Project 4: Calibration and Augmented Reality

Overview

The objective of this project was to learn and implement camera calibration and build an augmented reality system by inserting a virtual object in the scene. The target used for the stated purposes was a chessboard and the various OpenCV functions used were: findChessboardCorners(), cornerSubPix(), drawChessboardCorners() for finding chessboard corners, calibrateCamera() for camera calibration, solvePNP() to get the camera pose, projectPoints() to get the 3D projection of the image points. Additionally, we also implemented Harris Corner detection in an effort to get a better understanding of feature detection and extraction. In addition to these core tasks, I also implemented some extensions: detection of multiple targets in the scene and virtual object insertion; inserting virtual objects into static images and pre-captured videos; altering the target (chessboard) to something else; inserting a pre-recorded video into the target (chessboard) and testing out different cameras and comparing their calibration and quality.

Tasks

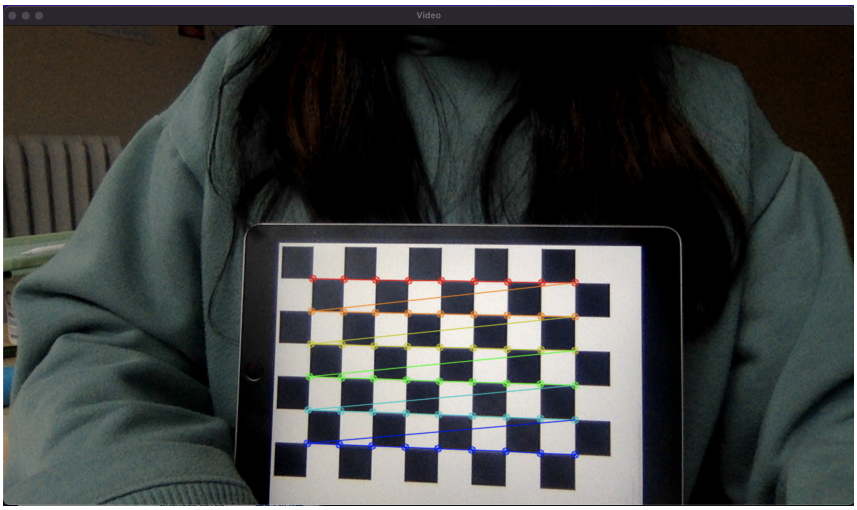
1. Detect and Extract Chessboard Corners

This task was implemented using the findChessboardCorners(), cornerSubPix() functions to find the chessboard corners in live-video. The following image shows the number of corners detected and the coordinates of the first corner i.e the top left corner.

```
Number of corners: 54
First corner: 429.06, 210.022
Number of corners: 54
First corner: 123.586, 134.72
Number of corners: 54
First corner: 759.202, 250.811
Number of corners: 54
First corner: 492.304, 112.553
Number of corners: 54
First corner: 171.415, 134.96
Number of corners: 54
First corner: 754.151, 145.82
Number of corners: 54
First corner: 74.0769, 141.68
Number of corners: 54
First corner: 646.106, 229.73
Number of corners: 54
First corner: 529.835, 30.4471
Number of corners: 54
First corner: 231.316, 48.6753
Number of corners: 54
First corner: 795.084, 65.6365
Number of corners: 54
First corner: 772.051, 52.5622
```

2. Select Calibration Images

The program saves calibration images and saves the image coordinates associated with the same on keypress 's'. The following is one such saved calibration image with chessboard corners highlighted. This was accomplished using the drawChessboardCorners() function.



3. Calibrate the Camera

The program uses the information stored via the calibration images in the previous task to calibrate the camera on keypress 'c'. The **re-projection error** obtained was **0.363863** over 6 calibration images. The following images shows the camera matrix, distortion coefficients and the Rotation and Translation vectors obtained. This was done using the calibrateCamera() function.

```
camera matrix: [968.8249409891069, 0, 621.1832815245218;
 0, 968.8249409891069, 357.1370703773134;
 0, 0, 1]
```

```
distance coeff: [-0.0698240105615798, 0.582151019195453, 0.001087238790181059,
    -0.001386918077533906, -1.232376090257053]
```

```
Rotation vector: [3.034876150034597, -0.00121406127366739, -0.01632690803088705;
 3.023504482416814, -0.125289229822528, 0.2248610116831931;
 3.006735162320923, 0.1989999606078616, -0.3965881867495268;
 -3.042587360768487, -0.07170151724288287, 0.2755971728340491;
 -2.964140887115042, 0.07292197923998645, -0.2449584391178031]
```

```
Translation vector: [-3.322059235757393, -3.976504967769404, 16.6150667125158;
 -6.337608507653039, -3.519522985002995, 15.65996099377435;
 0.4372969700583742, -4.37367395145628, 17.42445556953213;
 -1.790884255181881, -2.620788882052143, 16.33720754390722;
 -7.927018035546879, -2.469950577006393, 15.6044060829944]
```

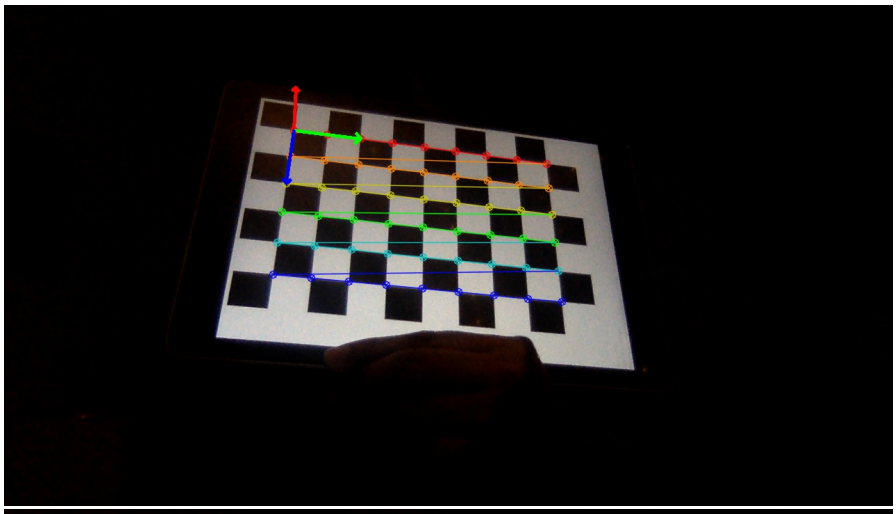
4. Calculate Current Position of the Camera

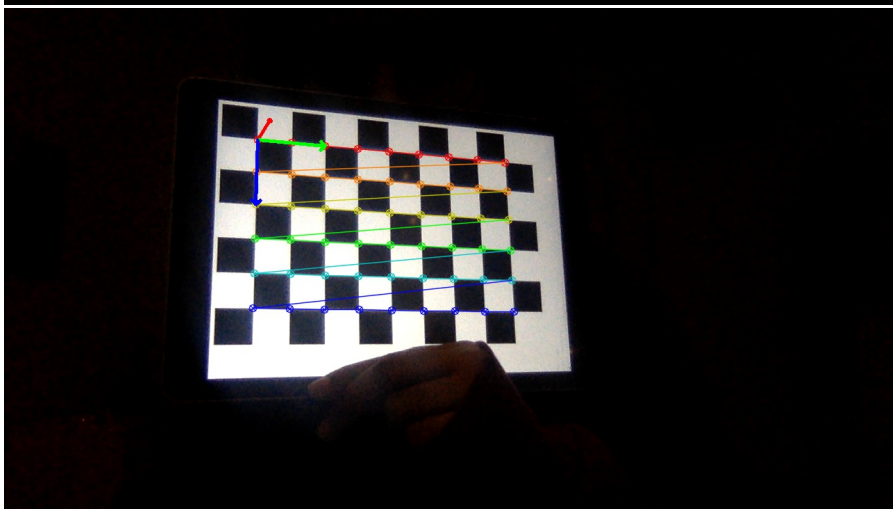
This task involved calculating the pose of the camera in real-time based on the camera calibration parameters obtained above. This was done using the sol vePNP() function. The following image shows the program printing out the rotation and translation matrix associated with each frame in real-time.

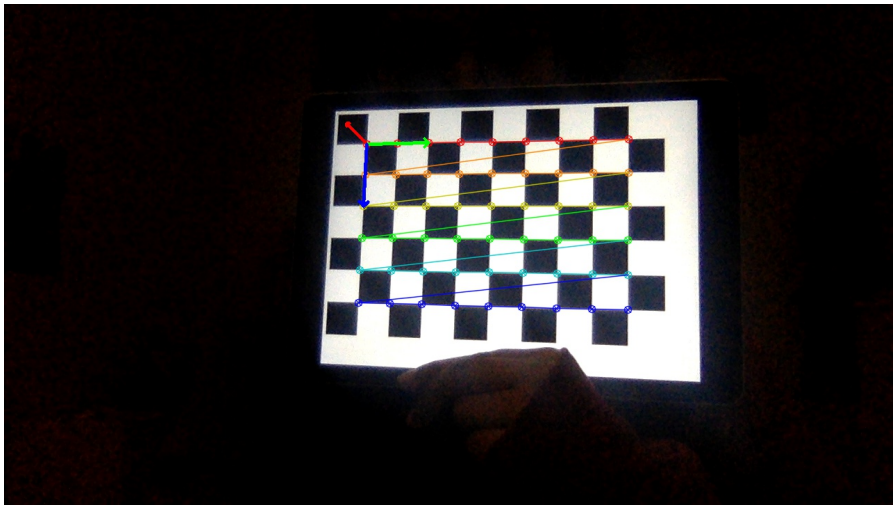
```
Translation matrix: [-3.028340394862618;
 -0.3920832008999178;
 16.89687748381007]
Rotation matrix: [2.87030751864203;
 -0.06979286557558884;
 0.1601974103319742]
Translation matrix: [-3.027955075310377;
 -0.3873876546141042;
 16.9042502495665]
Rotation matrix: [2.870836574317483;
 -0.0697610442041524;
 0.1599300038505471]
Translation matrix: [-3.027898841899968;
 -0.3917009508293831;
 16.90148949006296]
Rotation matrix: [2.870846510541177;
 -0.06982838707679835;
 0.1601623348034814]
Translation matrix: [-3.027935333246102;
 -0.3886719934320834;
 16.90243488160674]
```

5. Project Outside Corners or 3D Axes

The following images show the 3D axes projected from the corner of the chessboard. This was done using the projectPoints() function to find the 3D world points associated with the image coordinates passed as parameters.

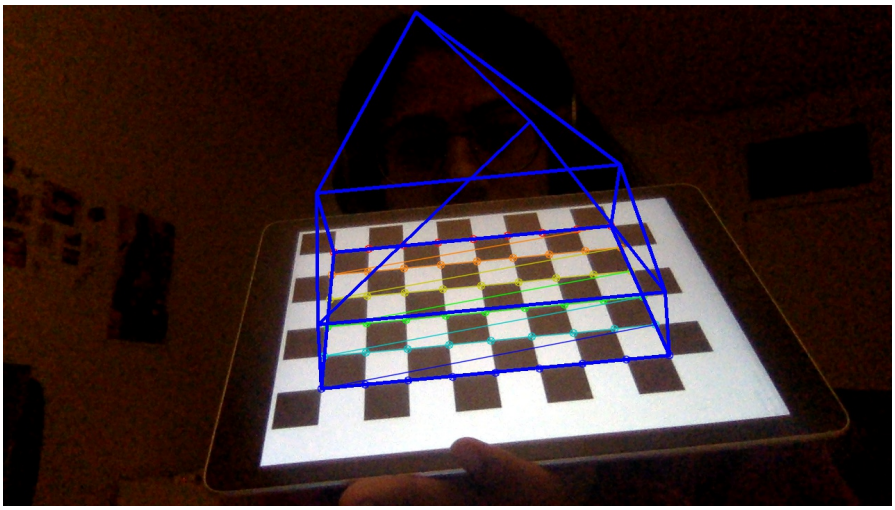






6. Create a Virtual Object

This task required us to create a virtual object made up of lines and project it on the chessboard, such that it maintains its orientation as the object moved around. This was again implemented using the projectPoints() function, and the various image points associated with the shape were passed as inputs. The following image shows a screenshot of the virtual object and the link shows the video of the system in real-time.



Task6.mov

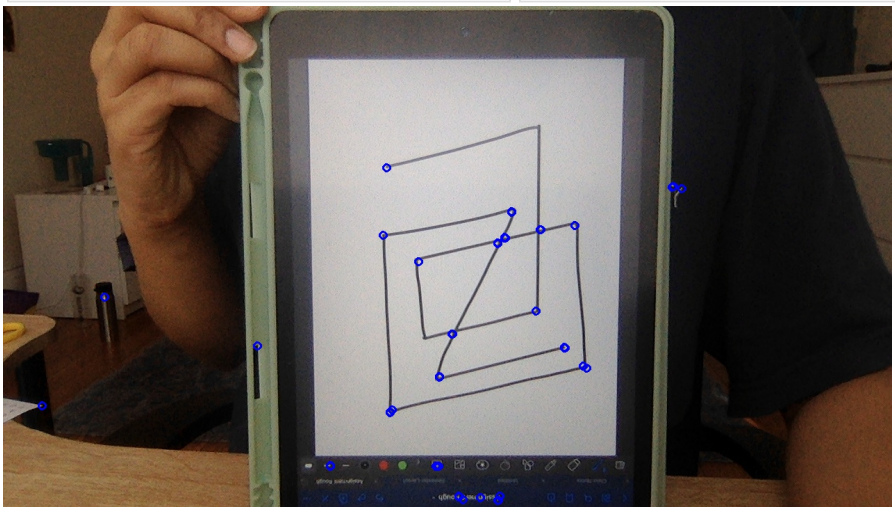
7. Detect Robust Features

The following videos show Harris corners detection using the detectHarris() function on the chessboard and AruCo markers. The image shows Harris corners detection on a random pattern of choice.

Detecting these features as opposed to chessboard corners will help us project virtual objects onto any surface of any shape as long as it has sufficient well-defined features that are able to be detected by the detection function. We could then save the image points associated with these corners, as we did for chessboard corners and pass them through projectPoints() to obtain the world coordinates associated with them. Specifically for Harris corners, the detection is not very accurate or stable hence I would use them for putting augmented reality into an image but not real time video.

Harris_1.mov

harris_2.mov



Extensions

1.Get your system working with multiple targets in the scene

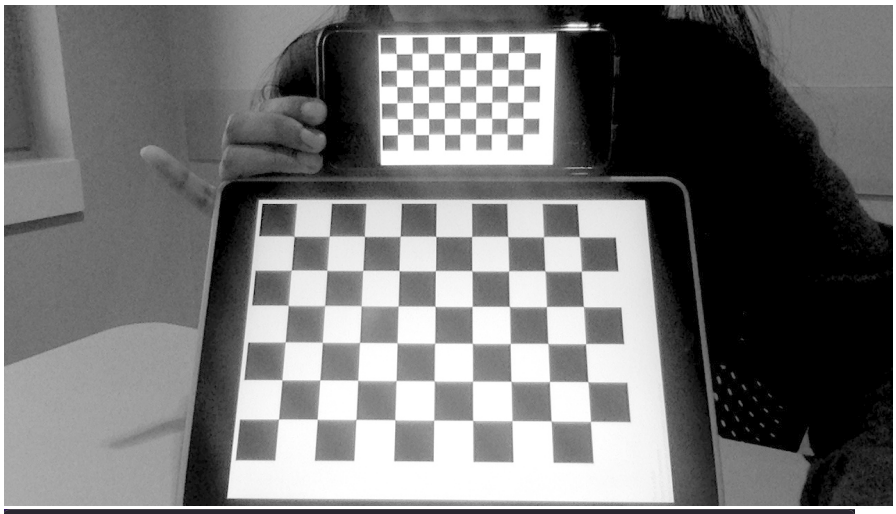
Since the findChessboardCorners() function is only able to detect one set of chessboard corners in a frame, a different workaround was required to be able to detect and then alter multiple targets in the same frame. I accomplished this by looking for chessboard corners in a loop, and every time the program found one, the pixels were all converted to white after all the processing was done. The program then proceeded to find all the chessboards in the frame and alter them. The following video shows the program detect multiple targets in the frame in real-time and project the virtual object on all of them.

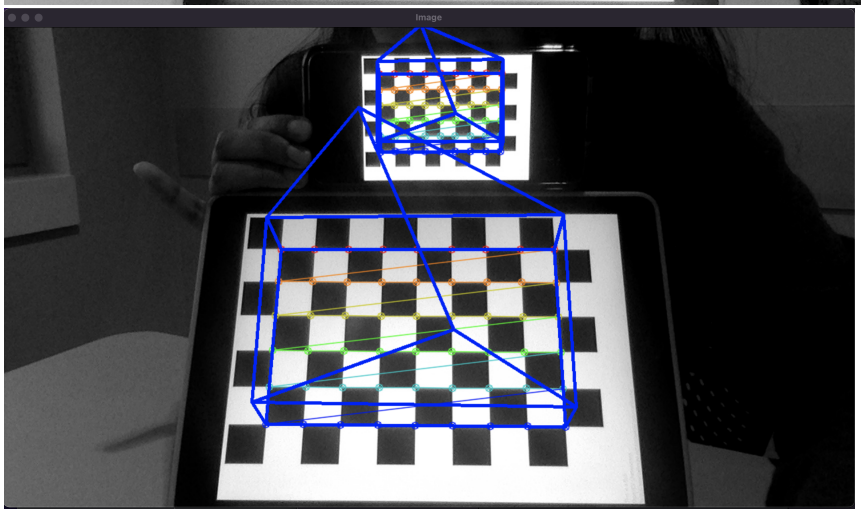
Extension_multiple.mov

2. Enable your system to use static images or pre-captured video sequences with targets and demonstrate inserting virtual objects into the scenes

The program gives the user the option to insert virtual objects in either static images, pre recorded videos and live video. The following images and video show the various features:

On Static images



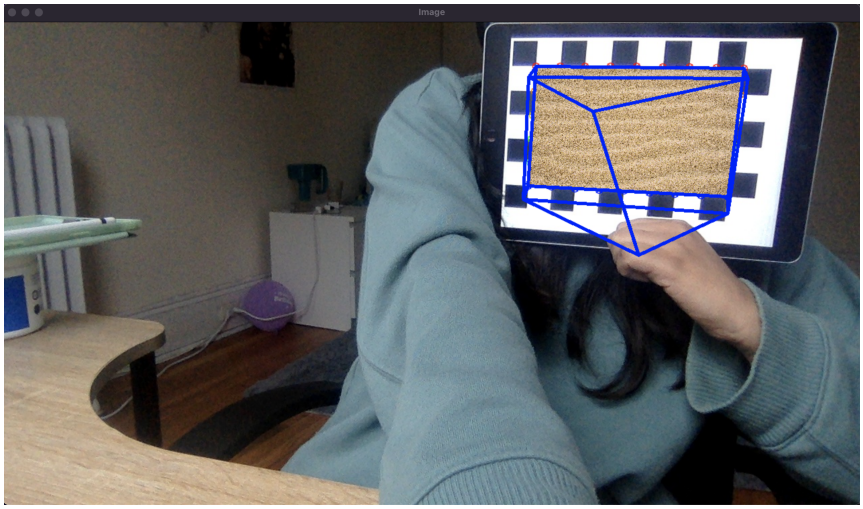


On pre-captured video:

pre_recorded.mov

3. Alter target to make it not look like a target any more

The program also has the capability to change the target (all targets in case of multiple) into any other image. The existing implementations of this functionality were all done using Aruco markers, but I was able to modify the code to be able to be implemented using the existing chessboard corners we have already found. This altered target is also able to align correctly as orientation changes. The following images and videos show the system in action.



, change_target.mov

4. Add video in target

Another application of AR that I came across while looking through various documents was the ability to insert a video - pre recorded or real-time into a target, in an effort to convert any surface into sort of a projection surface and project videos. The following video shows the implementation of the same:

change_target_video.mov

5. Test out several different cameras and compare the calibrations and quality of the results

The 2 cameras tested out were Macbook Pro 720p FaceTime HD Camera and Google Pixel 4a 5g camera. The **re-projection error** for the phone camera was **0.361845** as compared to **0.363863** for the laptop camera which is indicative of slightly better performance. But both the devices had relatively similar performance in terms of calibrations and quality of the results.

```
re-projection error: 0.361845
camera matrix: [1563.145505832088, 0, 954.9755529239125;
 0, 1563.145505832088, 544.2598420634514;
 0, 0, 1]
distance coeff: [0.1918733260794249, -1.037546746560332, 0.002647967575439087,
    0.0006605691379269086, 1.797356777153655]
Rotation vector: [-2.845408237032916, 0.03382154006136256, -0.01061057730103147;
 -2.841254323125598, -0.1071611863153995, 0.1555114987195042;
 -2.82100993928813, 0.3213929940232856, -0.4921914403217958;
 -2.699121180836287, -0.05481908373254087, 0.07854346725537487;
 -2.661902186999566, 0.199226925982303, -0.3471119353400625;
 -3.050871249644881, 0.03887881810522788, -0.03790170827823461]
Translation vector: [-3.099416398973025, -2.454255444428846, 11.66350857038822;
 -4.078785492456188, -3.364826167136139, 12.96318957249192;
 -2.158347722779029, -1.859710777174981, 12.67303096852882;
 -3.475849086341369, -1.392449459434664, 12.54666019804833;
 -1.215621729901908, -1.18013778164018, 11.4410978904999;
 -2.570301714636261, -2.590548434408333, 19.27576221146029]
```

Reflections

This project was a very fun introduction to Augmented Reality and all the various factors that go into it such as Camera calibration, calculating the pose etc. Augmented reality was one of the things that always seemed complicated to me but this project helped me understand how to go about it. It was also a very fun creative outlet and I spent many hours trying to integrate OpenGL with OpenCV. Even though I was unsuccessful and could not invest enough time due to time constraints, the potential of all the things that can be done is super exciting and definitely something I would like to dive deeper into. Additionally, this project really helped put into perspective the different coordinate systems, their corresponding conversions and how they're all utilised.

Acknowledgements

OpenCV documentation

https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d

https://learnopencv.com/camera-calibration-using-opencv/

https://learnopencv.com/reading-and-writing-videos-using-opencv/#read-video-from-file

https://pyimagesearch.com/2021/01/04/opencv-augmented-reality-ar/

https://learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/