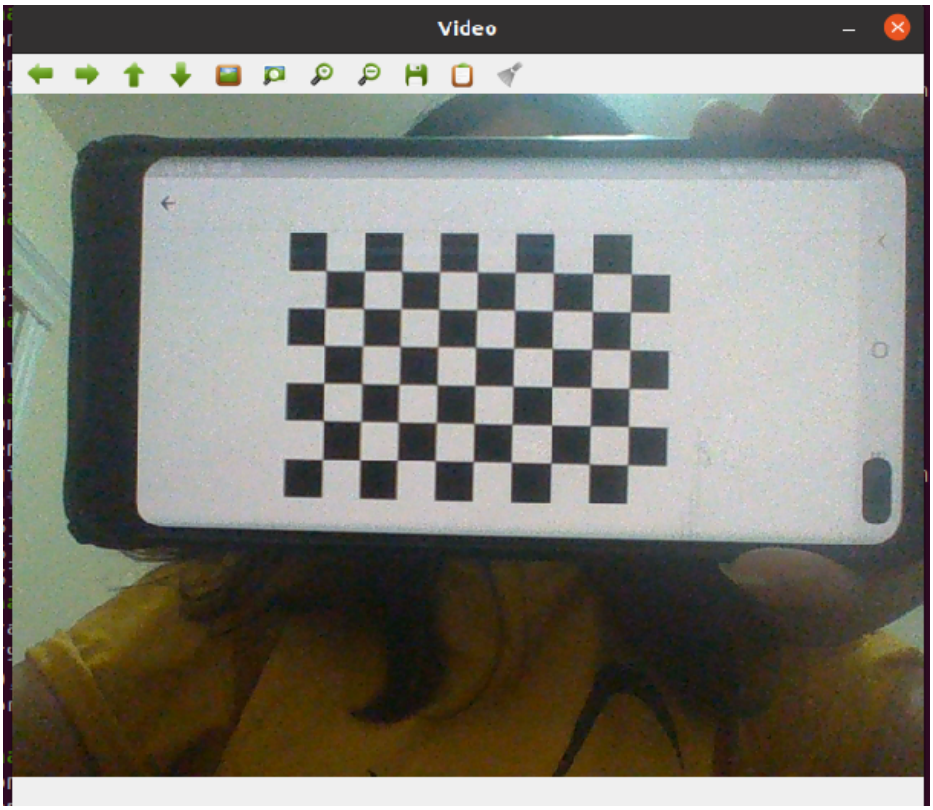
Created by Ravina Lad, last modified on Mar 31, 2022

Summary :

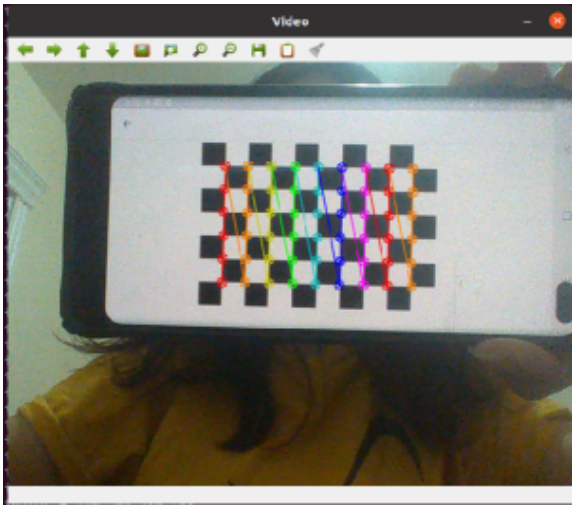
The goal of this project is to learn how to calibrate a camera and then use that calibration to create virtual objects in a scene. To begin, we must extract many frames from the video in order to obtain a set of picture coordinates and corresponding real-world coordinates with which to calibrate the camera matrix and distortion coefficients. Then, by converting the real-world coordinates to image coordinates and drawing lines between them, we might render a virtual object onto the target.

Task :

Setup :

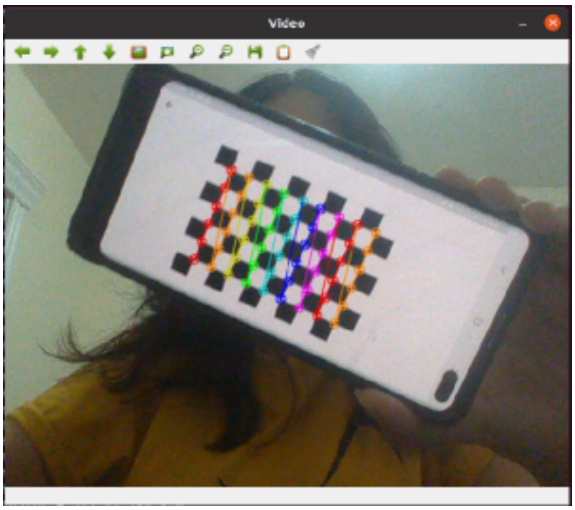


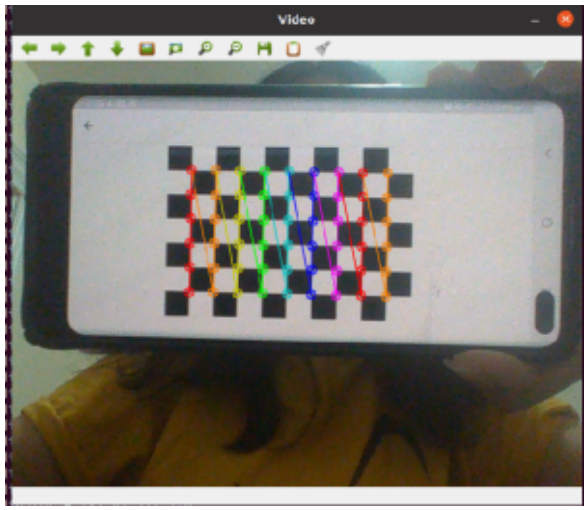
1. Detect and Extract Chessboard Corners
The corners of the checkerboard are extracted using OpenCV inbuilt functions like : *findChessboardCorners*, *cornerSubPix* and then corners are drawn on the target using *drawChessboardCorners* function.

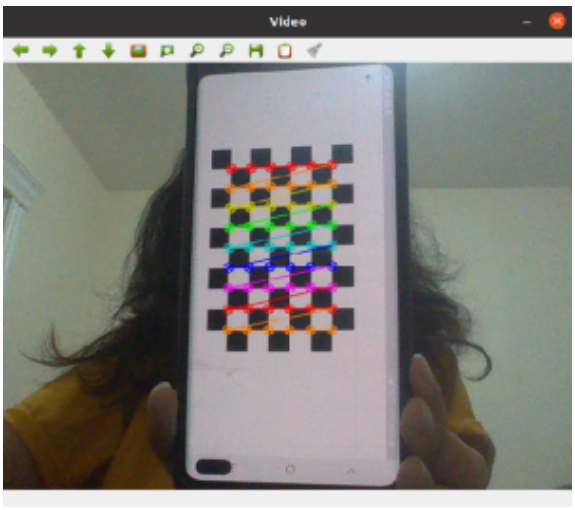


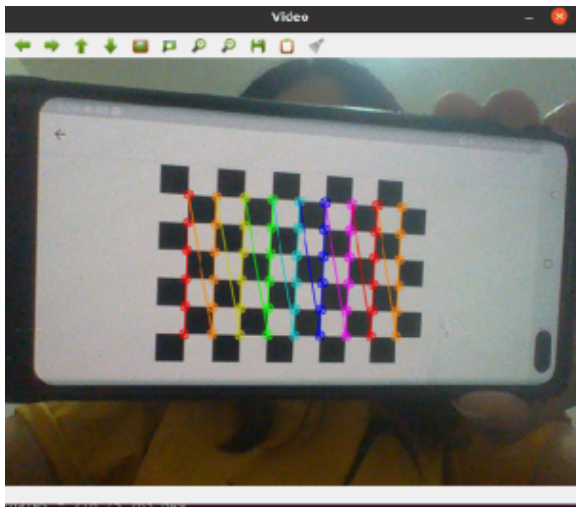
2. Select Calibration Images

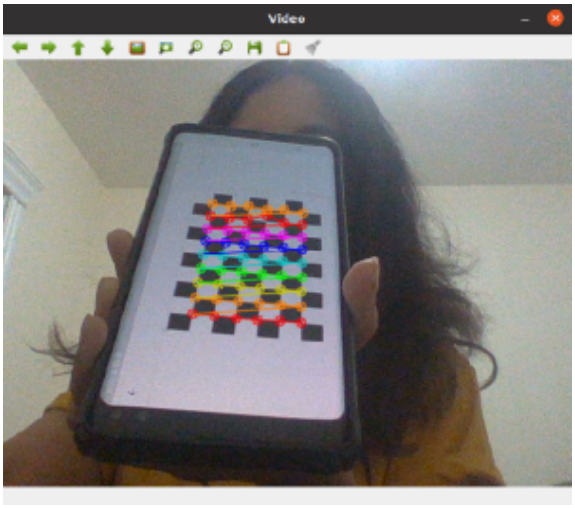
Below frames are captured for calibrating the camera. case 'c' : selecting camera calibration frames.











3. Calibrate the Camera

More than 5 frames are used for the camera calibration and re-projection error less than half pixel = 0.2962. Camera matrix and Distortion coefficients stored in CameraConfig.yml file. case "m" is used to calibrate the camera. Also, making sure that : the two focal lengths have the same value and the u0 & v0 values are close to the initial estimates**(240, 320)** of the center of the image.

```
Calibrating :
cameraMatrix : [798.5138989027281, 0, 239.5;
 0, 798.5138989027281, 319.5;
 0, 0, 1]
distCoeffs : [-0.1159301866526251, -0.5781698030028894, 0, 0, 0]
Rotation vector : [-0.01588064277740325, 0.02986276887276721, -1.502099412828044;
 0.02608624614388224, -0.05582382012644859, -1.507842583634784;
 -0.08985363153997826, 0.1186641159255782, -1.484315779227407;
 -0.3751706228458909, 0.3306824760986306, -1.414004356902562;
 -0.05805843238124055, -0.02392292077043504, -1.474826792106853]
Translation vector : [12.78773080035689, -10.7258563797844, 302.8382087882565;
 16.59120562569325, -10.53213187910248, 293.8658816866541;
 15.5535438504585, -9.038933477920992, 303.6874128314973;
 18.03316668026441, -15.17333447265766, 342.5140168767929;
 27.97184608432557, -13.64763417437742, 298.2120585120188]
Reprojection Calibration error : 0.296203
```

4. Calculate Current Position of the Camera

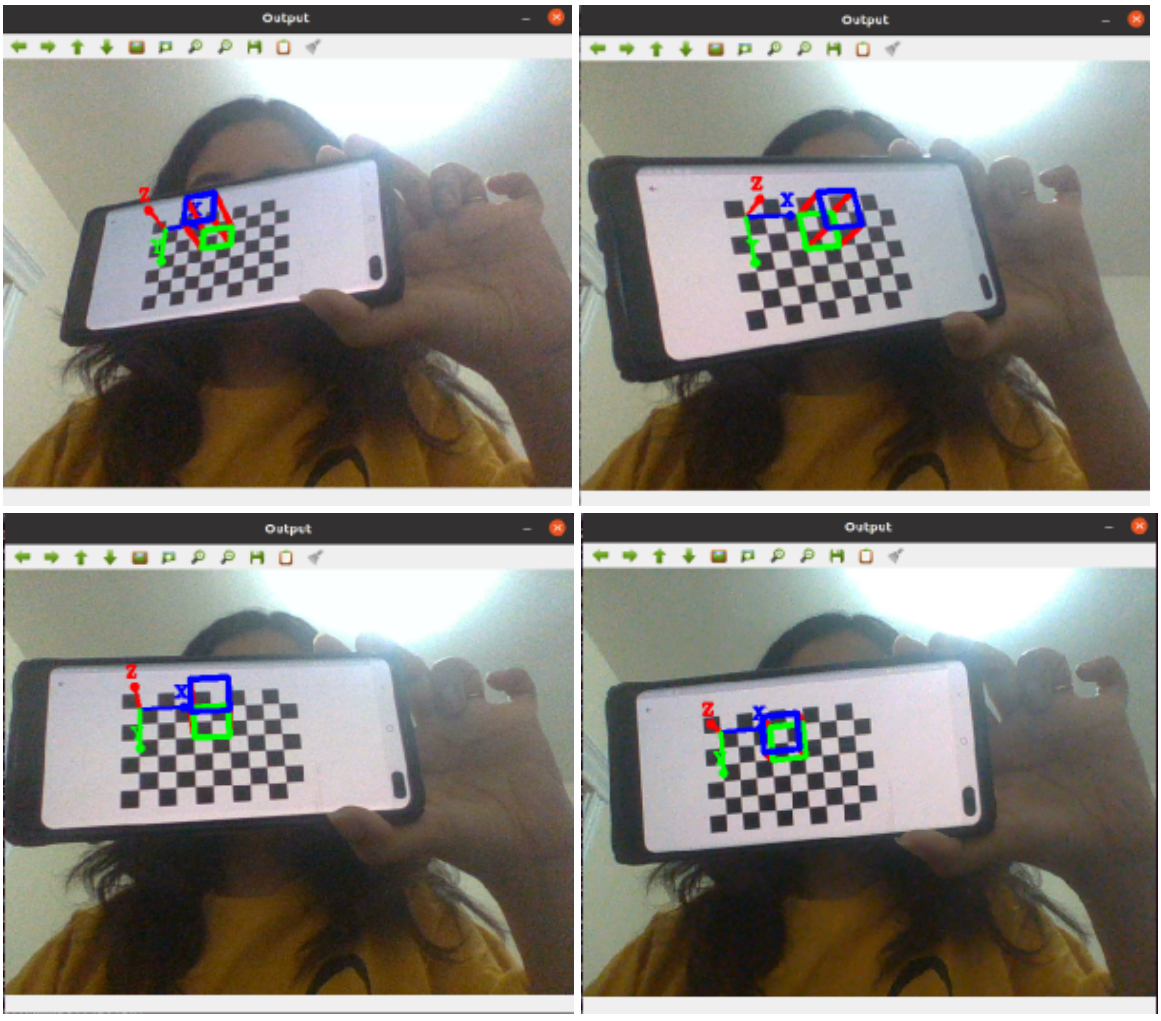
It will use the solvePnP() function to calculate the translation and rotation vectors that change a 3D point represented in the object coordinate frame to the camera coordinate frame once the corners have been discovered. It prints out these two vectors in real-time, as required:

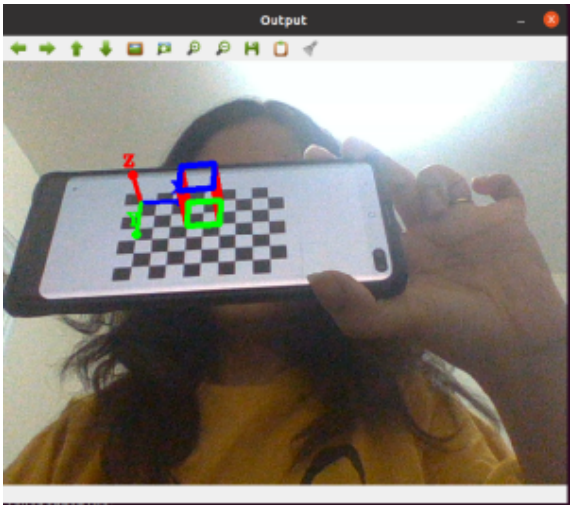
```
Rotation vector : [-0.468657751521592;
 0.02820857136142855;
 -0.9635192901984538]
Translation vector : [-14.70274476443606;
 -19.62906367893511;
 218.9641204002452]
Rotation vector : [-0.4751340027826815;
 0.05844915073949199;
 -0.9550479594999398]
Translation vector : [-14.9171591288609;
 -19.76967350078377;
 221.49994354753]
Rotation vector : [-0.4796481054692471;
 0.04483972999039573;
 -0.9576089261209378]
Translation vector : [-15.21188272079182;
 -19.68394248353187;
 220.9297080081103]
```

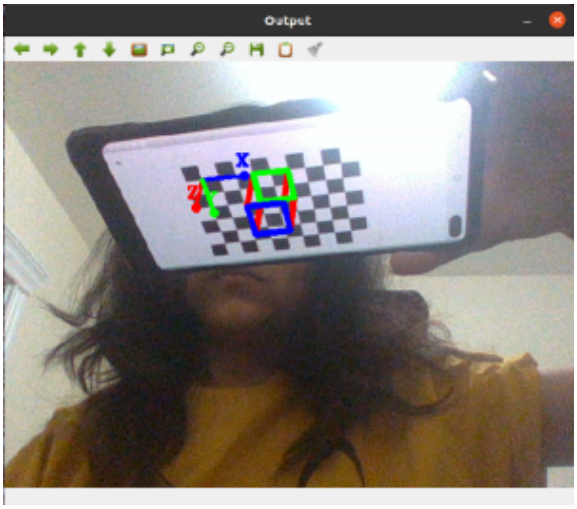
5. Project Outside Corners or 3D Axes
This task draws the 3D axes on the board attached to the origin.

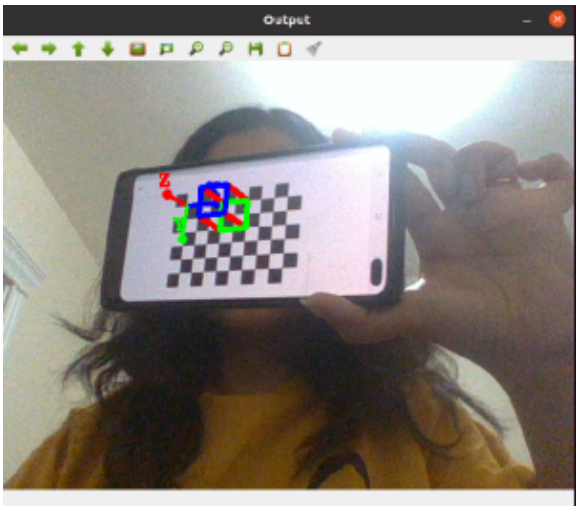
6. Create a Virtual Object
In this task, real world coordinates are successfully mapped, I can build a cube by real world dimensions.

Task 5 and 6 combined Output :



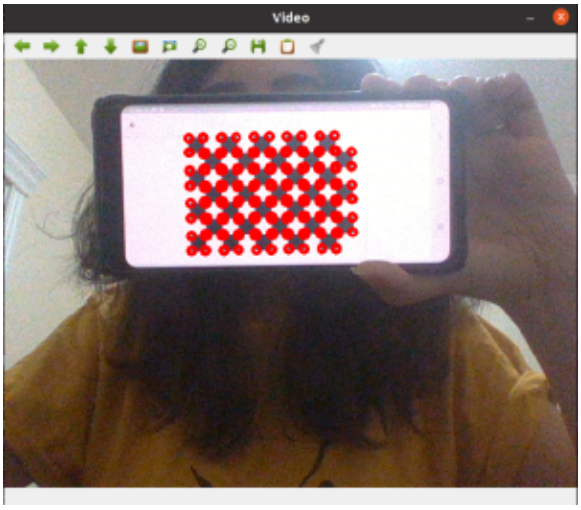


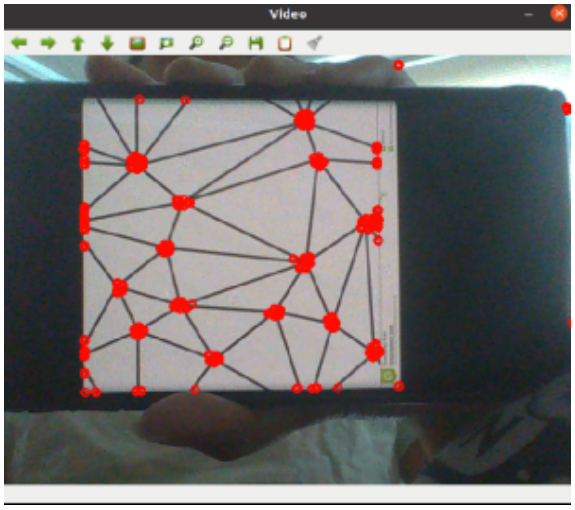


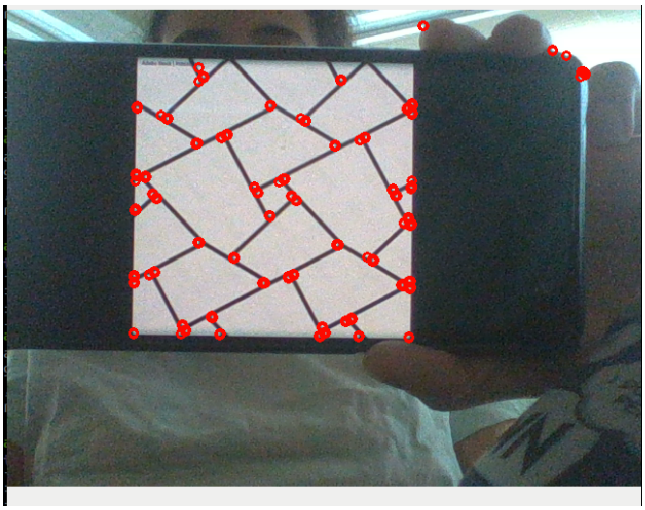


7. Detect Robust Features

Harris corners is feature detector which provides x and y coordinates of the feature in the image. It also perform a decent job with non-right angles, as you can see below images. Harris corner detector performs like the findChessBoardCorners() function, gives corners location. But, it does not give the particular order of the corner. For example, in task 1, we specified the board size as 9 X 6 and used findChessBoardCorners() to get points and place them in a vector with certain order. Unfortunately, Harris corner does not guarantee the exact number of corners, but instead only gives the location without order. This makes it difficult to to map 3D world coordinates on 2D plane.



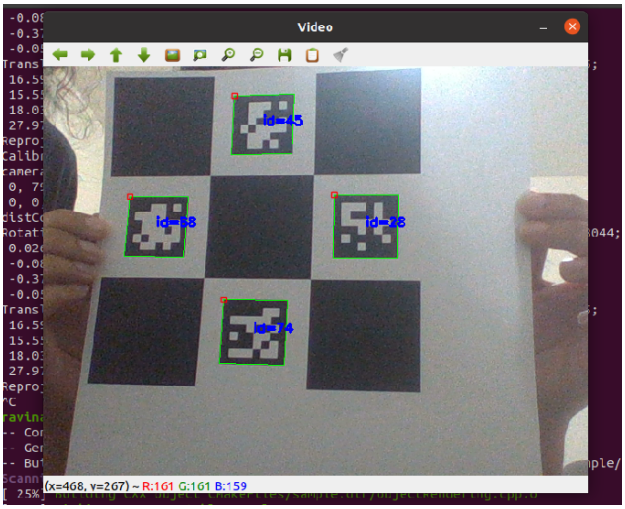


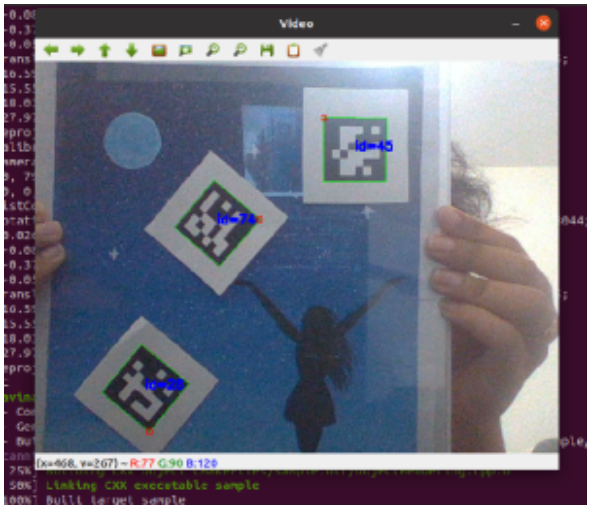


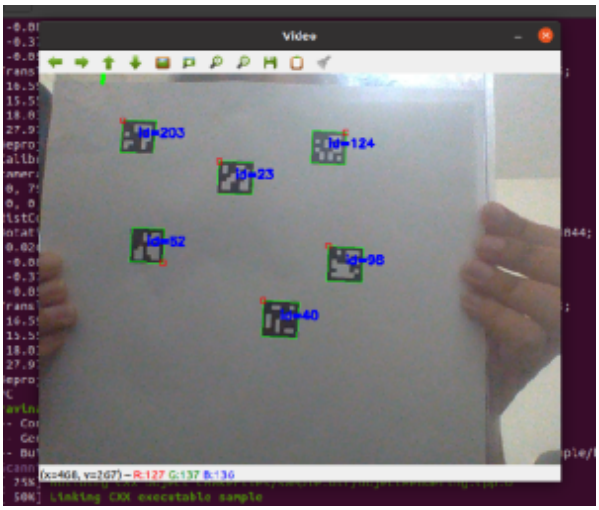
Extension :

1. Detect multiple Aruco markers:

I can identify many targets and their positions, then draw a green bounding box around them, because each marker is unique and correlates to a unique id in the arUco dictionary.







2. Augmented Reality Television:

In this task, I generated 4 markers from Id 0 to 3 and placed all these markers on the mobile screen as you can see below. I have used Aruco markers for an AR application that builds over flat surface having video frame running connecting the outer edges of aruco marker. The four markers are first detected and their pose is estimated. Another added condition is that is all four aruco markers are not found then projective video frames will not get considered for display. The final step is to overlap frames on aruco markers with projected video frames. Another 3D projection method known as "perspective transform" is used to accomplish this. This task was completed using OpenCV's getPerspectiveTransform and warpPerspective functions. (Demo video attached below)

Setup:

Video Links :

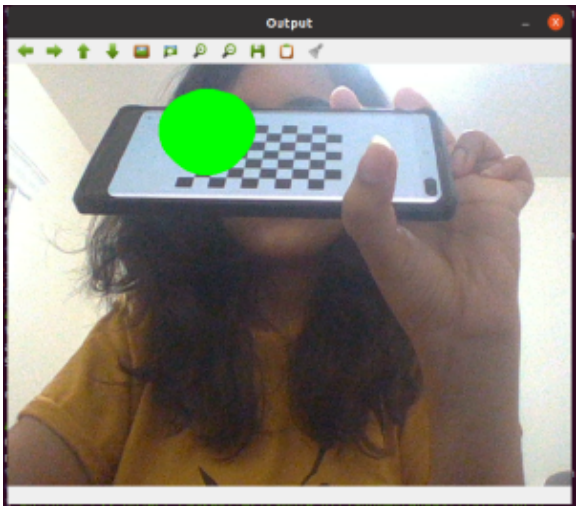
https://youtu.be/HO7w6VCAFcQ

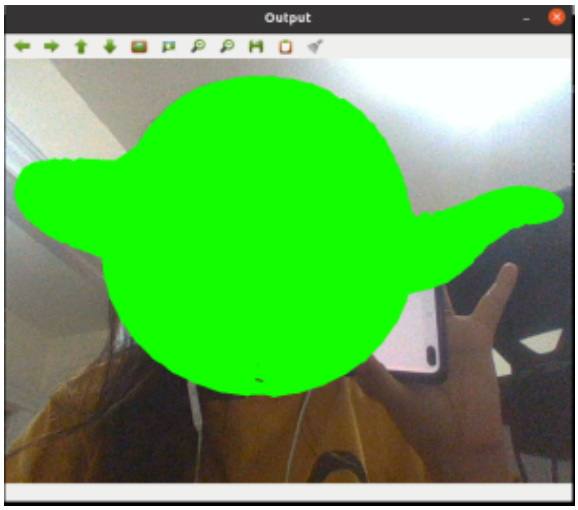
https://www.youtube.com/watch?v=1DOw8BqG23k

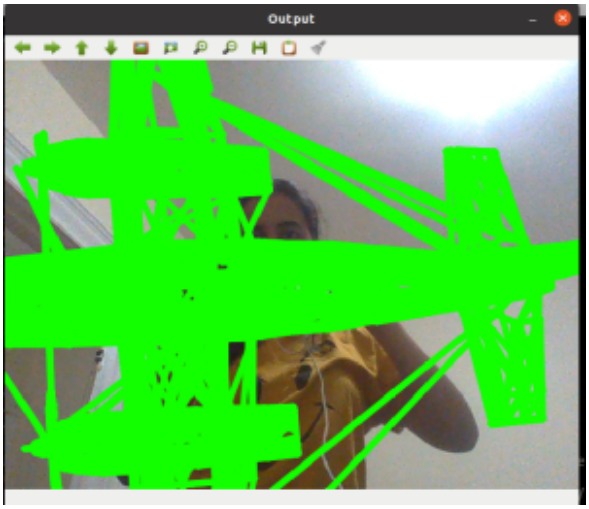
3. Render .obj File and Display on checkerboard:

For rendering, I looked for numerous open-source object files. Those object files are quite simple, comprising just of "v" and "f" fields, where "v" represents a vertex represented by a 3d coordinate and "f" represents a face consisting of the index of these vertices. As a result, the logic is to read those vertices and faces, then create multiple lines to connect the appropriate vertices for each face.

1. Sphere 2. Teapot 3. Aeroplane 4. Violin-case

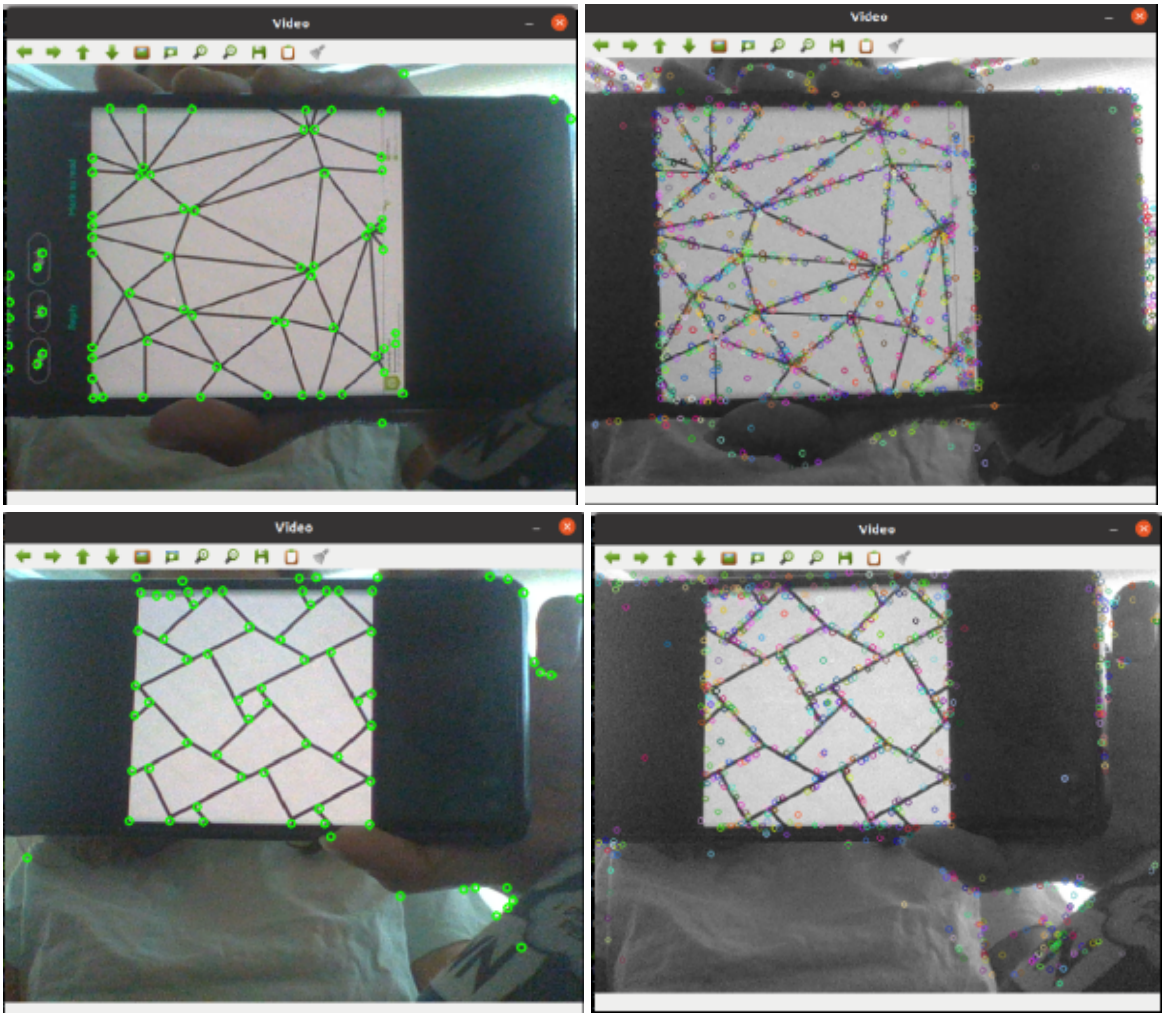






4. Implementation of shiTomasi and SIFT (feature detector and descriptor):

In addition to Harris Corners, my program also detects the corners by SIFT(Scale-Invariant Feature Transform) and shiTomasi algorithm. SIFT is scale-invariant, which Harris corners does not behave so. Moreover, these corners also show their orientations in the circle. SIFT is also feature descriptor. The input to descriptor is the location(x,y) in image, and the output is vector of numbers. Shi-tomasi feature detector provides x and y coordinates of the feature in the images.



Acknowledgments

1. https://groups.csail.mit.edu/graphics/classes/6.837/F03/models/
2. https://stackoverflow.com/questions/13214299/improving-the-result-of-harris-corner-detector
3. https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html
4. https://people.sc.fsu.edu/~jburkardt/data/obj/obj.html
5. Blog on Augmented Reality with Aruco Markers as OpenCV by Satya Mallick.
6. Bitwise operations on an Image - C++
7. Transformation of an image - C++
8. How to overlap two images - Python
9. OpenCV documents

Special thanks to TA Amit Mulay for guiding how to render .obj files.

Reflection

I learnt how to calibrate the camera's system. This project also taught me how to convert 3D points to 2D coordinates. I also looked into the arUco library, which makes augmented reality a lot easier. In addition, I now grasp the concept of perspective

transformation as well as the OpenCV methods that were used to convert a marker into a video frames.

No labels