Calibration and Augmented Reality

Malhar Mahant and Kruthika Gangaraju

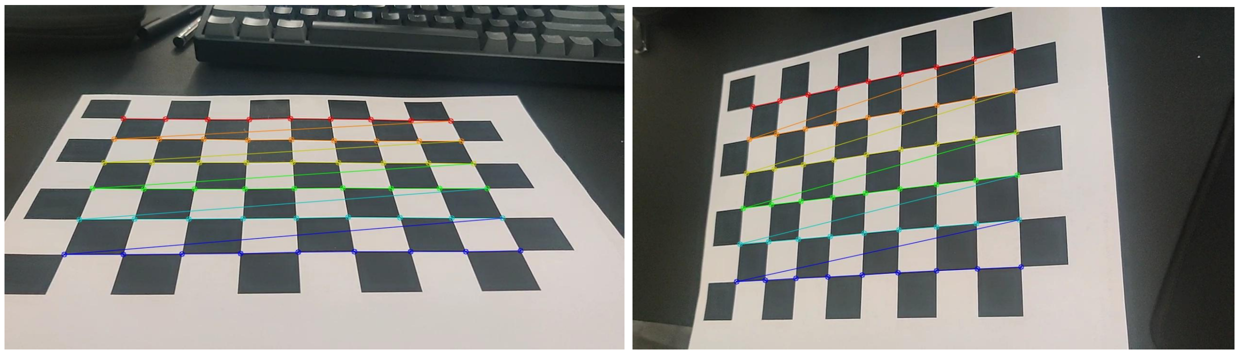
Description:

Augmented Reality deals with creating virtual objects in an existing environment which can be seen through a camera device. For this project, we detected and extracted the corners of the chessboard. After detecting these points, we proceed to calibrate our camera and store this calibration data. With the calibrated camera, we are able to determine the position and orientation of the camera relative to the target scene. This enables us to accurately place 3D axes and displayed 3D objects on the chessboard, ensuring that it adjusts and aligns properly as the camera or target moves.

Please refer to the readme for instructions on how to use the application.
Time Travel Days used for this project: 0.

Task 1:

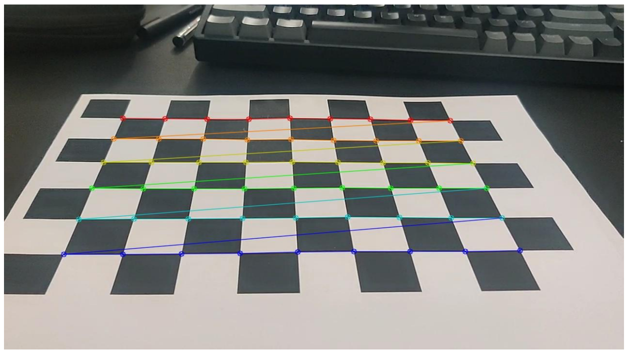
For this task, we first detected the corners of the chessboard using the findChessboardCorners() function which takes the input image and the size of the chessboard as inputs and detects whether corners are present. Once the corners are detected, the corners are refined using the cornerSubPix() function. After the corners are detected and refined, an image is displayed in which the corners of the chessboard are displayed using the drawChessboardCorners() function.
Example Images:

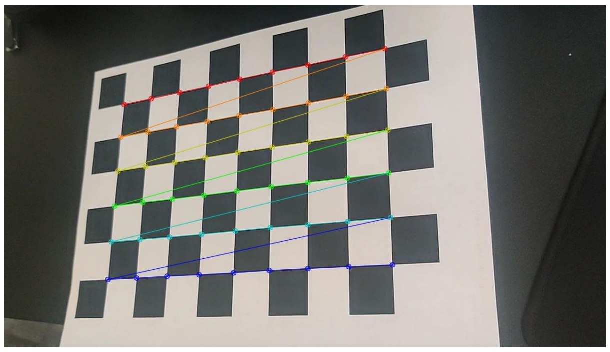


Task 2:

After the corners are detected, we let the user select if that particular image is to be used for calibration. If the image is chosen, the corner locations are stores and the corresponding coordinates in the world frame. This way we can get as many calibration images as needed.

Here are sample of calibration images in different orientations:

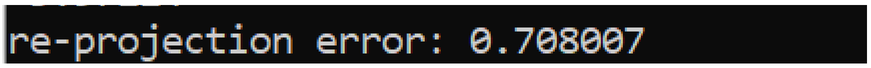




Task 3:

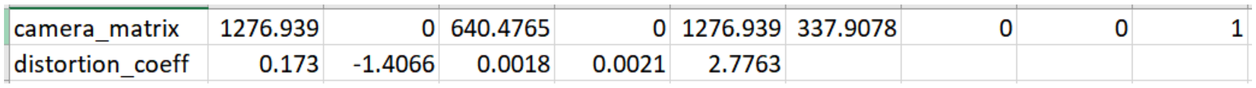
The process of estimating the parameters of a camera is called camera calibration. We first need to transform the point from the world coordinate system to the camera coordinate system using the extrinsic parameters (Rotation and Translation). using the intrinsic parameters of the camera, we project the point onto the image plane. Once we get the intrinsic and extrinsic parameters of the camera, the camera is said to be calibrated. We stored the values of the camera matrix and distortion coefficient to file at this stage and displayed the reprojection error.

Reprojection Error:



```
re-projection error: 0.708007
```

Camera Matrix and Distortion coefficient stored to file:

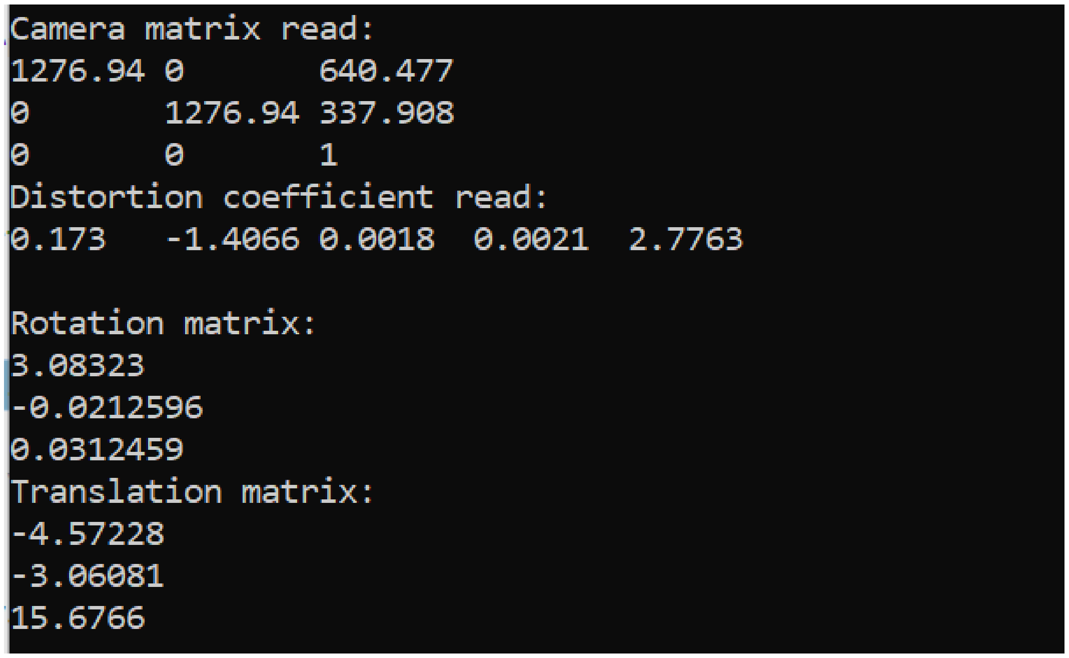


camera_matrix	1276.939	0	640.4765	0	1276.939	337.9078	0	0	1
distortion_coeff	0.173	-1.4066	0.0018	0.0021	2.7763				

Task 4:

We read the saved calibration parameters from the CSV file and extract corners of the chessboard from each video frame. The pose computation problem consists in solving for the rotation and translation that minimizes the reprojection error from 3D-2D point correspondences. We use the saved calibration parameters and the corners from the current frame in the solvePNP() function to obtain the rotation and translation vectors.

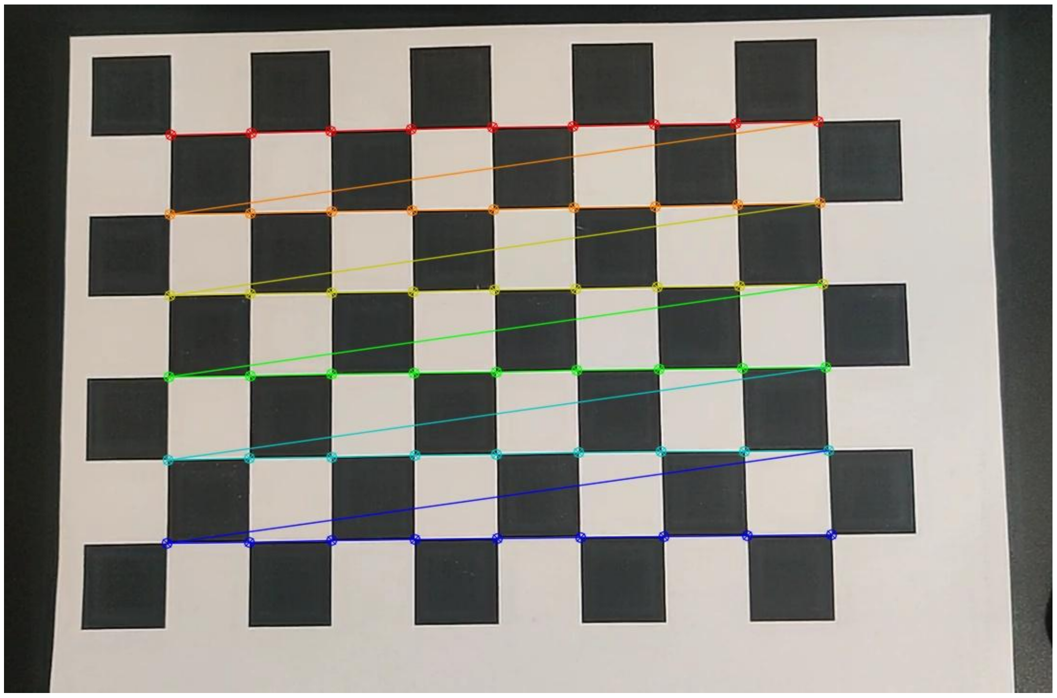
Example of printing rotational and translational data in real time:



```
Camera matrix read:
1276.94 0       640.477
0       1276.94 337.908
0       0       1
Distortion coefficient read:
0.173   -1.4066 0.0018  0.0021  2.7763

Rotation matrix:
3.08323
-0.0212596
0.0312459
Translation matrix:
-4.57228
-3.06081
15.6766
```

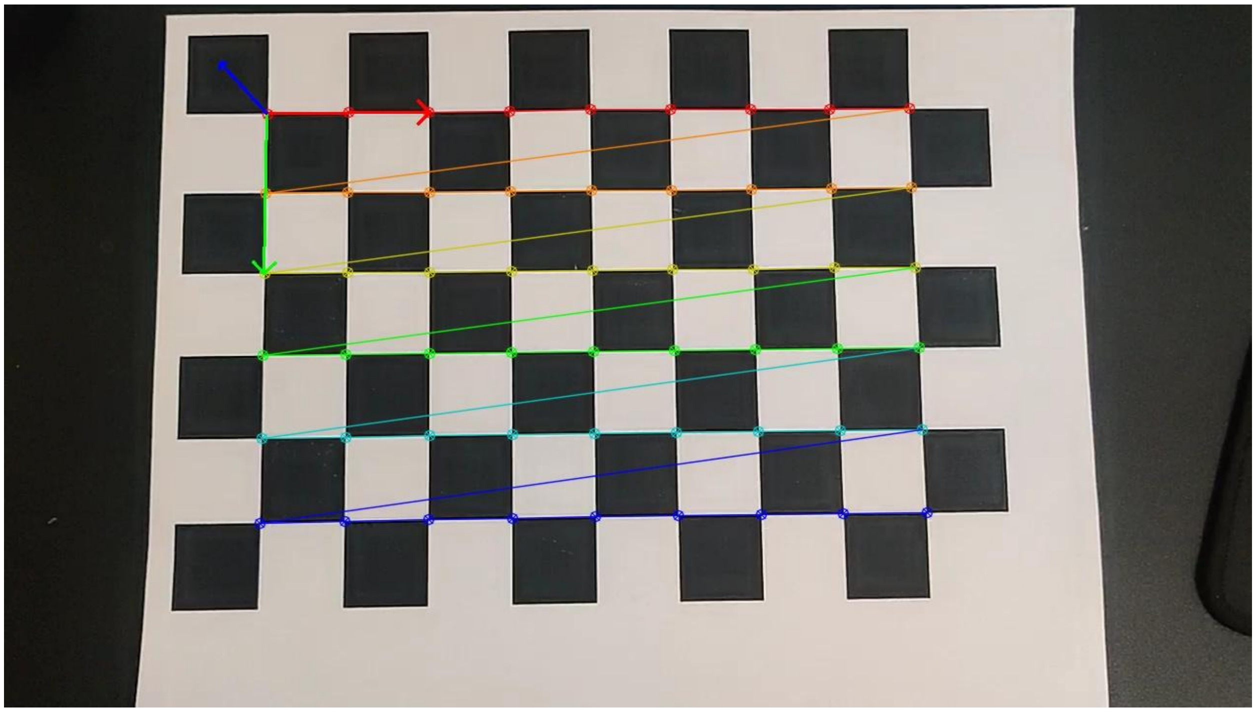
Image which gave the above result:

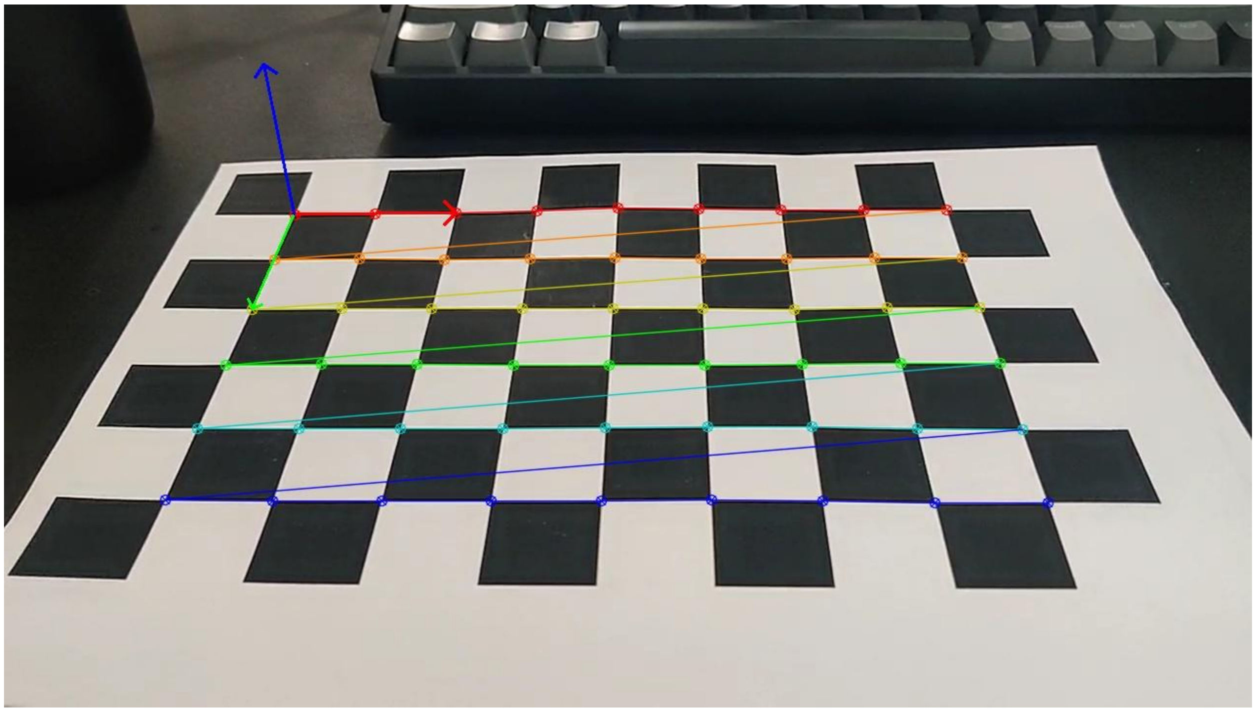


Task 5:

Once we collected all the calibration images and retrieved the intrinsic parameters of the camera, we displayed the image of the chessboard with the 3D axes. We used the OpenCV projectPoints function to map the 3D coordinates from the chessboard system to real coordinates in the image. And used OpenCV to draw the axes lines.

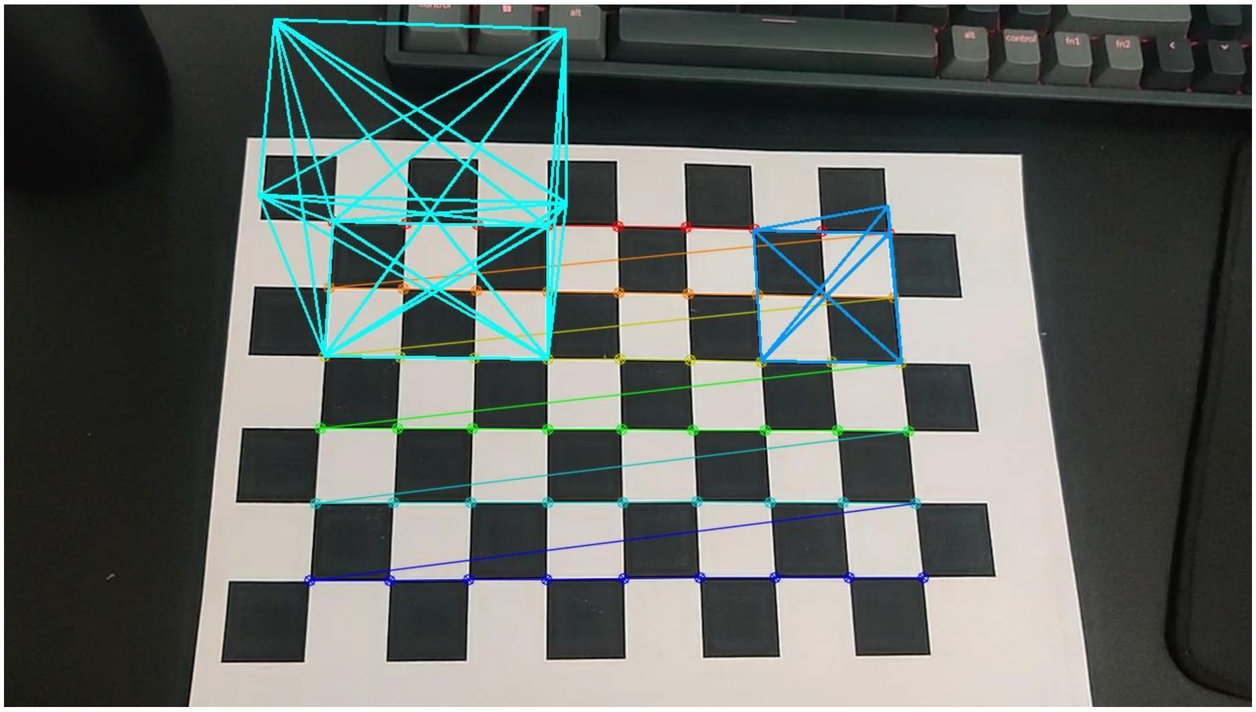
Example of drawing axes at the origin:

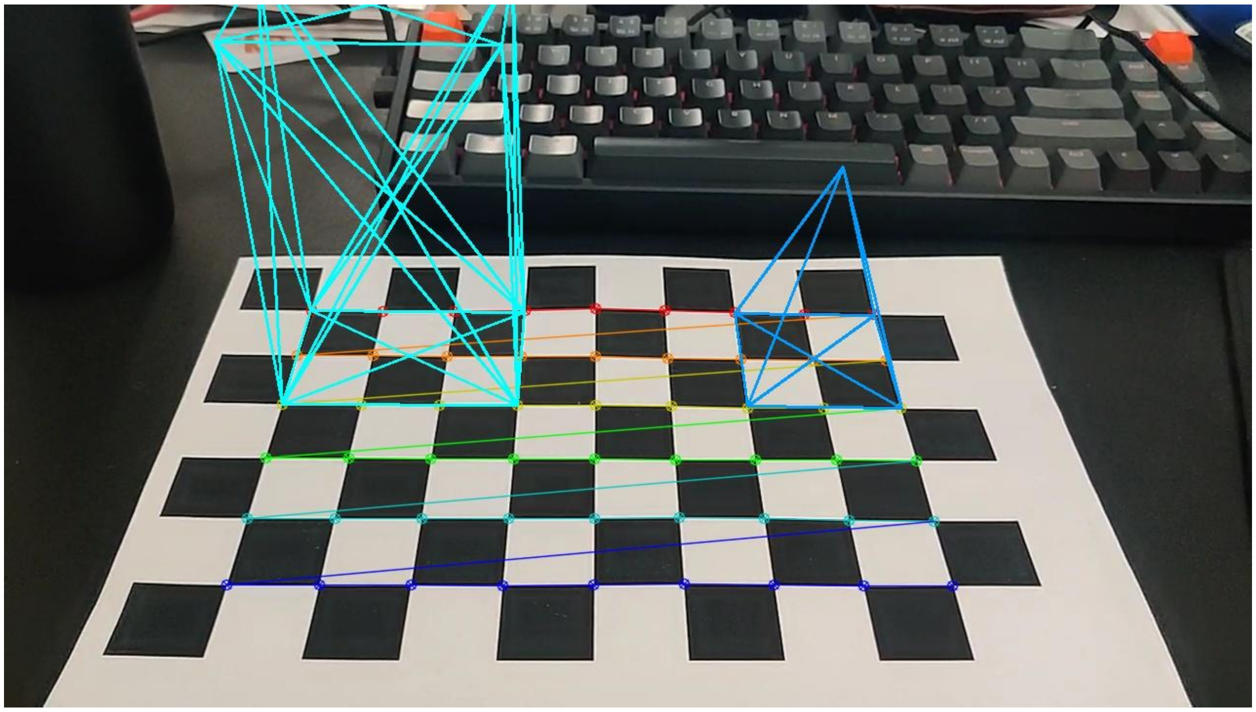




Task 6:

For this last step in creating virtual objects, we defined vertices for the 3D object and connected the lines. We defined 3D objects in the coordinate system of the chessboard corners. We used the OpenCV projectPoints function to map the 3D coordinates of the objects from the chessboard system to real coordinates in the image. And used OpenCV to draw the lines.





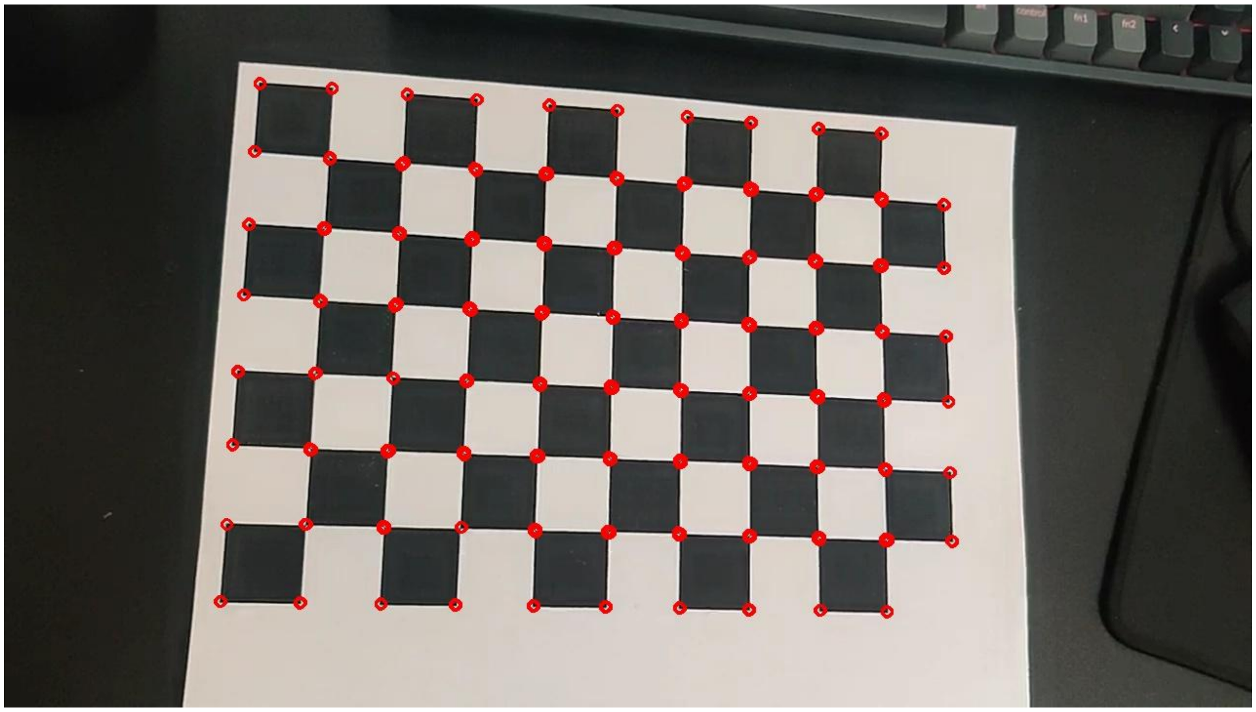
Task 7:

For this task, we decided to implement Harris corner detection. The first step for this is to convert the image to grayscale. The cornerHarris() function detects the corners using size of neighborhood considered for corner detection and aperture parameter of the Sobel derivative used. Harris corners detect local changes in intensity using the second moment matrix of image derivatives. Circles have a continuous and smooth boundary therefore, the second-order derivatives of the image intensity at any point on a circle are zero, which means that the Harris corner detector would not be able to detect these points as corners. Hence, it suffers with detection in patterns with circles.

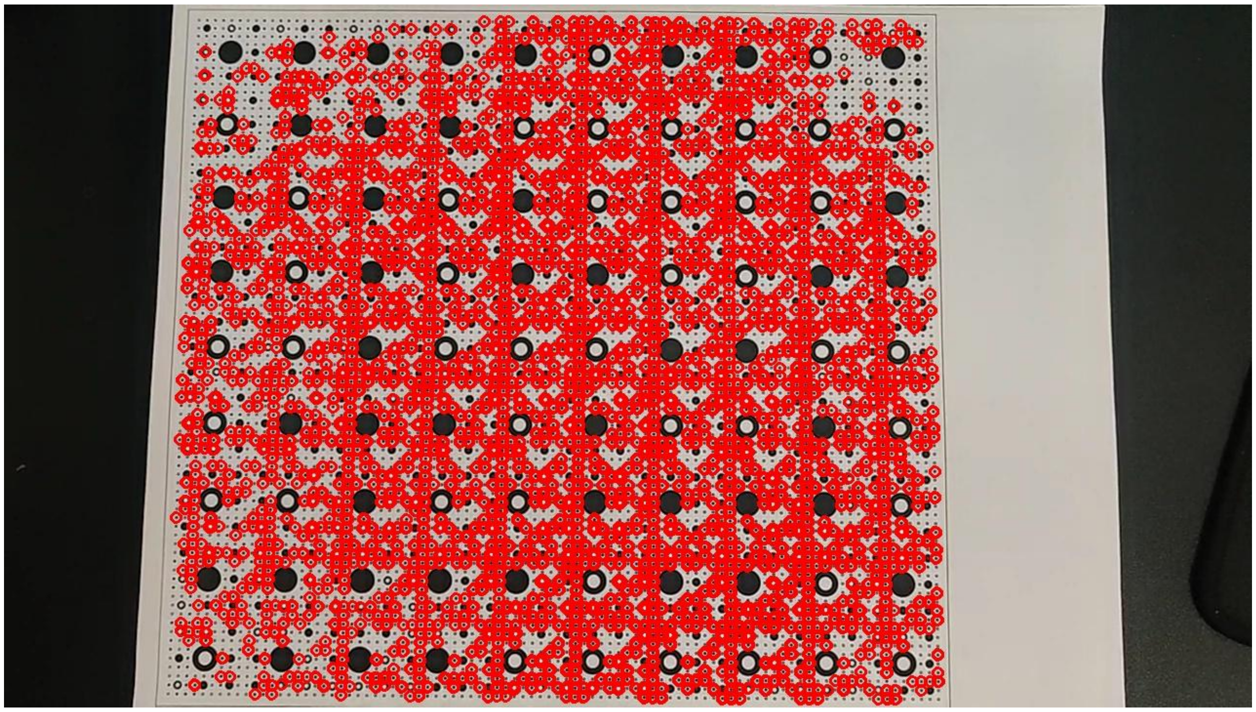
Harris corners can be used as anchor points for placing virtual objects just like corners returned from findChessboardCorners. Using a pattern that has more identifiable Harris corners could provide a more granular coordinate system to define complicated VR shapes.

Example of Harris Corner detection:

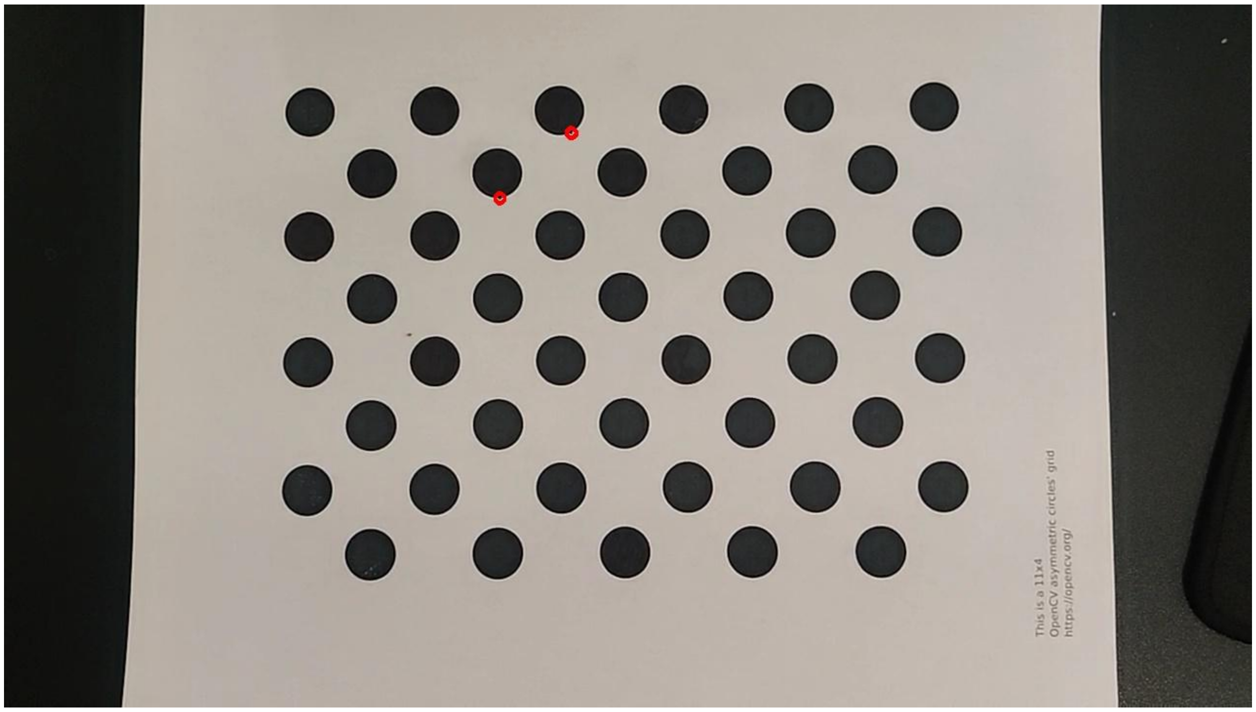
With chessboard pattern:



With a different pattern:



With a asymmetric circles pattern:

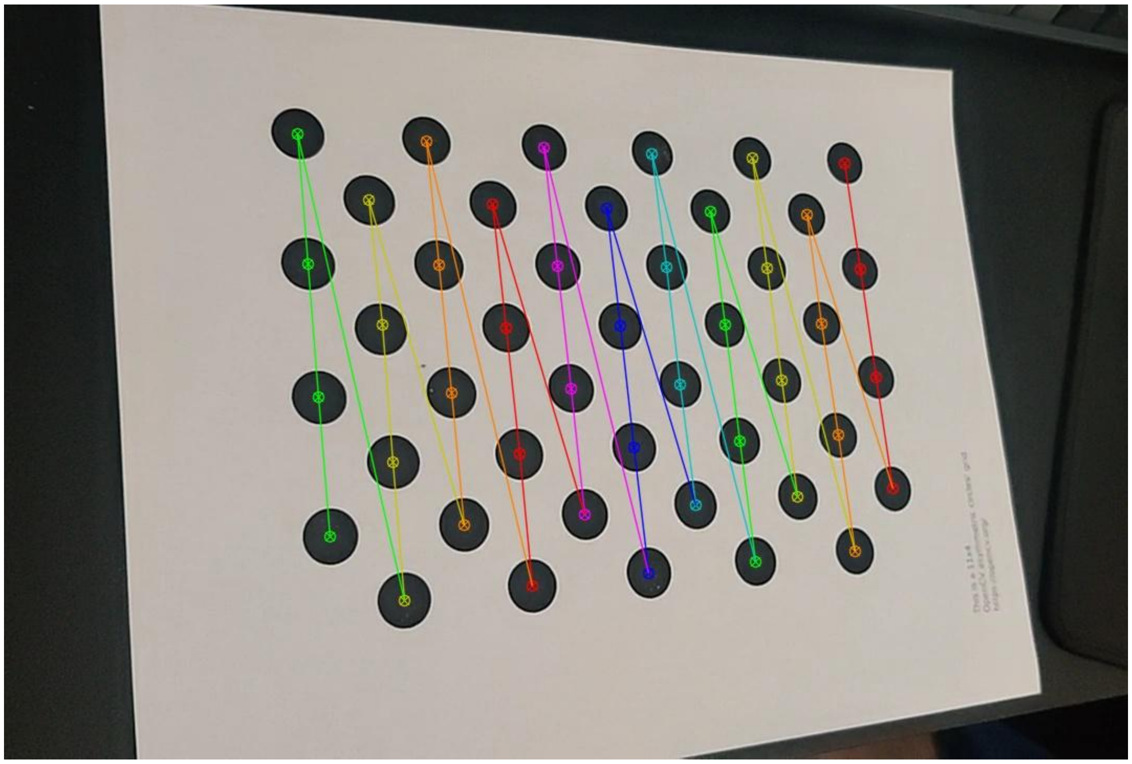


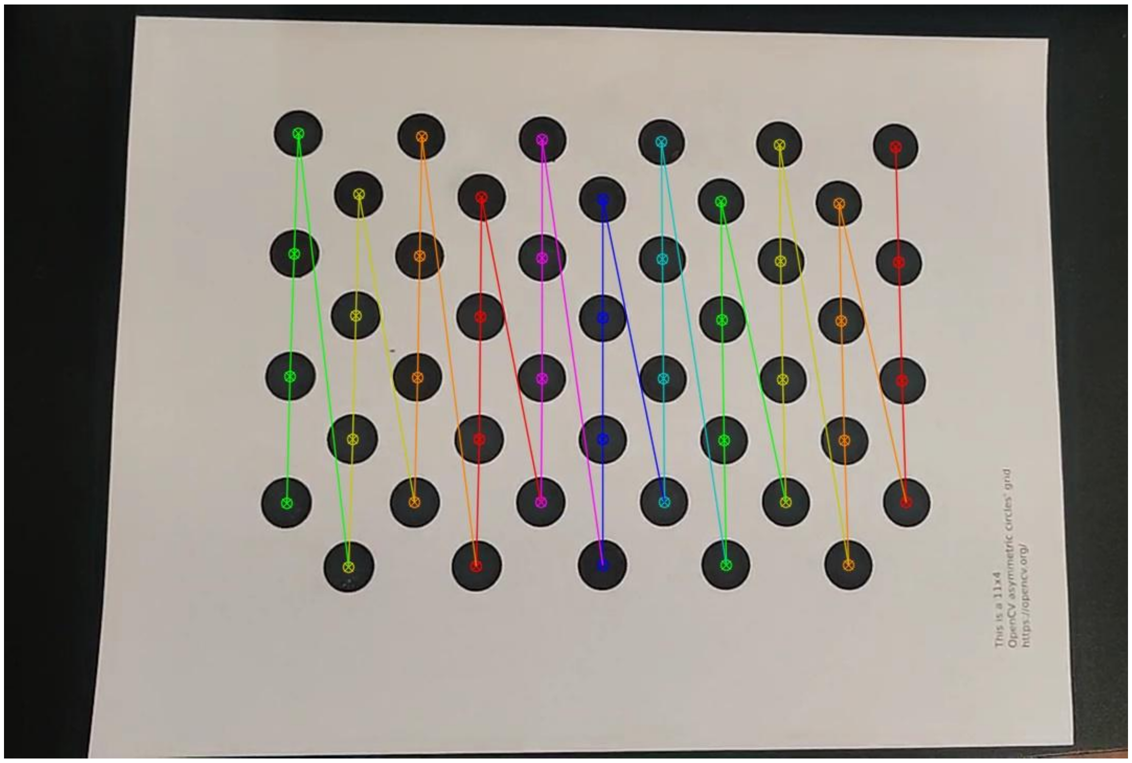
Extensions:

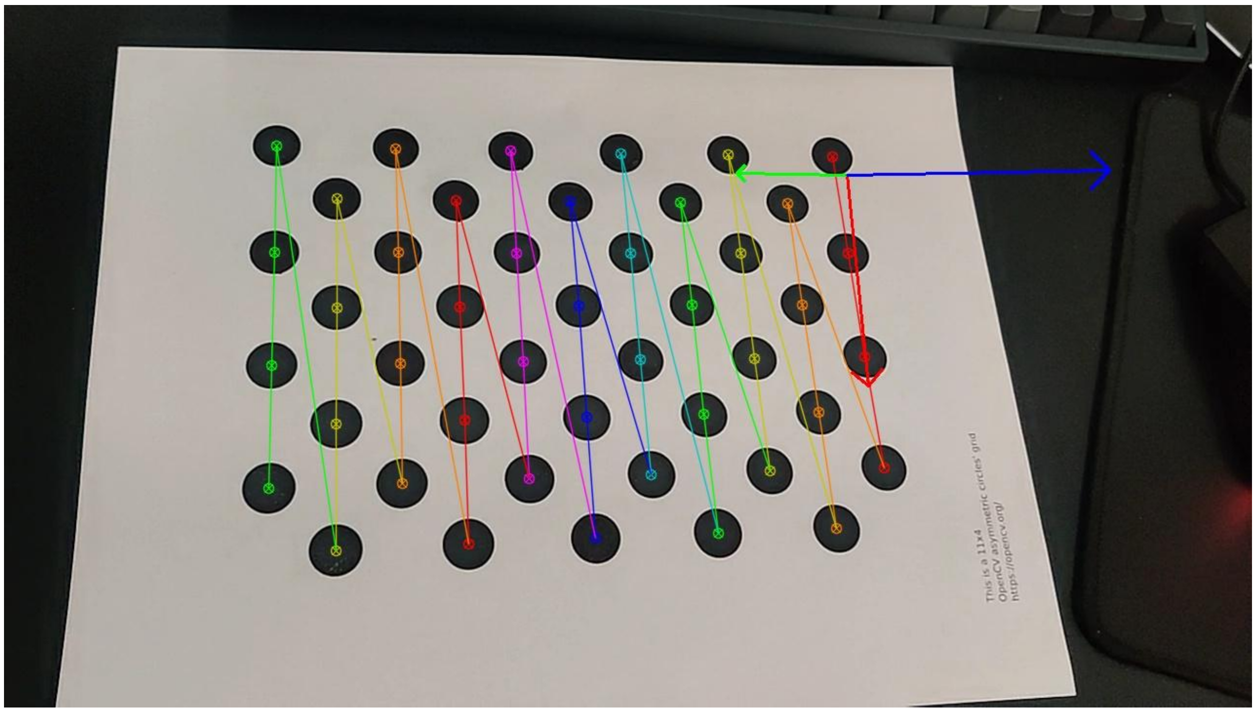
For extensions we decided to perform the following tasks:

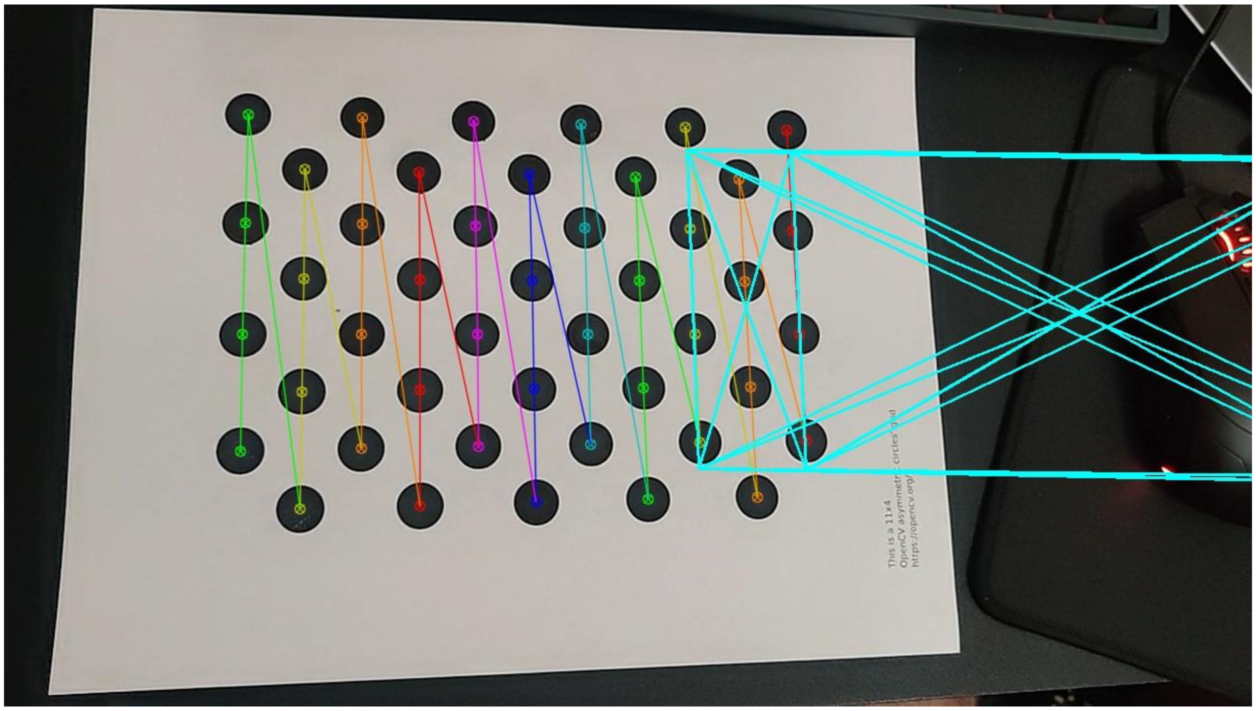
Extension 1: Get your AR system working with a target other than the checkerboard

For the first extension, we got the system working with a target of asymmetric circles grid pattern instead of a chessboard. We used a 4 x 11 grid of circles, it differs from the checkerboard pattern as the identifiers are circles and additionally the adjacent circles are staggered. We also implemented placing of 3D axes and 3D objects on this asymmetric circle's grid pattern. This was achieved by using the findCirclesGrid function from openCV to identify marker points in the circle pattern. findCirclesGrid uses a blob detector that can be fine tuned to detect circles even when camera is at an angle (which makes circles look elliptical) or if circle radius changes. Example images:




This is a 11x4
OpenCV asymmetric circles' grid
https://opencv.org/


This is a 11x4
OpenCV asymmetric circles' grid
https://opencv.org/

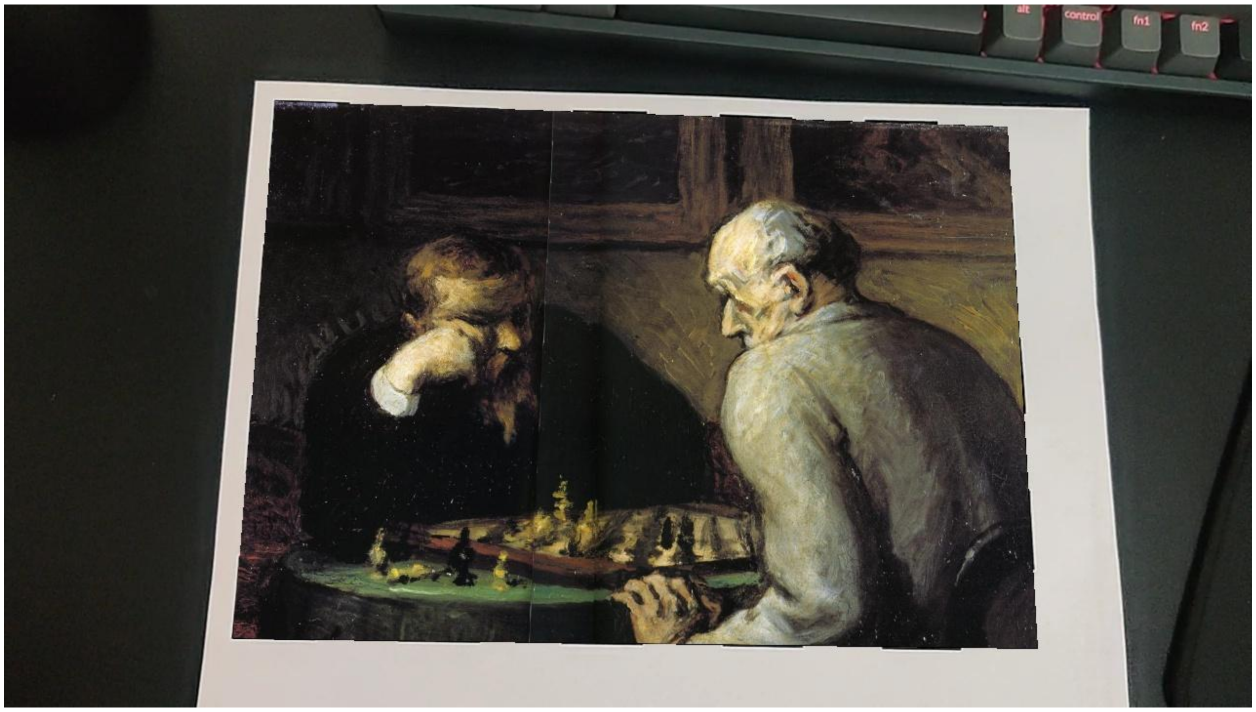
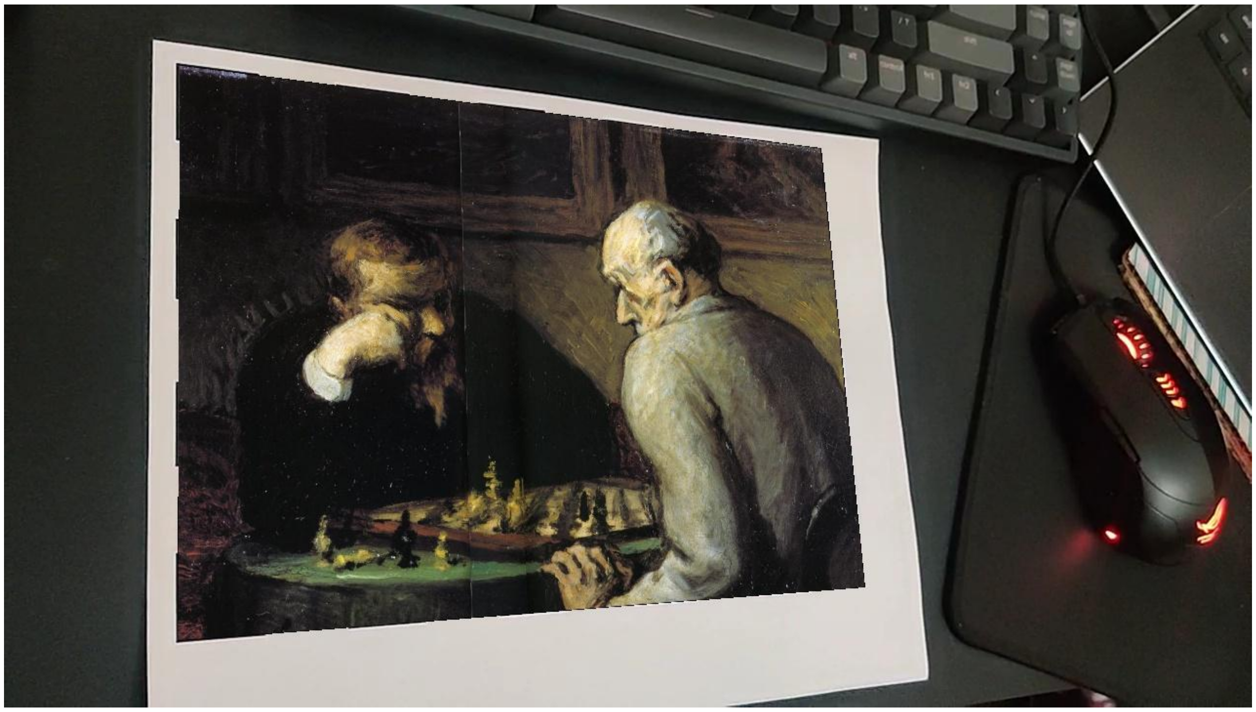


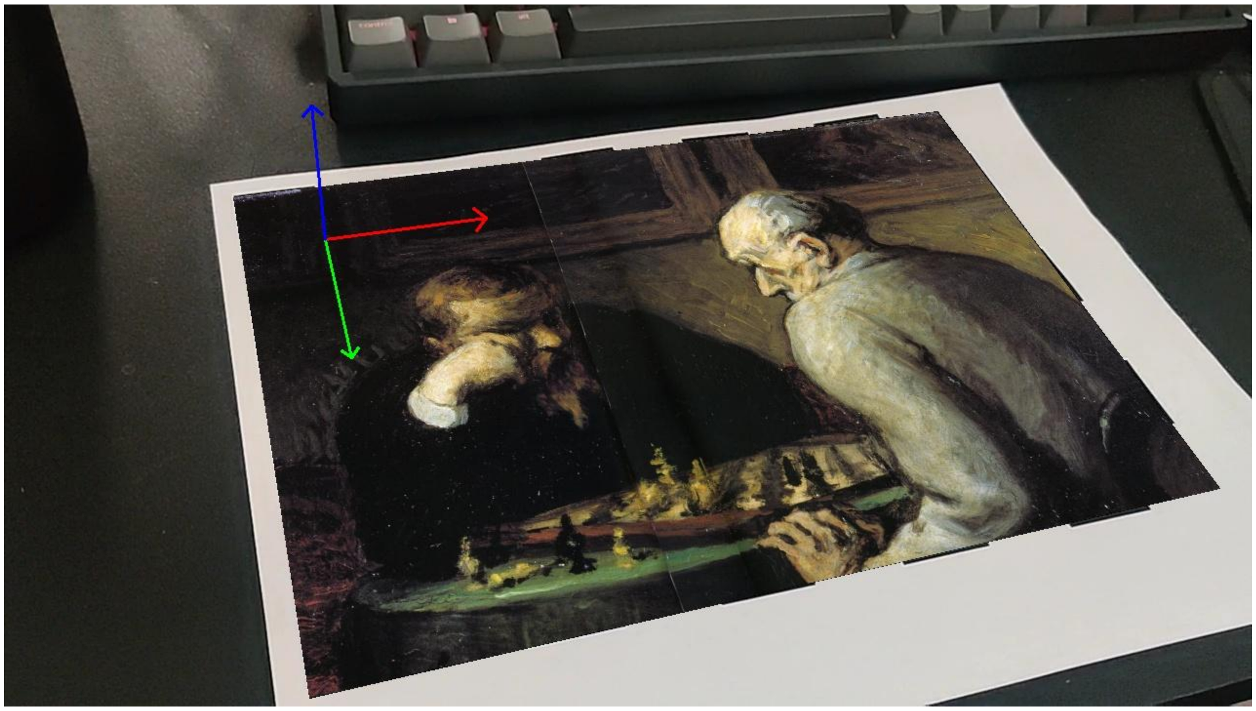
We observed a higher reprojection error for the circles pattern which resulted in the projected 3D objects being a little off from the markers detected in the pattern.

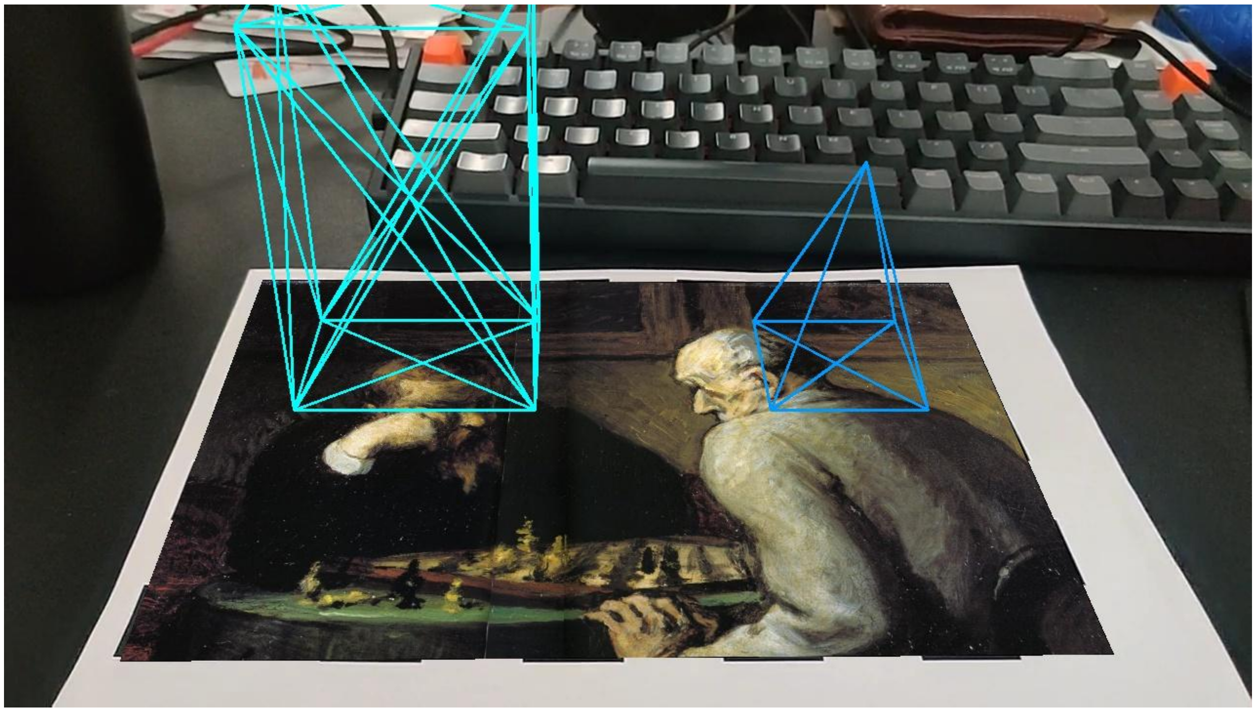
Extension 2: Do something to the target to make it not look like a target any more

For this extension we used an artwork as background for the virtual objects to be displayed instead of a patterned background.We used the 3D coordinate system to find outer corners of the pattern. Then we used the projectPoints method to find the outer coordinates in the actual image. Further we used getPerspectiveTransform to get the homography matrix to resize and change perspective of the outer coordinates of the artwork and map those to the outer coordinates of the identified pattern in the image. Finally we used warpTransform with the obtained homography matrix to transform the perspective of the artwork to fit the area of the pattern. We used a mask and some bitwise operations to apply this warped artwork to the image.
We used a painting of chess players to hide the pattern.

Example Images:

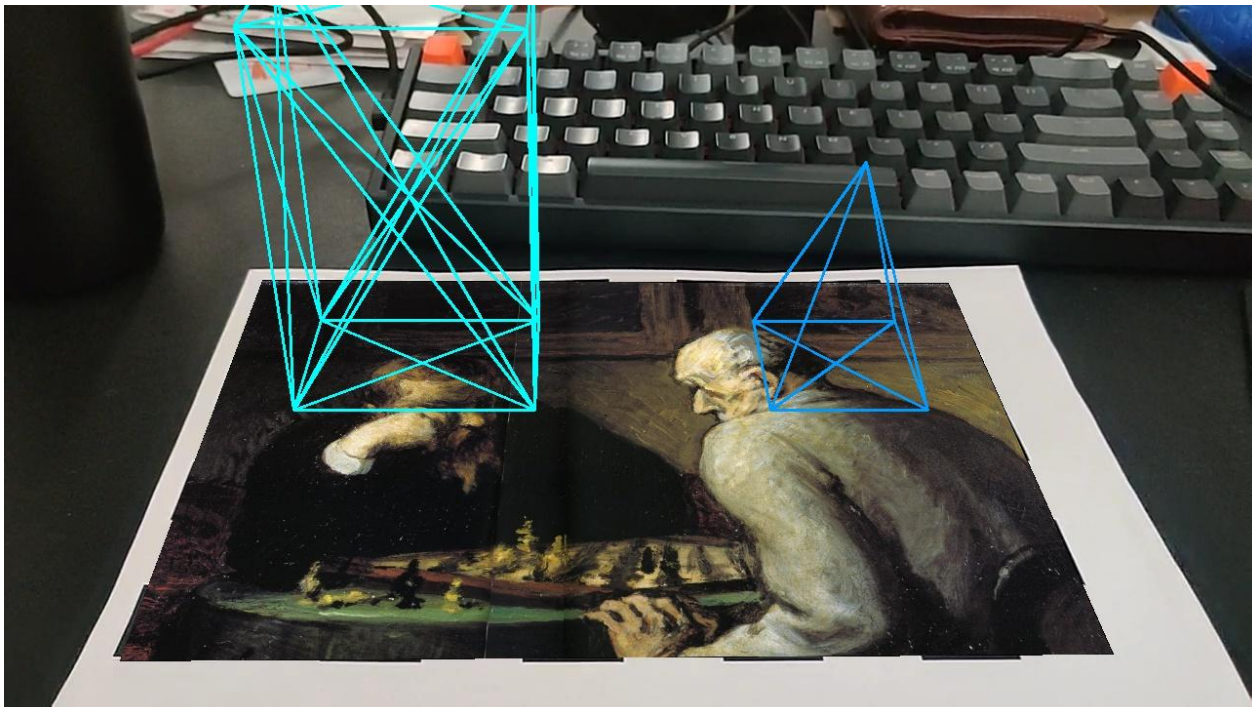



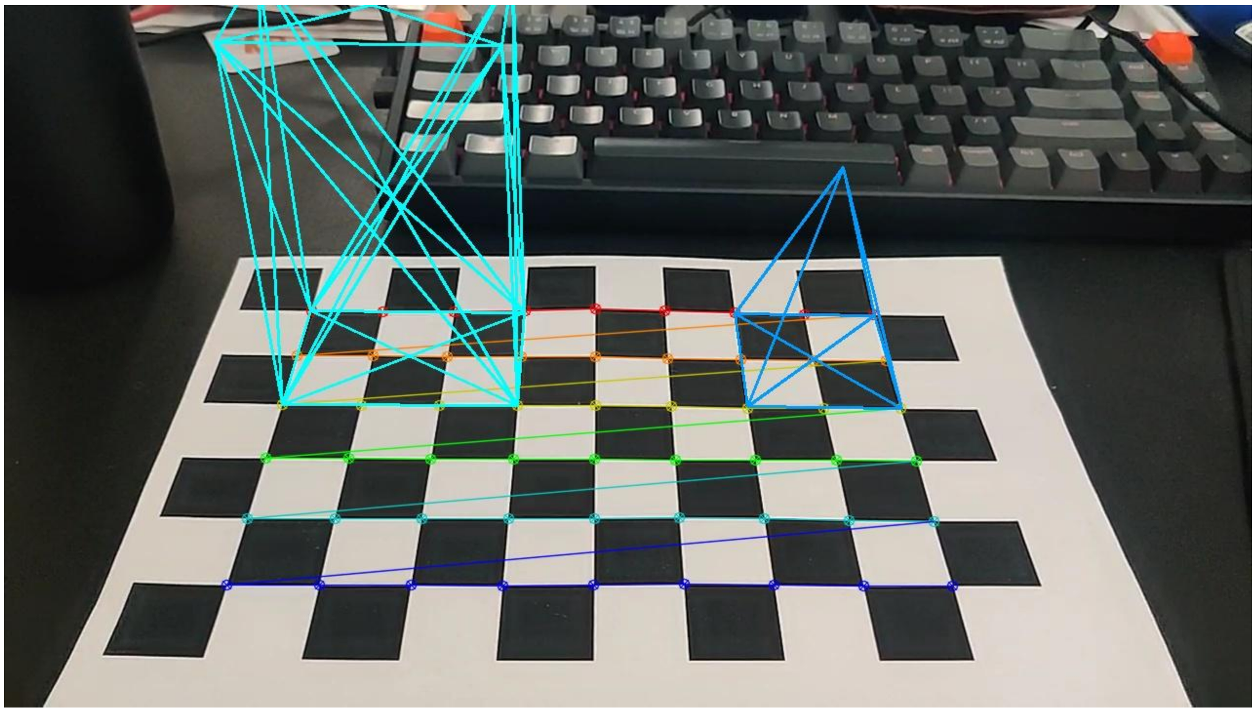




Extension 3: Get your system working with multiple targets or objects in the scene

In this extension we added multiple 3D objects to the target scene. We added a cuboid alongside a pyramid. The objects are displayed even when the target pattern is being masked by a painting.





Extension 4: Video Recording and Saving Images with the VR object projection

Users can use the video recording feature to record the video of the displayed effects. Instructions of usage in ReadME.

You can find a video demo here

Reflection:

By working on this project, we learned the different steps needed to create a virtual object on a patterned background. We could understand the algorithm behind detecting edges and saving different calibration features required for calibrating the camera.We learned the calibration method for the camera and used different patterns as background for displaying the 3D objects. We also used two different features for corner detection, Harris and Sub Pixel corner detection. We could create different 3D shapes just using lines on a patterned background as well as replacing that background with an artwork. This project opened up a lot of learning opportunities for us in the field of augmented reality.

Acknowledgements:

Firstly we would like to thank Prof. Bruce Maxwell for introducing this topic and always being available to clarify any doubts we encountered. Next we would like to thank the TAs, for helping us out understand the problems and helping us with the errors we kept getting. Finally we would like to mention a few sites that helped us understand few OpenCV and C++ functions:

- OpenCV: Camera calibration With OpenCV
- OpenCV: Camera Calibration and 3D Reconstruction
- OpenCV: Camera Calibration and 3D Reconstruction
- Advanced Camera Calibration Technique with C++ and OpenCV: A Practical Guide - YouTube
- OpenCV: Camera Calibration and 3D Reconstruction
- OpenCV: Camera Calibration and 3D Reconstruction
- Harris Corner Detection Explained | Baeldung on Computer Science