CS 5330 PATTERN RECOGNITION AND COMPUTER VISION
Project 4: Calibration and Augmented Reality

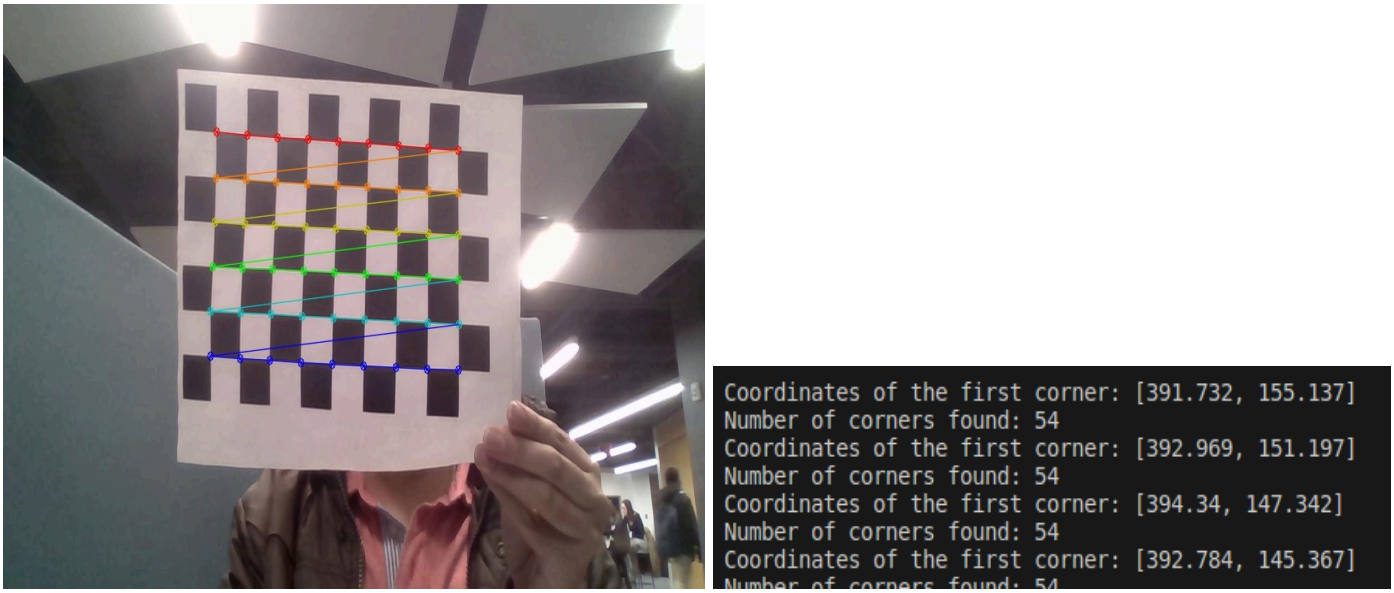
Ruchik Jani 002825482

1) Project description:

The goal of the project is to develop a camera calibration and augmented reality system using OpenCV. Initially, the system detects and extracts corners from a target, such as a checkerboard pattern, leveraging functions like findChessboardCorners, cornerSubPix, and drawChessboardCorners. Users can save calibration images and corresponding corner locations for calibration. The camera is then calibrated using the calibrateCamera function, producing intrinsic parameters like the camera matrix and distortion coefficients. The system estimates the camera's pose relative to the target in real-time, utilizing solvePnP. Subsequently, it projects 3D points onto the image plane and renders virtual objects accordingly, ensuring correct orientation as the camera moves. Lastly, robust feature detection techniques like Harris corners are explored for potential use in augmented reality applications. The project involves tasks like handling calibration images, calculating reprojection errors, and visualizing camera pose and virtual objects.

2) Image descriptions for project tasks:

Task 1 - Detect and Extract Target Corners:



Chessboard corner detected and output showing the no. of corners and coordinates of the first corner

The detectAndExtractCorners function takes a frame as input, converts it to grayscale, and attempts to find the corners of a chessboard pattern with a size of 9x6 squares. If corners are found, they are refined for accuracy, drawn on the frame, and the number of corners found along with the coordinates of the first corner are printed to the console. This function is part of the camera calibration process where identifying corners is essential for determining intrinsic camera parameters.

I am using a print of the chessboard image as my target. The system has certain limitations:

1. **Target size relative to the camera** - When the target object is too far or too near the camera, the system is not able to detect the corners properly. The target object needs to be kept at a proper distance so that we can accurately extract the target corners.
2. **Changes in lighting conditions** - The detection process is affected by lighting conditions as the system will not detect corners even if it is at the correct distance in case there is a strong glare or very low lighting. The lighting conditions should be appropriately set to start the detection process.

Task 2 - Select Calibration Images:

```
Select input source:
1. Webcam
2. Video feed from phone
Enter your choice (1 or 2): 1
[ WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (935) open OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
Not enough calibration frames selected (need at least 5).
Not enough calibration frames selected (need at least 5).
Not enough calibration frames selected (need at least 5).
Not enough calibration frames selected (need at least 5).
```

Chessboard corner detected and output showing whether the minimum no. of images required for calibration have been saved

The project pipeline has been set up such that when the user presses the ‘q’ key, the program will be terminated. When the user presses the ‘s’ key, the chessboard image with corners detected and highlighted shall be saved in a folder whose location needs to be entered in the code as per the user’s system.

When ‘s’ is pressed, the saveCalibrationData function saves the detected corner locations and their corresponding 3D world points. First, it checks if corners were successfully detected. Then, it computes the 3D world points based on the corner indices and stores them along with the corner locations. Additionally, it captures the current frame containing the corners and saves it as a calibration image. This function facilitates the calibration process by preserving the necessary data for accurate camera calibration.

Task 3 - Calibrate the Camera:

```
Camera matrix before calibration:
[1, 0, 0;
 0, 1, 0;
 0, 0, 1]
Distortion coefficients before calibration:
[0, 0, 0, 0, 0]
Camera matrix after calibration:
[2186.129283781112, 0, 441.7195101973173;
 0, 2186.129283781112, 141.9335121596583;
 0, 0, 1]
Distortion coefficients after calibration:
[0.8173437203712093, -3.261528141762953, -0.08463074312531534, -0.02893438126907133, 17.59859696806003]
Reprojection error: 1.62421
```

Chessboard corner detected and output showing the camera calibration parameters and the reprojection error

The calibrateCamera function performs camera calibration using the provided calibration data. It first checks if a sufficient number of calibration frames (at least 5) have been collected.This happens after the user has continuously pressed s five times so save the calibration images. If so, it initializes necessary variables and performs the calibration process using OpenCV's calibrateCamera function. This process estimates the camera matrix and distortion coefficients based on the collected calibration data. The resulting camera matrix and distortion coefficients are printed to the console, and they are also saved to an XML file specified by the xmlFile parameter which is passed from the main function. The XML path address needs to be changed by the user in the code. Additionally, if there are not enough calibration frames, an error message is displayed indicating the minimum requirement for calibration. The reprojection error is calculated and printed out along with the calibration parameters once minimum 5 calibration images have been saved.

Camera Matrix:

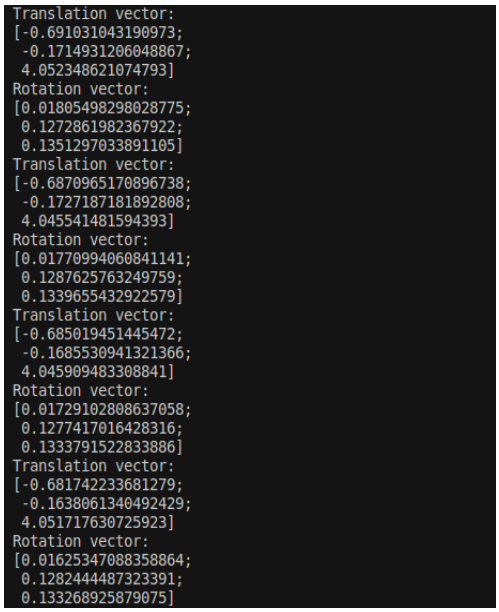
[2186.129283781112, 0, 441.7195101973173;
0, 2186.129283781112, 141.9335121596583;
0, 0, 1]

Distortion Coefficients:

[0.8173, -3.216, -0.0846, -0.02893, 17.5985]

Reprojection error/Error estimate: 1.62421

Task 4 - Calculate Current Position of the Camera:



Output showing the translation and rotational data

The code for this task utilizes several functions for camera calibration and pose estimation. Firstly, it employs cv::cvtColor() to convert the captured color image to grayscale for better corner detection. Then, it utilizes cv::cornerSubPix() to refine the detected corner positions with sub-pixel accuracy. Following this, it utilizes cv::drawChessboardCorners() to visualize the refined corners on the original image. Subsequently, it employs cv::solvePnP() to estimate the camera's pose by computing rotation and translation vectors. Finally, it prints the rotation and translation data for analysis and debugging purposes.

As I move the camera side to side, the translation values are changing a lot as compared to the rotation values which have minor changes. This is expected since we are moving the camera in translational motion, and not rotational. The rotational values might change slightly during this motion due to change in the angle relative to the target object.

Task 5 - Project Outside Corners or 3D Axes:

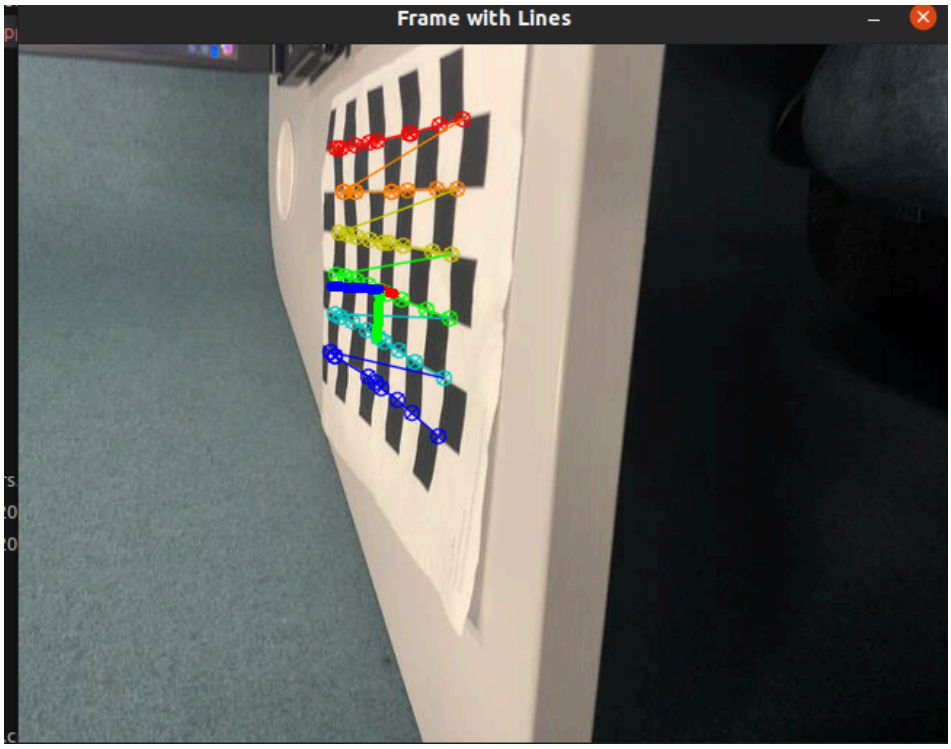
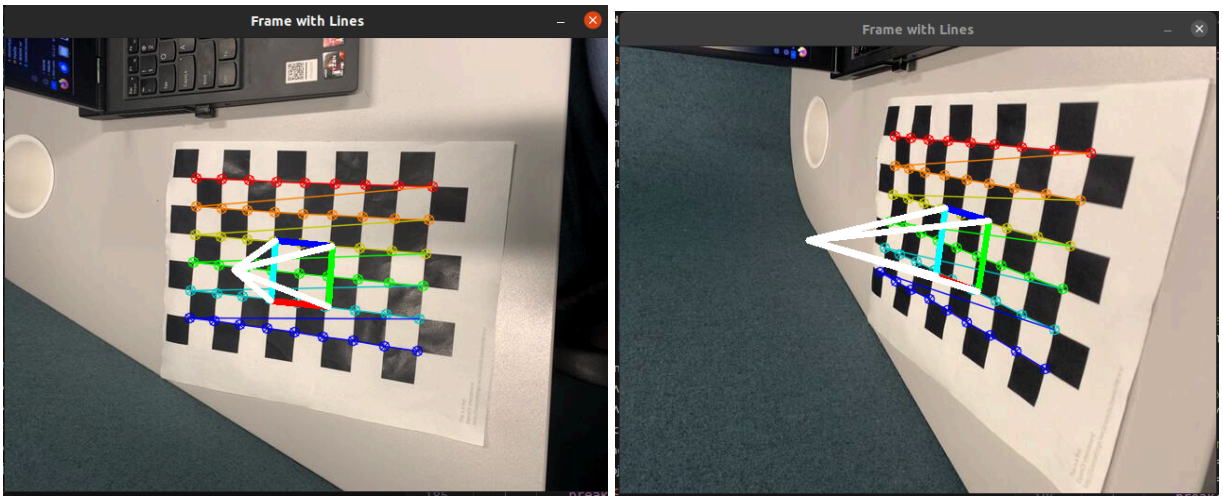


Image showing the 3-D axes projected on the target

When the user presses the ‘x’ key, this task will be implemented. The code segment for this task will draw the 3D axes on the checkerboard detected in the frame. It first defines the origin and endpoints of the X, Y, and Z axes in the 3D world coordinate system. Then, using the cv::projectPoints function, it projects these 3D points onto the 2D image plane based on the camera's pose (rotation and translation vectors) obtained from the solvePnP algorithm. The projected 2D points are stored in the imagePoints vector. Finally, it draws lines connecting the projected points to visualize the X-axis in red, Y-axis in green, and Z-axis in blue on the frame using the cv::line function. This helps visualize the orientation of the checkerboard relative to the camera.

Yes, the 3D axes (X, Y & Z) are projected in the right place as evident from the image. It is as defined in my code.

Task 6 - Create a Virtual Object:



Images showing the top view and side view of the pyramid projected on the target

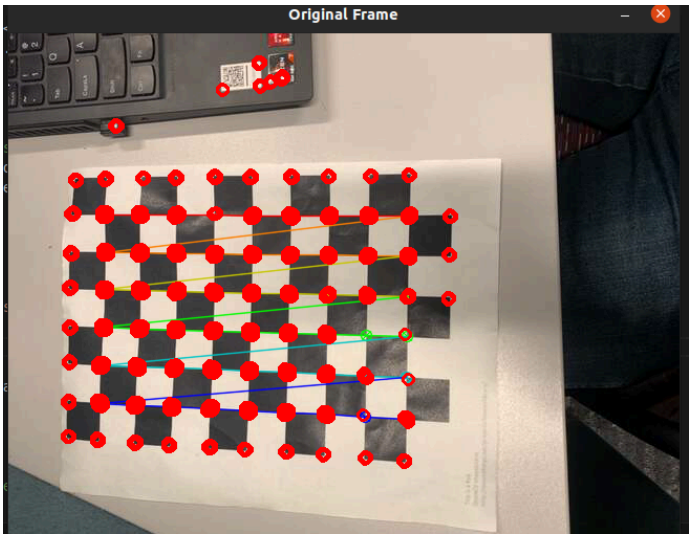
When the user presses the ‘p’ key, this task will be implemented. In this task, I defined a set of 3D points representing a larger pyramid-shaped virtual object. These points are adjusted to increase the size of the pyramid, with the apex set at a specified negative z-coordinate to position it above the base. Using the cv::projectPoints function, these 3D object points are projected onto the 2D image space using the provided rotation and translation vectors (rvec and tvec), along with the camera matrix and distortion coefficients. The resulting 2D points are stored in the vector imagePointsObj. Subsequently, lines are drawn between these projected points to visualize the pyramid on the frame, depicting its base and sides. The cv::line function is utilized to draw the lines, with different colors assigned to different sides of the pyramid.

The virtual object is a pyramid defined by five 3D points: four for the base arranged in a square and one for the apex above the base. The size of the pyramid is adjusted by setting its height, with the base drawn using blue, red, and green lines and the sides drawn using white lines on the image frame.

Demo Video Link:

https://drive.google.com/file/d/1gHle0tYBz8ALc4jjy4D0i3S_HfKWoQNS/view?usp=sharing

Task 7 - Detect Robust Features:



Images showing the Harris corners detected on the pattern

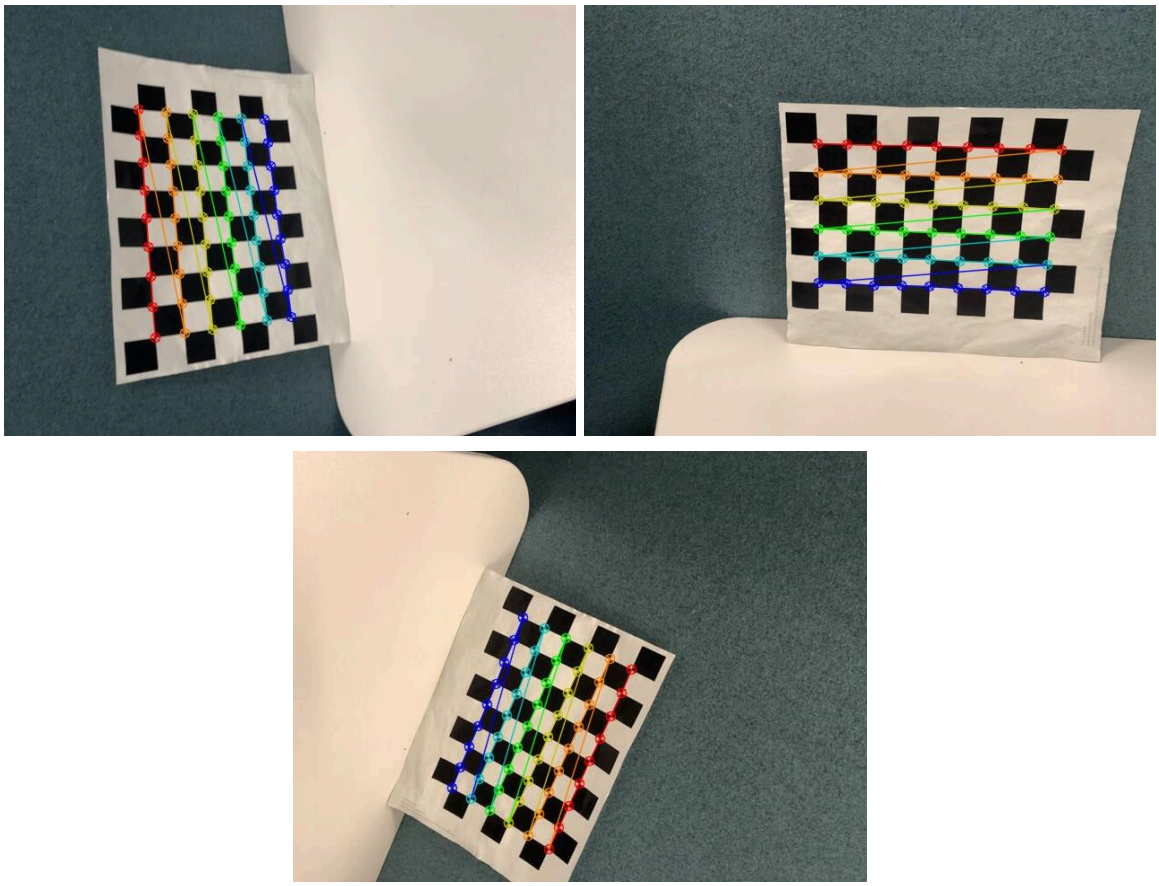
When the user presses the 'h' key, this task will be implemented in the original video frame. In this task, I have performed Harris corner detection on a grayscale image (color to greyscale conversion done in the code). It first applies the Harris corner detection algorithm (cv::cornerHarris) with specified parameters such as block size, aperture size, and a constant k. The resulting corner map is normalized and thresholded to obtain a binary corner map. Corners are then extracted from the binary map and drawn onto the original frame as red circles (cv::circle). The detected corners are displayed in real time.

The Harris corner detection algorithm identifies key feature points in the image. By using these points as reference markers, AR systems can overlay virtual objects onto the real-world scene with accurate alignment and perspective. This involves matching the detected corners with corresponding points in virtual object models and calculating transformations to ensure seamless integration of virtual content into the live camera feed.

Demo Video Link:

https://drive.google.com/file/d/1gHle0tYBz8ALc4jjy4D0i3S_HfKWoQNS/view?usp=sharing

Extension-1: Test out different cameras



Calibration images taken from a mobile phone video stream

```
Select input source:
1. Webcam
2. Video feed from phone
Enter your choice (1 or 2): 2
Not enough calibration frames selected (need at least 5).
Not enough calibration frames selected (need at least 5).
Not enough calibration frames selected (need at least 5).
Not enough calibration frames selected (need at least 5).
Camera matrix before calibration:
[1, 0, 0;
 0, 1, 0;
 0, 0, 1]
Distortion coefficients before calibration:
[0, 0, 0, 0, 0]
Camera matrix after calibration:
[512.1899980670953, 0, 346.7434759253816;
 0, 512.1899980670953, 227.5549669447771;
 0, 0, 1]
Distortion coefficients after calibration:
[0.265934670549313, -1.184249248000743, -0.005739120476314529, 0.01966679763070873, 1.420024883793275]
Reprojection error: 0.765358
```

output showing the camera calibration parameters and the reprojection error

This extension has been implemented in the Task-1 to 3 program. The code will ask the user to select between webcam and mobile phone camera stream inputs for camera calibration. When the user presses 1, the webcam is selected. When 2 is pressed, the video feed from the phone is selected. Both of these options can be implemented for comparison of the quality of calibration results.

Camera matrix:

[512.1899980670953, 0, 346.7434759253816;
0, 512.1899980670953, 227.5549669447771;
0, 0, 1]

Distortion Coefficients:

[0.265934670549313, -1.184249248000743, -0.005739120476314529, 0.01966679763070873, 1.420024883793275]

Reprojection error/Error estimate: 0.765358

These are the results of the Webcam calibration from Task 3:

Camera Matrix:

[2186.129283781112, 0, 441.7195101973173;
0, 2186.129283781112, 141.9335121596583;
0, 0, 1]

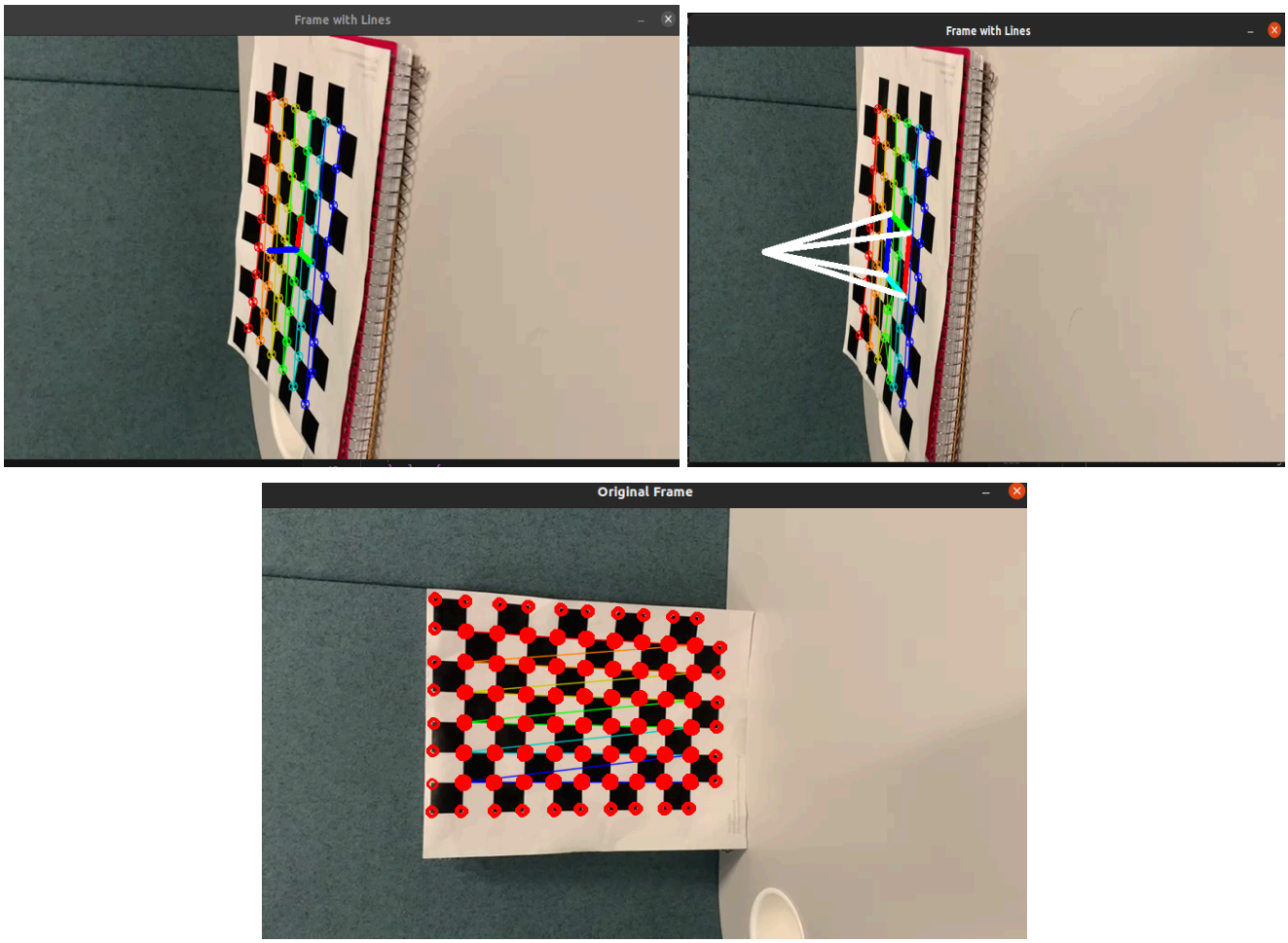
Distortion Coefficients:

[0.8173, -3.216, -0.0846, -0.02893, 17.5985]

Reprojection error/Error estimate: 1.62421

The calibration results demonstrate a stark contrast in quality between the webcam and mobile phone. The mobile phone exhibits superior calibration, evident from its lower reprojection error of 0.765358 compared to the webcam's 1.62421. Additionally, the camera matrix for the mobile phone shows more reasonable focal lengths (512.19 pixels) and principal points (346.74, 227.55 pixels), whereas the webcam's values are notably higher (2186.13 pixels for focal length, principal point near 441.72, 141.93 pixels). Furthermore, the distortion coefficients for the mobile phone are more balanced and within reasonable ranges, whereas the webcam's coefficients display higher magnitudes, indicating potentially greater distortion. These differences suggest better image quality and accuracy for the mobile phone camera calibration.

Extension-2: Insert virtual objects in a pre-recorded video:



Images showing the projection of 3-D axes, pyramid and Harris corner detection on a pre-recorded video

This extension has been implemented in the Task-4 to 7 program. The code will ask the user to select between a mobile phone and a pre-recorded video stream. When the user presses 1, the mobile phone is selected. When 2 is pressed, the pre-recorded video is selected as video stream input. The user can then press x to project 3D axes, p to insert a pyramid or press h for Harris corner detection on the pre-recorded video. From the screenshots, we can see that the program works perfectly well on a pre-recorded video sequence as well.

,

3) Short reflection on learnings:

During this project, I learned a lot about how to build an augmented reality system from scratch. I learned a lot about camera calibration by implementing the code for target corner extraction and then using it further for the calculation of camera calibration parameters like camera matrix and distortion coefficients. By reading these parameters from an XML file, I was able to determine the pose of the target relative to the camera on a live video feed. I then learned how to project the 3D axes and a virtual object on our target checkerboard image using the calibration parameters and the estimated pose. I also learned how to implement the Harris Corner Feature Detection algorithm by visualizing these features on a video feed through experimentation with different thresholds. Overall, the project gave me exposure to a wide variety of tasks related to an augmented reality system and integrating them to create a single product.

4) Acknowledgement:

We have referred to the following sources to complete this project:

1. https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html
2. https://www.youtube.com/watch?v=E5kHUs4npX4&list=PLkmvobsnE0GHMmTF7GTzJnCISue1L9fJn&index=14
3. Computer Vision by Linda Shapiro and George Stockman
4. https://docs.opencv.org/4.x/d4/d7d/tutorial_harris_detector.html
5. www.google.com