Project 4:Calibration and Augmented Reality
Pattern Recognition and Computer Vision (CS-5330)

Team Members:
Poojit Maddineni (NUID: 002856550)
Venkata Satya Naga Sai Karthik Koduru (NUID: 002842207)

1. **Pattern Detection and Corner Extraction**: We were amazed as to how the function quickly latched on to the corners, althoguh it wasn't perfect, when we dug a little deeper to find out how it works, it was a culmination of many other functions that each had a task like detecting squares then finding corners and also removing any other corners that wasn't inside the pattern.
2. **Calibration and Coordinate System:** In the second task we are saving the corner_set into a corner list and point_sets into point list. We were initially perplexed as to why we were saving these values butit became evidently clear when we moved on to the subsequent tasks as we were able to calibrate the camera.
3. **Camera Calibration:** The previous tasks gave us the foundation to perform the calibration process using `calibrateCamera` function, we obtained intrinsic parameters like camera matrix and distortion coefficients. We learned that it was necessary step as it removes distortion both radial and tangential and also relate the 3d world to the image plane.
4. **Pose Estimation:** Using functions like `solvePnP`, we calculated the rotation and translation vectors, it helped us guage the position of camera with respect to the target.
5. **Projection of 3D Points:** We projected the 3D world points onto the image plane in real-time using `projectPoints`, enabling us to visualize the spatial relationship between the target and camera.
6. **Creation of Virtual Objects:** We implemented virtual objects like tetrahedron, dodecahedron, a car, and a simple cube, and overlaid them onto the image plane, understanding the process of transforming 3D object coordinates to image coordinates.
7. **Feature Detection and Augmented Reality:** By utilizing Harris corner detection, we extended our project to detect corners on new patterns, such as those drawn on an iPad. This helped us understand how the Augumented Reality apps will find the corners and take the feature points with out a target pattern.

On a whole, the project was really exciting as we learnt how the 3d world coordinates are related to that of the image plane, how to measure things using the cameras.

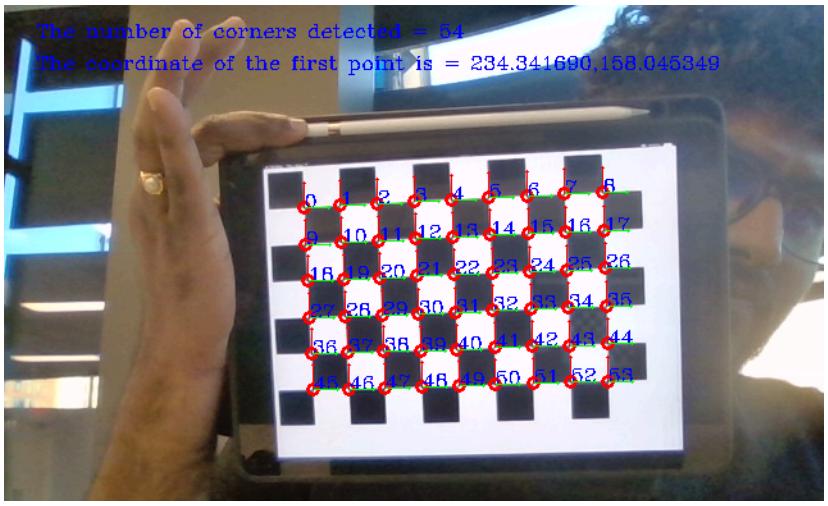
Workspace Setup:

For our workspace setup, we printed the checkerboard pattern on A4 paper also we used an Ipad screen with a picture of the same pattern and created patterns on our iPad for Task 7. To stream the video feed, we used the Iriun app, enabling live video transmission from a mobile device to a PC. Additionally, we utilized our laptop's webcam for video streaming in all other tasks, ensuring versatility and efficiency throughout our project.

Tasks:

Task 1: Detect and Extract Target Corners

In this task, we used the provided checkerboard pattern as the target. We used OpenCV's `findChessboardCorners` function to accurately detect the corners of a checkerboard pattern measuring 9 by 6, totaling 54 corners. The same can be seen the image below.



```
The number of corners detected = 54
The coordinate of the first point is = 234.341690,158.045349
```

We checked using the function cornerSubPix to see if there is any considerable refinement in the positions of the corners, but we didnt see much, so we opted to not use it.

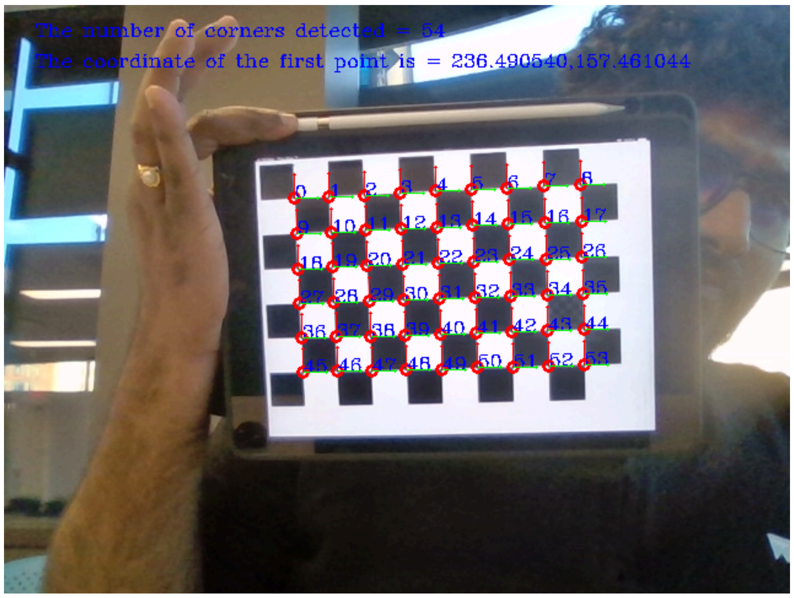
We wrote our program to mark the corners and to draw a coordinate system for better visualizaation. This way were were also able to understand if the corners are being tracked properly and if the order we assumed was right.

We did not face any challenges in particular, but some limitations we observed is that the viewing angle for the chess board is really shallow, meaning if the pattern deviates more than a certain value, the chessboard corners would not be detected even though they are visible. Also, the lighting conditions also played a crucial role as a bad glare on the ipad screen would make the program not detect the pattern.

Task 2: Select Calibration Images

In Task 2, we wrote a custom function named `generate_3d_points`. This function will generate the 3d points based on the size of the pattern. We made sure to note this point(Note that if the (0, 0, 0) point is in the upper left corner, then the first point on the next row will be (0, -1, 0) if the Z-axis comes towards the viewer) and followed the same logic.

When the user presses “s”, the frame will be saved and an acknowledgement is given to help the user identify that the file has been saved. We used have saved around 8 images for calibrating.

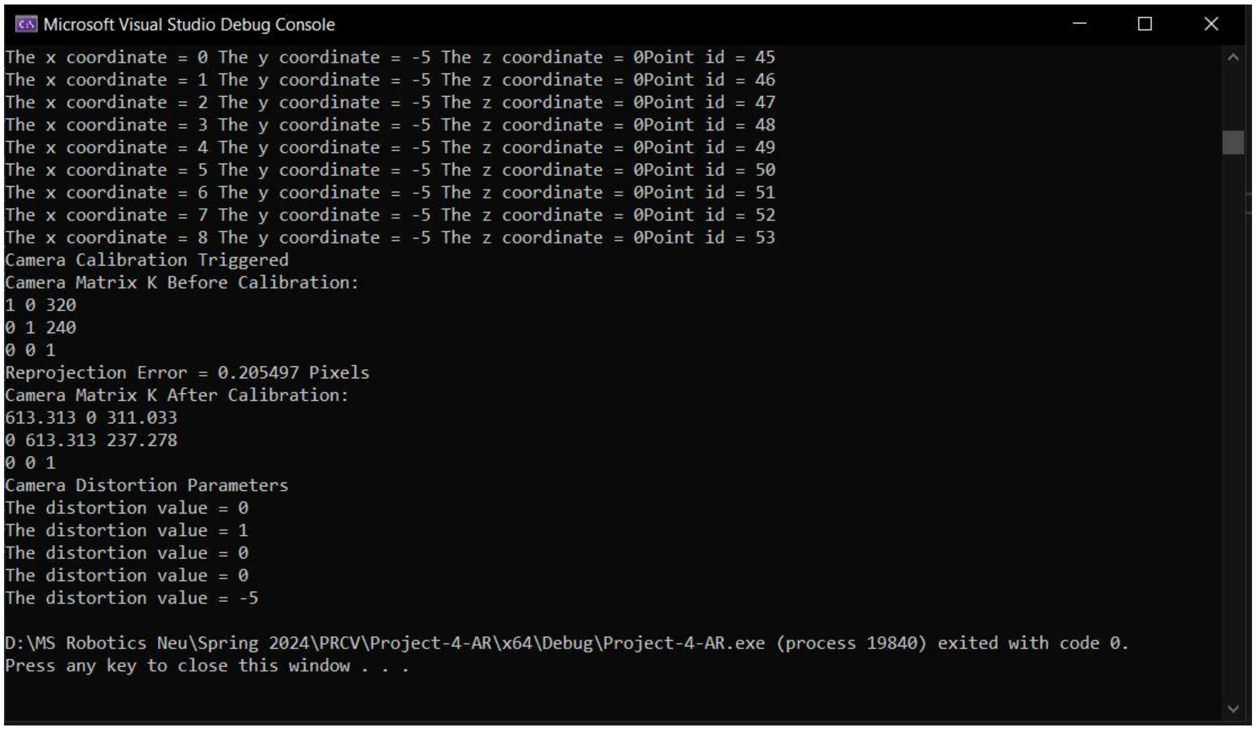


```
The number of corners detected = 54
The coordinate of the first point is = 236.490540,157.461044
```

Task 3: Calibrate the Camera

Task 3 involved calibrating the camera using the OpenCV calibrateCamera function. We used 8 images for calibration and added a check point for meeting the minimum number of the images required for calibration. If the requirement is not met, the calibration function will not run.

The user can calibrate the camera by pressing “c” once the minimum requirement for the number of images is met. Then the camera intrinsic parameters are written to an “.yml” file. Both the camera matrix(K) and the the distroriton parameters(k) are stored in it. Below is an screen shot of the initial camera matrix, camera matrix after calibration and the distortion parameters. We observed the **Reprojection Error to be 0.205 pixels**.



```
The x coordinate = 0 The y coordinate = -5 The z coordinate = 0Point id = 45
The x coordinate = 1 The y coordinate = -5 The z coordinate = 0Point id = 46
The x coordinate = 2 The y coordinate = -5 The z coordinate = 0Point id = 47
The x coordinate = 3 The y coordinate = -5 The z coordinate = 0Point id = 48
The x coordinate = 4 The y coordinate = -5 The z coordinate = 0Point id = 49
The x coordinate = 5 The y coordinate = -5 The z coordinate = 0Point id = 50
The x coordinate = 6 The y coordinate = -5 The z coordinate = 0Point id = 51
The x coordinate = 7 The y coordinate = -5 The z coordinate = 0Point id = 52
The x coordinate = 8 The y coordinate = -5 The z coordinate = 0Point id = 53
Camera Calibration Triggered
Camera Matrix K Before Calibration:
1 0 320
0 1 240
0 0 1
Reprojection Error = 0.205497 Pixels
Camera Matrix K After Calibration:
613.313 0 311.033
0 613.313 237.278
0 0 1
Camera Distortion Parameters
The distortion value = 0
The distortion value = 1
The distortion value = 0
The distortion value = 0
The distortion value = -5

D:\MS Robotics Neu\Spring 2024\PRCV\Project-4-AR\x64\Debug\Project-4-AR.exe (process 19840) exited with code 0.
Press any key to close this window . . .
```

Task 4: Calculate Current Position of the Camera

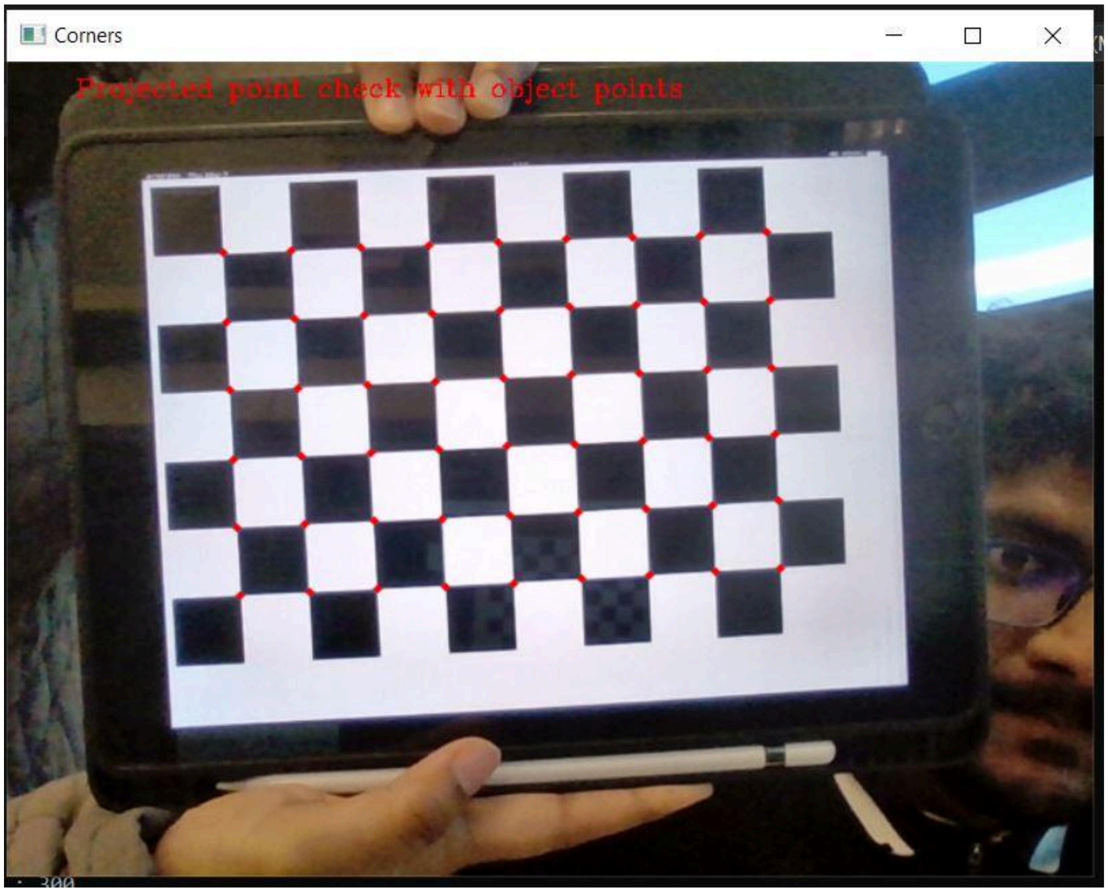
Task 4 involves calculating the current position of the camera using the solvePnP function. This function estimates the camera's pose (rotation and translation) relative to the target using provided 3D world points and their corresponding 2D image points.

The camera intrinsic parameters are loaded form the yml file and read from it. These parameters are then passed to solvePnP function for the pose calculation of the camera. We printed out these values to understand how they vary. The x,y,z values of the vary when moving the pattern or the camera side to side. The values of x increases positively in the direction of +ve x axis and same with y axis and since the z axis is coming out of the screen, or out of the pattern towards the user, the z value increases of the camera is moved away from the pattern.

Similarly, the rvecs also change when rotating the camera or the pattern about any axis of the camera or the pattern. This makes complete sense and the data aligns in the way we are moving the target pattern or the camera.

Task 5:Project Outside Corners or 3D Axes

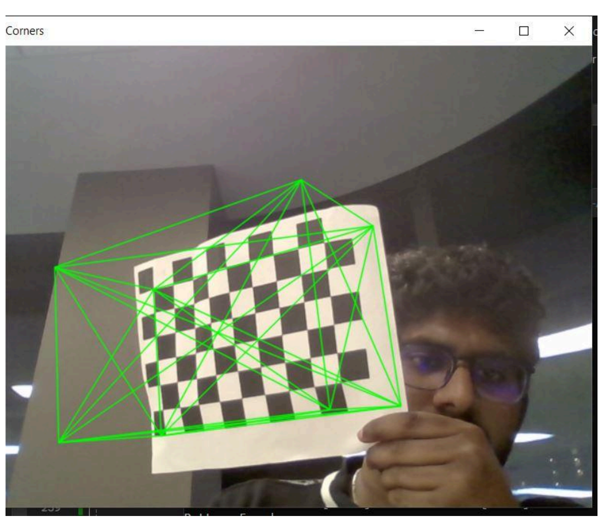
Task 5 involved projecting 3D world points onto the image plane using the projectPoints function. The projected points from the checkerboard pattern accurately appeared on the image plane. We have used the generated object points to check if the points are being projected at the right places. The red dots that can be seen at the corners are in perfect alignment with the checker board.

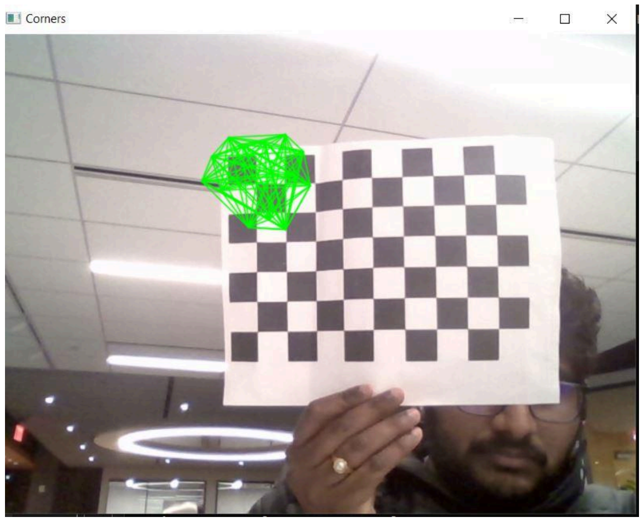


Task 6:Create a Virtual Object

We have inserted a cuboid shaped figure to for visulaization. The cuboid stays in place when the camera is moved around meaning that the object is properly located and we can see around the object when we move the camera around. The same can be observed in the video too.
Link to the [video]()

We have also created car and a tetrahedron figure. Some screen shots of objects such as cuboid and decaheedron are below.





Task 7:Detect Robust Features

Task 7 delved into employing Harris corner detection for identifying corners in hand-drawn patterns captured via a smartphone camera. Initially, a high threshold constrained corner detection, necessitating significant intensity changes for identification. Gradually lowering the threshold improved detection, increasing sensitivity to subtle intensity variations. First a diamond pattern was used, where multiple corners tended to cluster at certain points. This clustering phenomenon occurred due to the converging lines of the diamond shape, leading to regions with heightened intensity variations. Consequently, the Harris corner detection algorithm interpreted these regions as potential corner locations, resulting in multiple corners being detected in close proximity. Despite the clustering, the algorithm effectively identified the corner-rich regions of the pattern, showcasing its capability to detect features even in complex and intricate patterns. Optimal corner detection was achieved by setting the threshold between 120 to 150, achieving a balance between detecting sufficient corners within the pattern and minimizing false detections. This task emphasized the significance of parameter optimization in Harris corner detection, showcasing its adaptability to diverse image conditions and pattern complexities.
Link to the video for task - 7

How can we use these for placing virtual objects?

These feature points can be used as basis for placing augmented reality objects where there are no patterns to track. What we can do is get the feature points using harris corners and then use optiflow to track these corners over time to maintain and remember the way the camera is moving and the do the usual calibration and projection of the points to place them virtual objects.

Extensions:

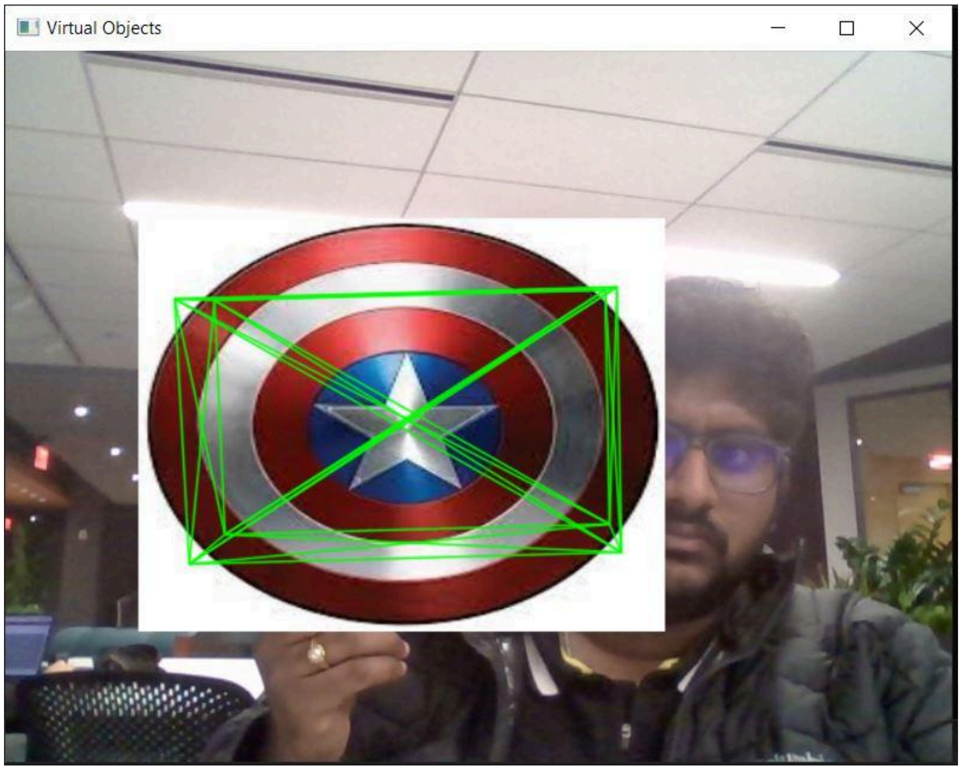
Extension-1

Not only add a virtual object, but also do something to the target to make it not look like a target any more.

In our project extension, we enhanced the overlay functionality by introducing a dynamic method for determining the region of interest (ROI) using the `overlayImage` function. This function facilitates precise placement of a foreground image onto a background, ensuring that the location and size are within valid bounds to prevent runtime errors. The program dynamically identifies the outermost corners of a checkerboard pattern, enabling us to expand the ROI to cover the entire A4-sized paper. This approach guarantees full coverage of the pattern area, enhancing the accuracy and effectiveness of the overlay process.

Pressing ‘m’ will toggle the mask on or off.

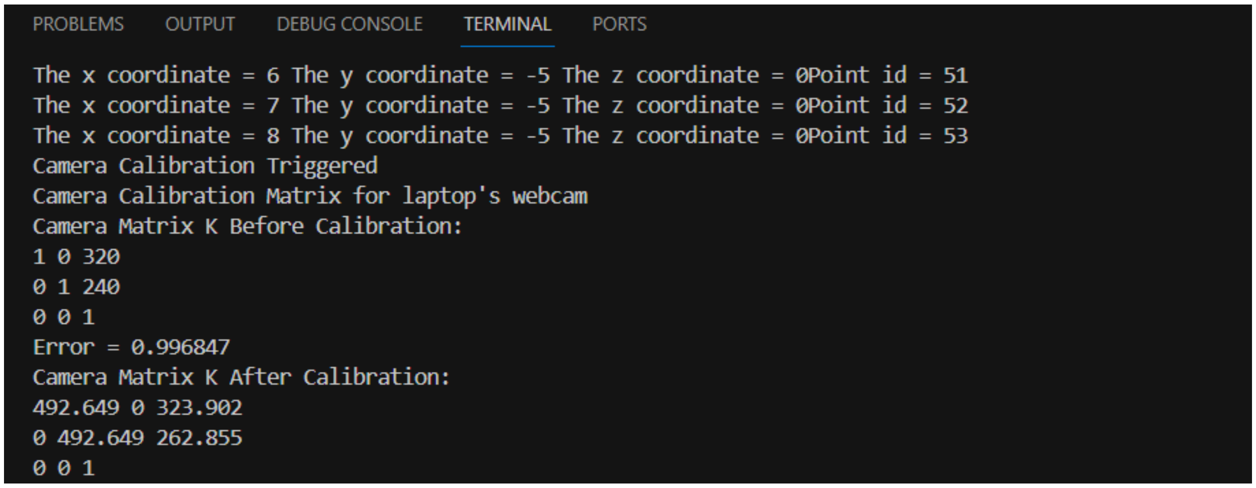
Link to the video



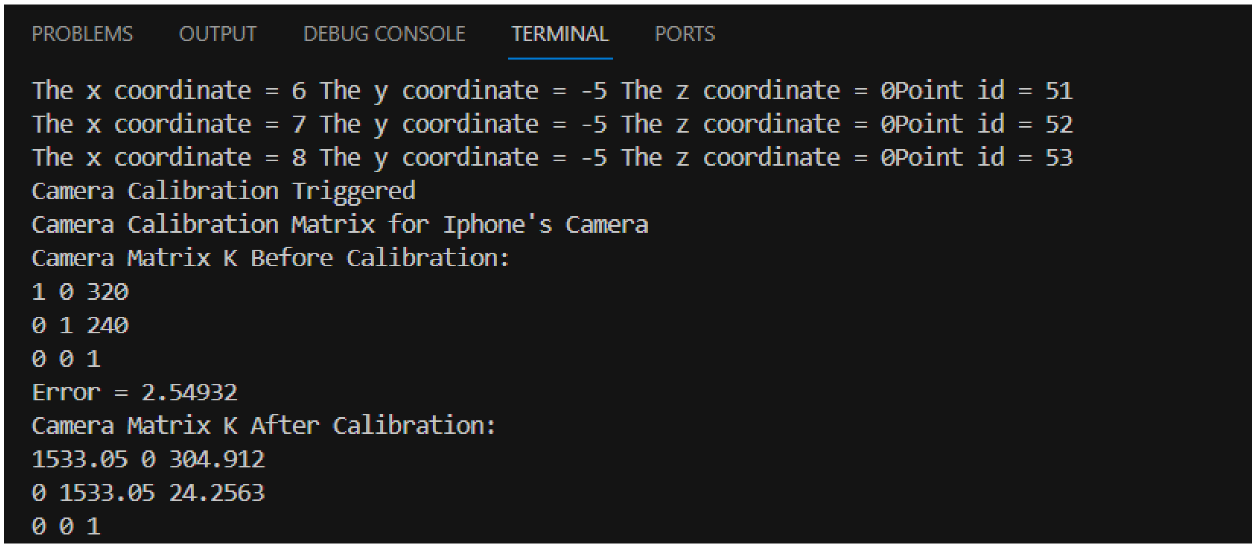
Extension 2:

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

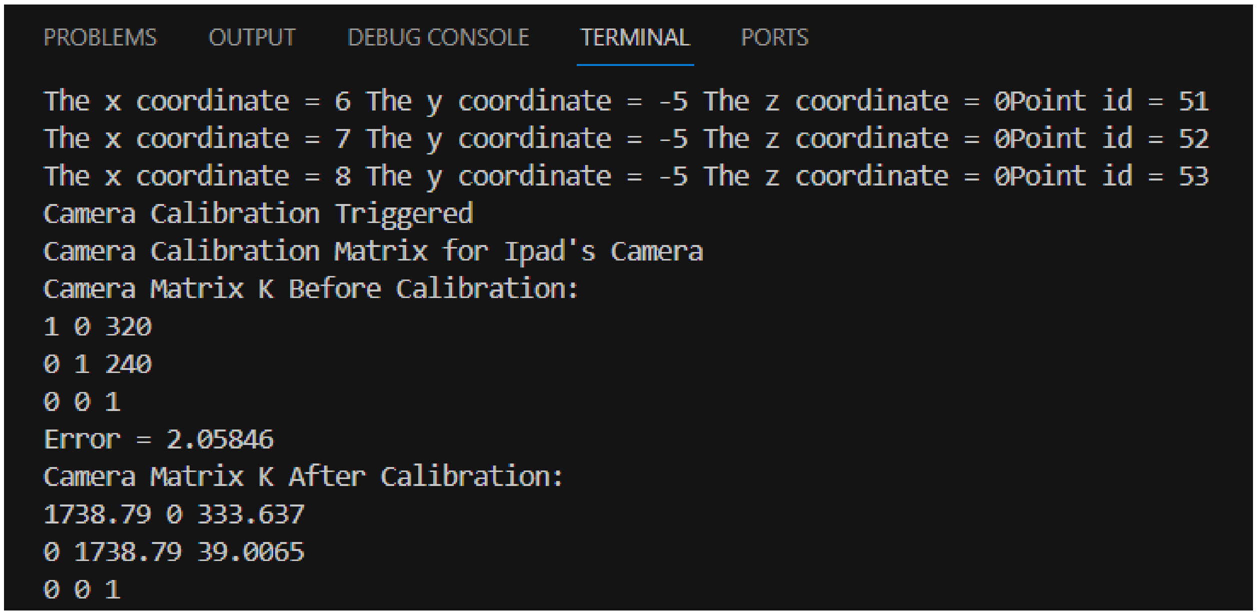
Camera Calibation Matrix for Laptop Webcams, Iphone and an Ipad



```
The x coordinate = 6 The y coordinate = -5 The z coordinate = 0Point id = 51
The x coordinate = 7 The y coordinate = -5 The z coordinate = 0Point id = 52
The x coordinate = 8 The y coordinate = -5 The z coordinate = 0Point id = 53
Camera Calibration Triggered
Camera Calibration Matrix for laptop's webcam
Camera Matrix K Before Calibration:
1 0 320
0 1 240
0 0 1
Error = 0.996847
Camera Matrix K After Calibration:
492.649 0 323.902
0 492.649 262.855
0 0 1
```



```
The x coordinate = 6 The y coordinate = -5 The z coordinate = 0Point id = 51
The x coordinate = 7 The y coordinate = -5 The z coordinate = 0Point id = 52
The x coordinate = 8 The y coordinate = -5 The z coordinate = 0Point id = 53
Camera Calibration Triggered
Camera Calibration Matrix for Iphone's Camera
Camera Matrix K Before Calibration:
1 0 320
0 1 240
0 0 1
Error = 2.54932
Camera Matrix K After Calibration:
1533.05 0 304.912
0 1533.05 24.2563
0 0 1
```



```
The x coordinate = 6 The y coordinate = -5 The z coordinate = 0Point id = 51
The x coordinate = 7 The y coordinate = -5 The z coordinate = 0Point id = 52
The x coordinate = 8 The y coordinate = -5 The z coordinate = 0Point id = 53
Camera Calibration Triggered
Camera Calibration Matrix for Ipad's Camera
Camera Matrix K Before Calibration:
1 0 320
0 1 240
0 0 1
Error = 2.05846
Camera Matrix K After Calibration:
1738.79 0 333.637
0 1738.79 39.0065
0 0 1
```

In this extension, we calibrated four different cameras: two laptop webcams, an iPhone, and an iPad.To ensure robust calibration, we meticulously selected 12 images of tbe checkerboard pattern captured from various positions and orientations for each camera. The calibration matrices stored in yml files reveal differences in intrinsic camera parameters such as focal length and distortion coefficients. Variations in these parameters affect corner detection accuracy and overall image quality.While the iPhone and iPad cameras have higher focal lengths compared to the webcam, potentially offering advantages in capturing distant objects and achieving greater depth of field, there was no discernible difference in performance among the three cameras; all performed equally well. However, it's important to note that the term "better" is subjective and depends on specific application requirements.The quality of a camera also depends on factors such as sensor size, pixel density, lens quality, and image processing capabilities. While the iPhone and iPad cameras may excel in certain scenarios, such as photography and videography, they may not be optimal for all computer vision tasks. Additionally, the suitability of a camera depends on the specific requirements of the task at hand.

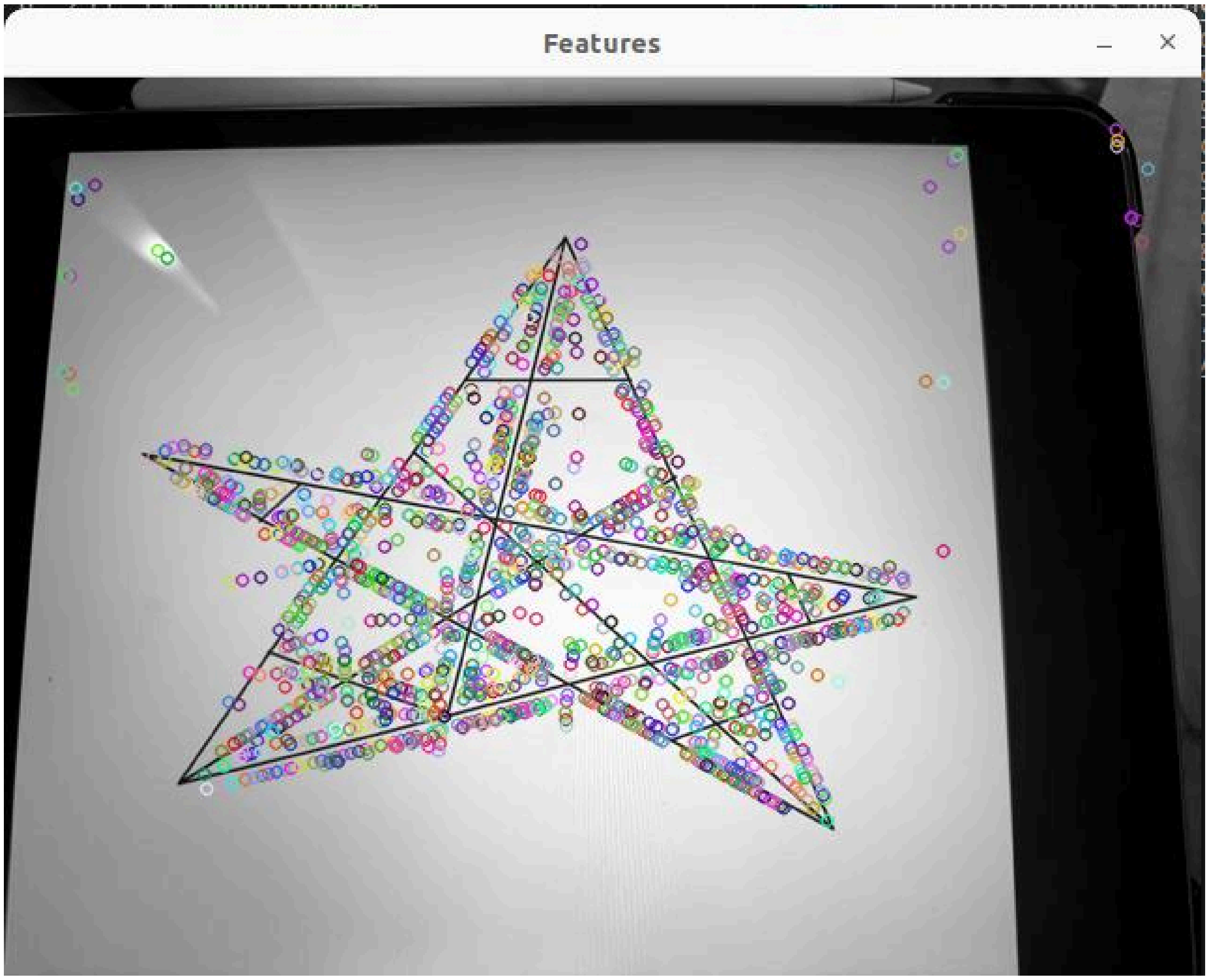
Extension 3: Inserting objects into precaptured videos

The below is the video link to the raw video and the video with the object inserted.

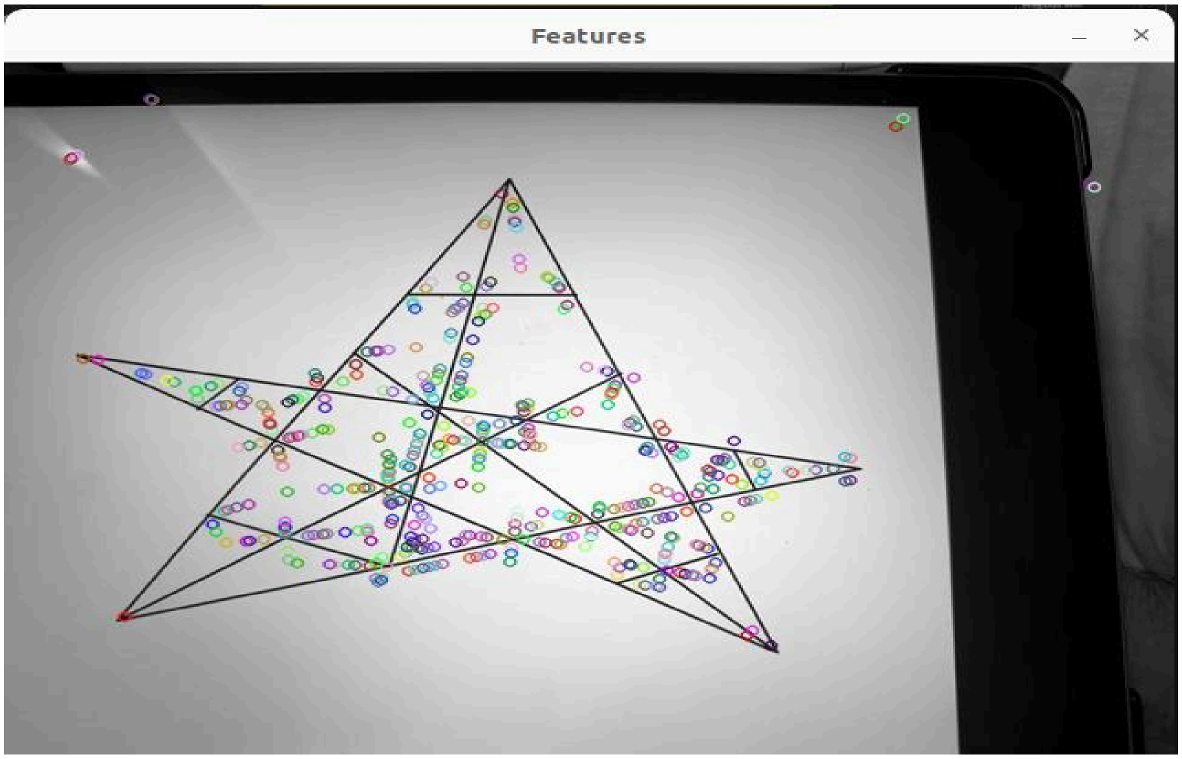
Raw Video

Video with object inserted

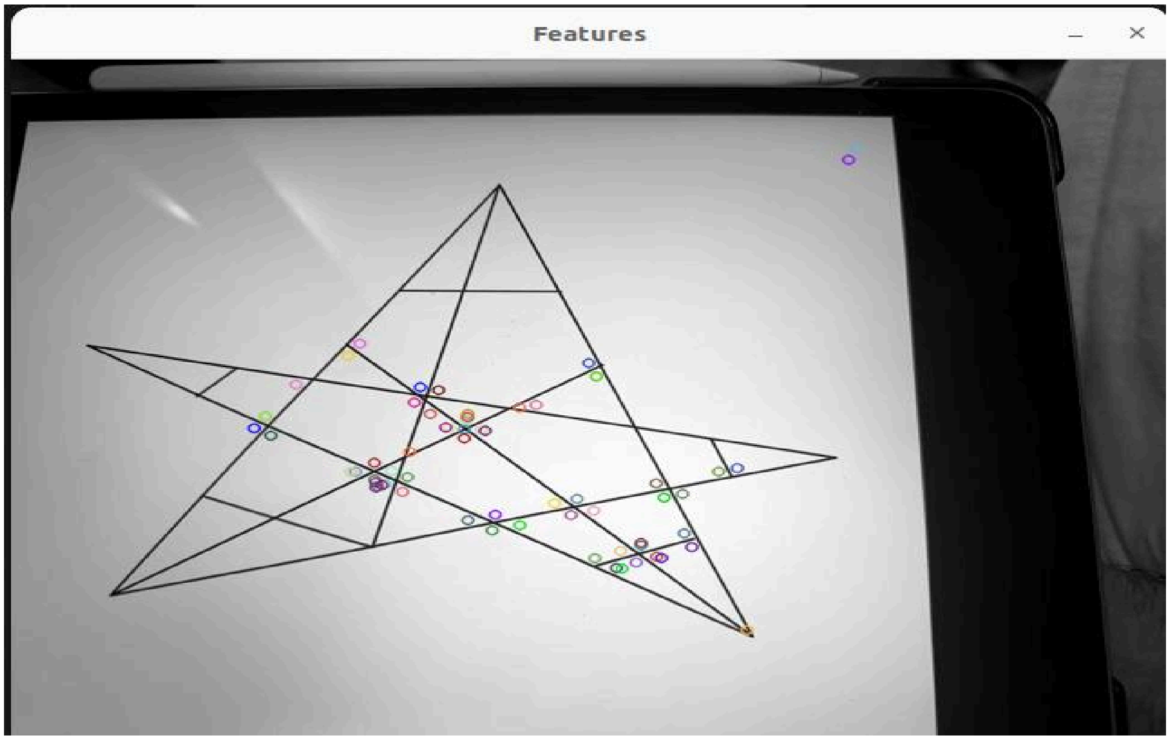
Extension 4: Implementing SURF and SIFT algorithms:
In this extension, we experimented with SURF and SIFT algorithms using the same pattern used Task 7. With SURF, we adjusted the threshold: at 400, many keypoints were detected, while at 1000 and 5000, fewer were found, some outside the pattern. These algorithms help identify distinctive points in images. SIFT is similar to SURF but computes more and is better at handling rotation. However, SIFT is slower due to its extensive calculations. Both are crucial in computer vision for recognizing objects and scenes. Our experiences highlight the importance of choosing the right algorithm based on task requirements. Understanding these tools' strengths and limitations helps in better image processing.



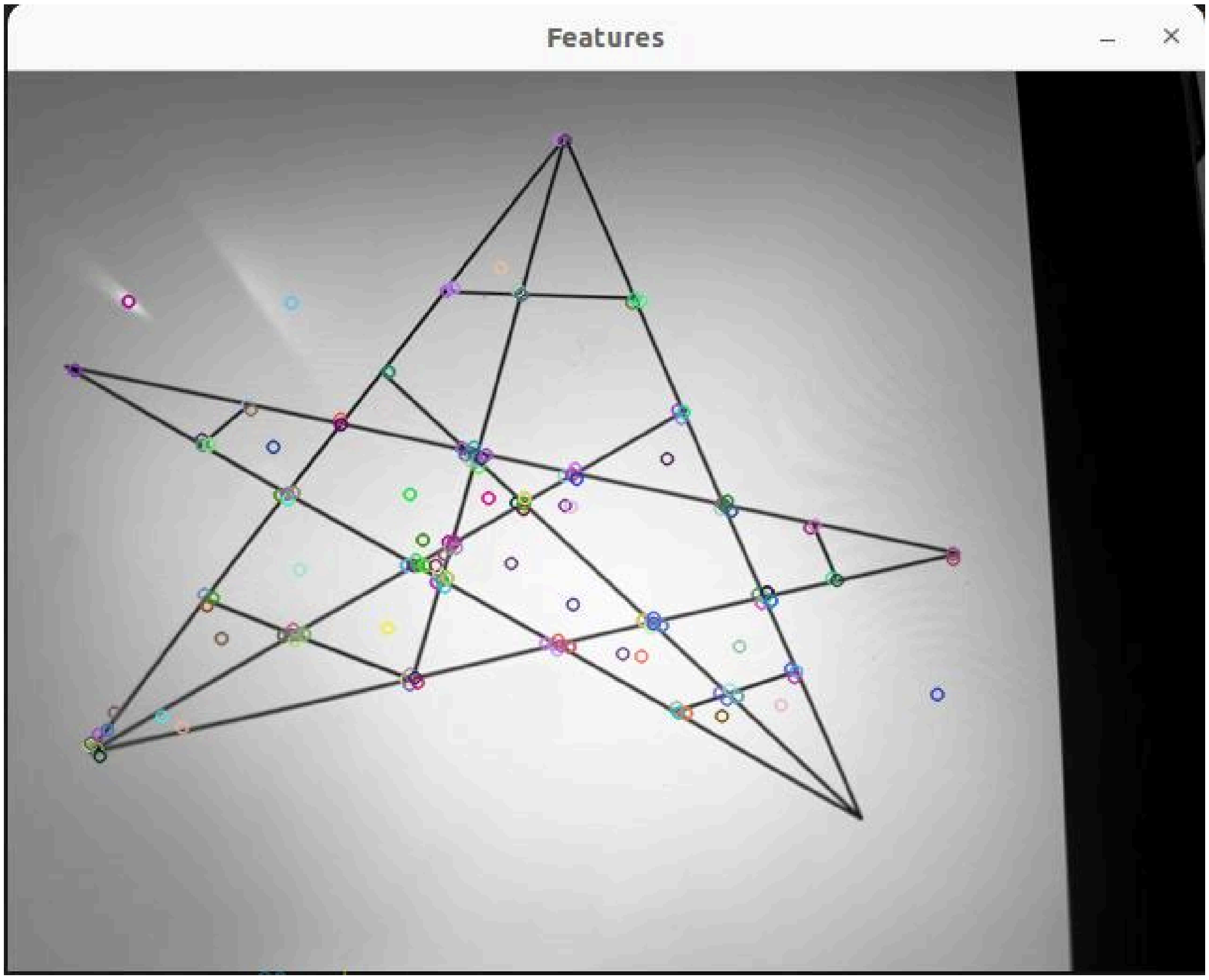
SURF ,Threshold value:400



SURF, Threshold value:1000



SURF, Threshold value:5000



SIFT Algorithm

Acknowledgment:

We would like to express our gratitude to Professor Bruce Maxwell for his lectures. His teaching was instrumental in helping us think constructively about our solution. We also extend our thanks to Stack Overflow and to all the developers and contributors on the platform. The wealth of resources available there was invaluable during our debugging process. We appreciate the assistance of OpenAI's ChatGPT, which was useful for conducting deeper research on the general topic. Additionally, we recognize the contributions of the numerous technical writers on Medium who have shared their expertise in computer image processing through their writings.

References:

- https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html
- https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c
- Youtube-https://www.youtube.com/watch?v=E5kHUs4npX4
- YouTube - https://www.youtube.com/watch?v=bs81DNsMrnM
- YouTube - https://www.youtube.com/watch?v=pDImLazOPrQ
- https://code.visualstudio.com/docs/cpp/cpp-debug
- https://www.reddit.com/r/computervision/
- https://en.wikipedia.org/wiki/Harris_corner_detector
- https://medium.com/@webnxoffice/augmented-reality-ar-and-its-applications-ca1a2ed484b8
- Google scholar- http://www.cs.yorku.ca/~kosta/CompVis_Notes/harris_detector.pdf
- https://medium.com/@deepanshut041/introduction-to-harris-corner-detector-32a88850b3f6
- https://www.youtube.com/watch?v=26nV4oDLiqc
- https://www.youtube.com/watch?v=H5qbRTikxI4