Syed Azhar Hussain Quadri

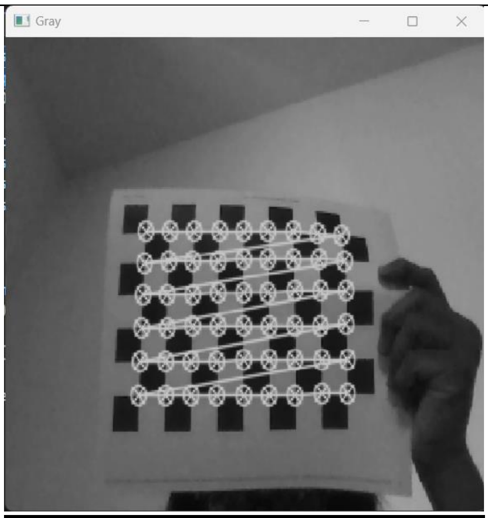
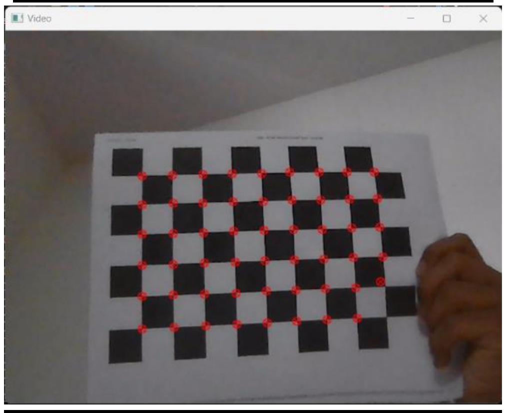
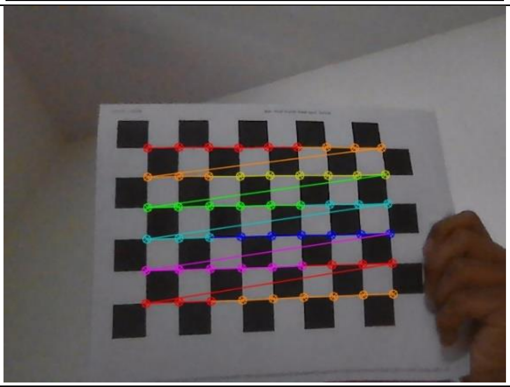
CS 5330 PRCV

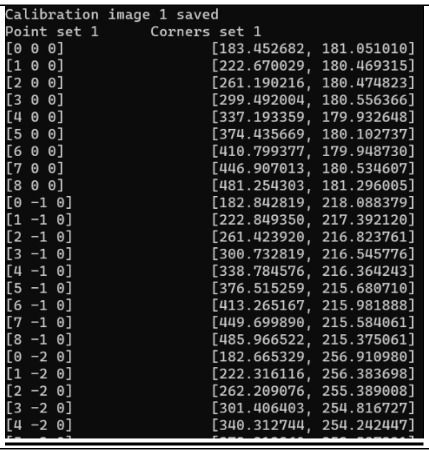
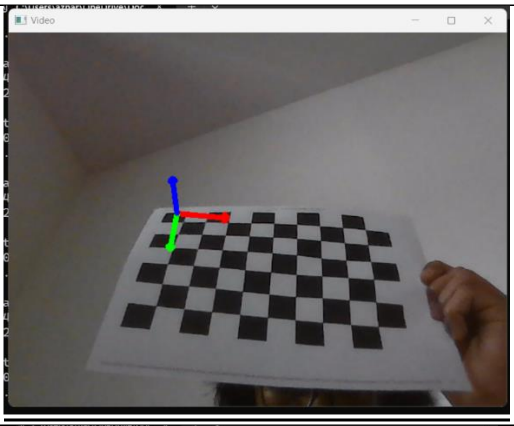
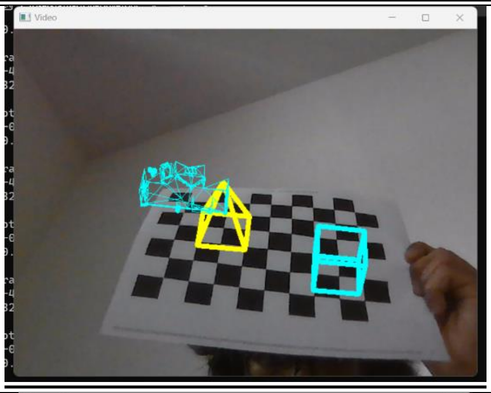
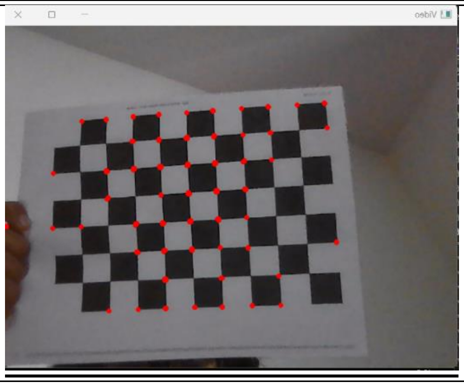
Project 4 Report

Calibration and Augmented Reality

The aim of this project is to learn how to calibrate a camera and apply the calibration to produce virtual objects in a scene. The desired outcome is a software application that can identify a target, and accurately position a virtual object in relation to the target, adjusting its orientation according to the motion of the camera or the target.

WORKING:

Task 1: Given a cv::Mat of the image frame, cv::Mat for the output and vector of points This function takes in an image frame, an output matrix, and a vector of points. The function detects the corners present in a checkerboard grid in the image frame, draws them on the output matrix and populates the vector with the coordinates of the detected corners.	 
Task 2: This function takes in a vector of points representing the image pixel coordinates of detected corners. The function converts these points to their corresponding world coordinates for a checkerboard target and populates the resulting vector with these world coordinates. The function also populates additional vectors to store multiple sets of points and corners, which will be used in calibration.	

Task 3: This function takes in vectors containing point sets and corner sets, as well as an initial camera matrix. The function then performs a calibration and calculates the calibrated camera matrix and distortion coefficients.	
Task 4: Given vector containing current point set and corner set, calibrated camera matrix and distortion coefficients, This function estimates the position of the camera relative to the target and populates arrays with rotation and translation data	
Task 5: Given a cv::Mat of the image frame, calibrated camera matrix, distortion coefficients, rotation and translation data from the current estimated camera position, this function projects 3D world coordinates of axes to image pixel coordinates on the image frame and draws lines between these points to generate the 3D axes at origin.	
Task 6: Given a cv::Mat of the image frame, calibrated camera matrix, distortion coefficients, rotation and translation data from the current estimated camera position, this function projects 3D world vertices of virtual shapes to image pixel coordinates on the image frame and draws lines between them to generate 3D virtual objects on the target.	
Task 7: This C++ function detects corners in an image using OpenCV library. It takes two image inputs and returns an integer. It sets variables for corner detection parameters, creates a grayscale copy of the input image, detects corners, normalizes the output, and draws circles around detected corners in the destination image. It's a simple and useful tool for detecting corners in an image.	

```
Calibration image 1 saved
Point set 1      Corners set 1
[0 0 0]                   [183.452682, 181.051010]
[1 0 0]                   [222.670029, 180.469315]
[2 0 0]                   [261.190216, 180.474823]
[3 0 0]                   [299.492004, 180.556366]
[4 0 0]                   [337.193359, 179.932648]
[5 0 0]                   [374.435669, 180.102737]
[6 0 0]                   [410.799377, 179.948730]
[7 0 0]                   [446.907013, 180.534607]
[8 0 0]                   [481.254303, 181.296005]
[0 -1 0]                  [182.842819, 218.088379]
[1 -1 0]                  [222.849350, 217.392120]
[2 -1 0]                  [261.423920, 216.823761]
[3 -1 0]                  [300.732819, 216.545776]
[4 -1 0]                  [338.784576, 216.364243]
[5 -1 0]                  [376.515259, 215.680710]
[6 -1 0]                  [413.265167, 215.981888]
[7 -1 0]                  [449.699890, 215.584061]
[8 -1 0]                  [485.966522, 215.375061]
[0 -2 0]                  [182.665329, 256.910980]
[1 -2 0]                  [222.316116, 256.383698]
[2 -2 0]                  [262.209076, 255.389008]
[3 -2 0]                  [301.406403, 254.816727]
[4 -2 0]                  [340.312744, 254.242447]
```

LEARNING AND REFLECTION

This project was such a great learning experience! I loved learning how to detect patterns and calibrate a camera using multiple images of the same scene from different angles. It was really cool to see how the camera calibration allowed us to project 3D objects onto a 2D image, and to learn how to generate rotation and translation matrices to keep the orientation of the 3D object intact relative to the checkerboard, even as the camera moved to a different angle.

Overall, this project was a great way to get hands-on experience with camera calibration and projection of 3D objects. It was really rewarding to see how different cameras affected our results and to experiment with different approaches to get the best possible outcome.

SOURCES AND REFERENCES

Stack overflow and OpenCV tutorials

https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c

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

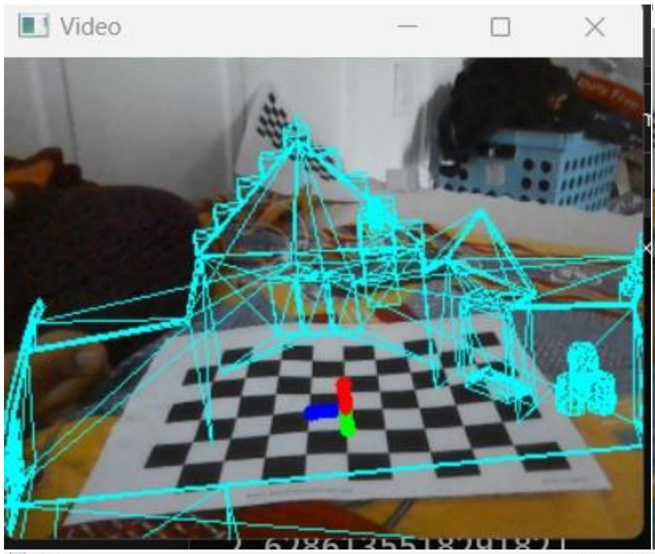
https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga3207604e4b1a1758aa66acb6ed5aa65d

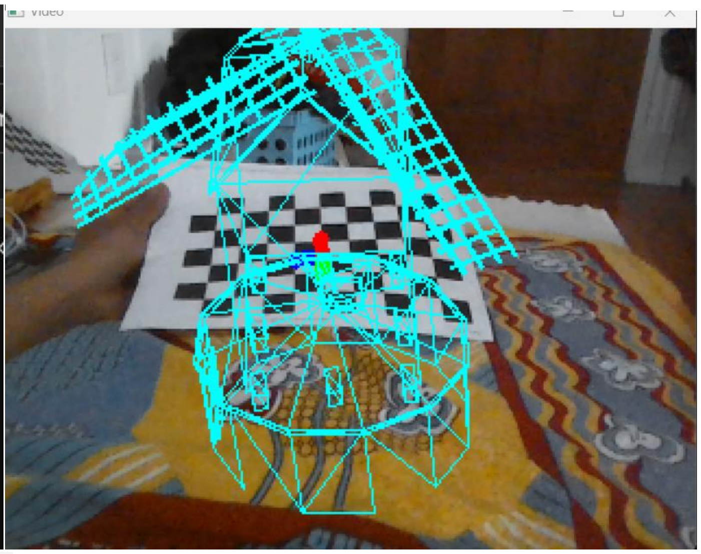
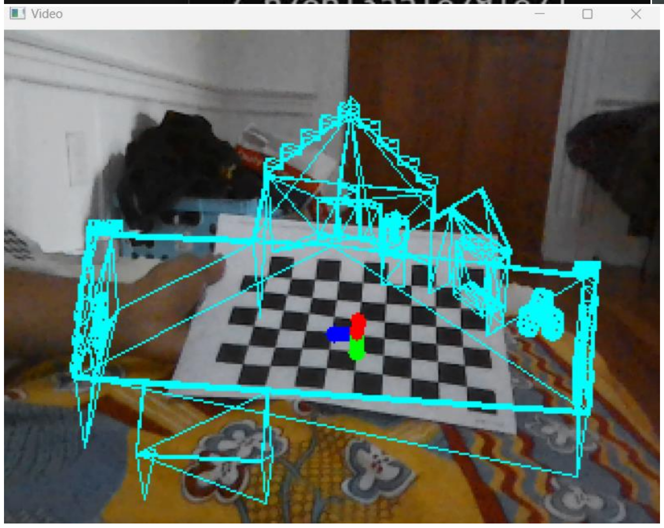
https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html

Extensions 1:

For the first extension I was able to load and project 3D wavefront objects, which is super exciting! Instead of just building simple 3D shapes using lines, we are now using a wavefront .obj file that contains the vertex coordinates of an object in the 3D world, as well as information about its triangle faces.

To accomplish this, we first parse the .obj file to create a vector of coordinates and store the vertex index from the face information. We then use this data to draw lines that build the triangular faces, ultimately resulting in the creation of the 3D object. This allows us to work with much more complex shapes and objects and gives us even more freedom to create exciting visual displays.

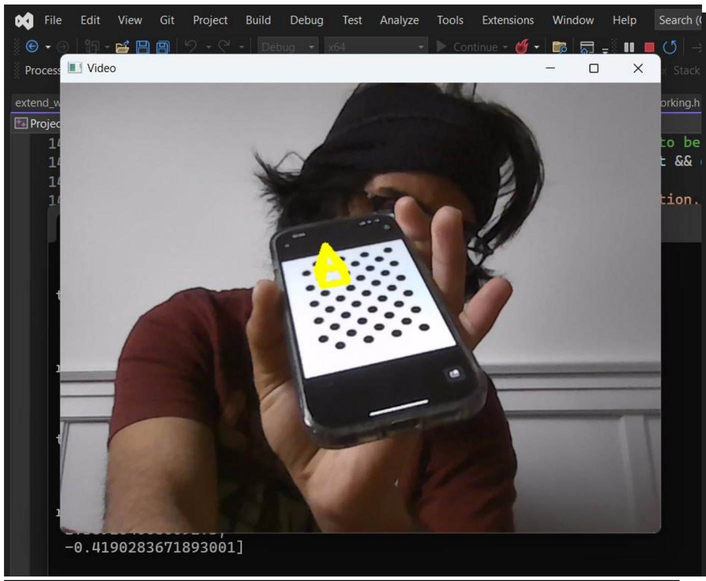


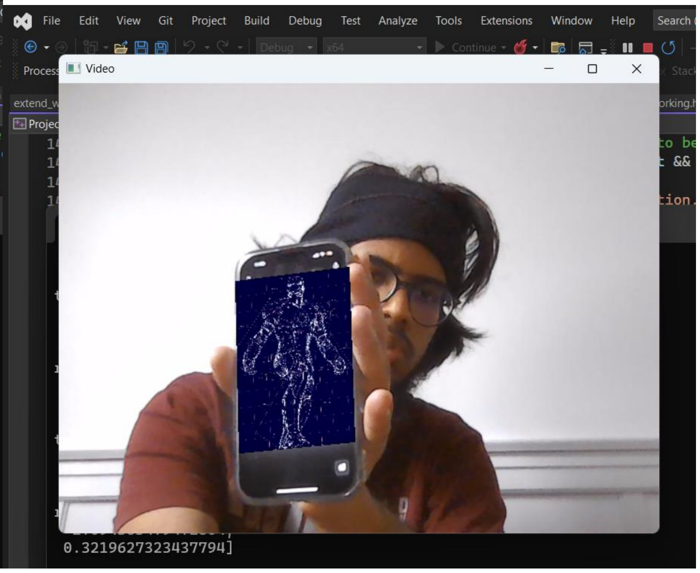
Extension 2:

I have now successfully implemented the use of an asymmetric circle grid as our target, rather than the checkerboard that was previously used. This has allowed me to expand my capabilities and work with a wider range of targets, giving even more freedom to create engaging and interactive visual displays.

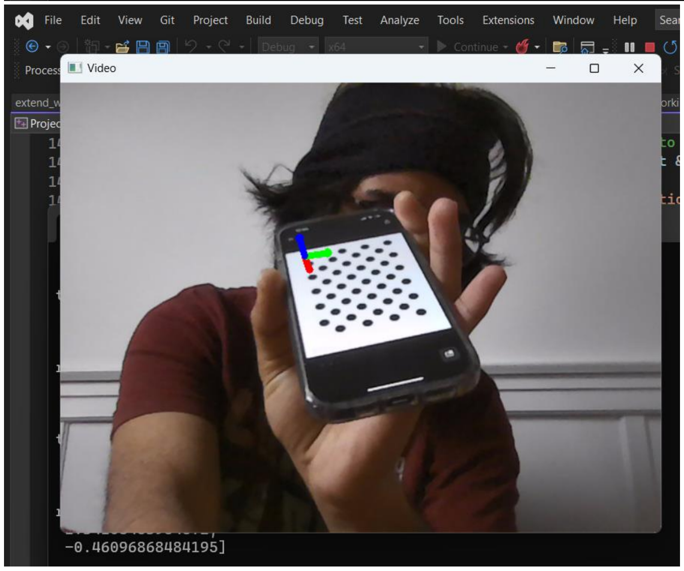
To take things even further, I have also been experimenting with ways to make the target itself less noticeable and more seamlessly integrated into the overall scene. This involves not only adding virtual objects to the scene, but also doing something to the target itself to make it less recognizable as a target. This could involve using visual effects to blend the target in with its surroundings or adding additional physical elements to the target that make it appear more like a natural part of the environment. By doing this, I was able to create even more immersive and realistic AR experiences that truly capture the imagination of the viewer.


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Video

```
-0.4190283671893001]
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Video

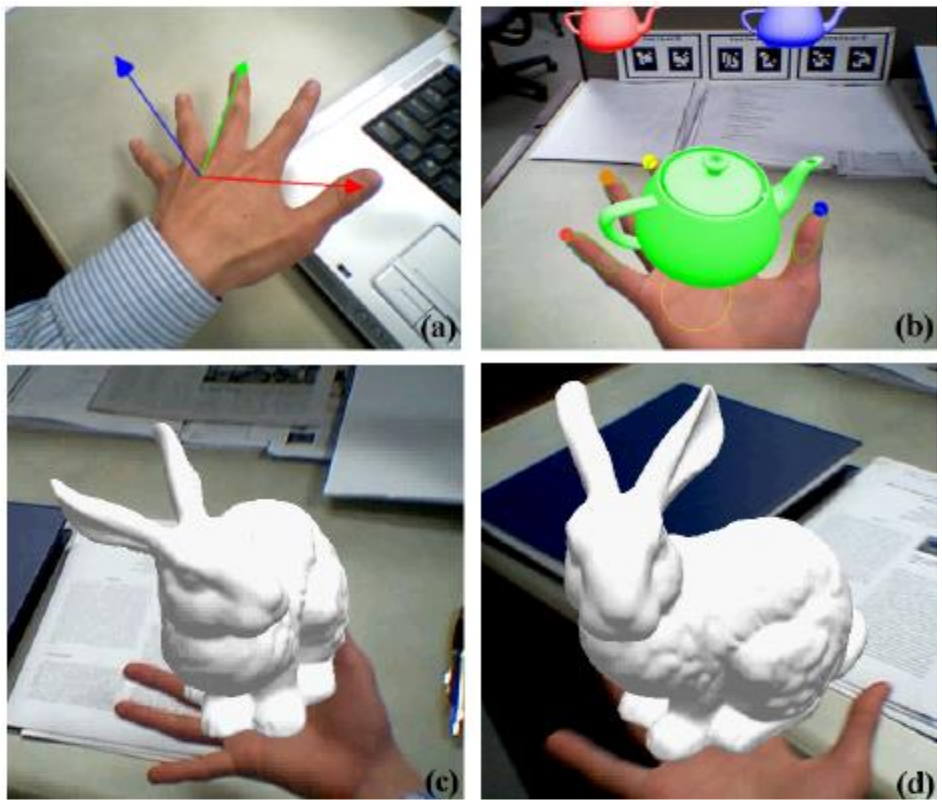
```
0.3219627323437794]
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Video

```
-0.46096868484195]
```

EXTENSION THAT I AM CURRENTLY TRYING AND WORKING ON:

Projecting AR on hand



(Image source from internet)