Project 4: Calibration and Augmented Reality

Hardik Devrangadi devrangadi.h@northeastern.edu
Thejaswini Goriparthi goriparthi.t@northeastern.edu

Camera Calibration and Augmented Reality

This project aims to perform Camera Calibration and use the intrinsic parameters to process the image sequence output from a webcam, to isolate a known pattern "the chessboard" placed in view. The points are projected as 3D points so that they can be used to place a 3D object or wireframe that tracks with movement (translation and rotation). Robust features are detected (Harris Corners) and analysis of the detected features is done.
The working demo of the project can be viewed here.

Tasks

Task 1: Detect and Extract Chessboard Corners

This program extracts and displays the corners of a chessboard pattern in a certain video frame. The frame, the size of the pattern, and a vector to store the extracted corners are sent to the extractChessboardCorners function. It extracts the corners using the findChessboardCorners function and then uses the cornerSubPix method to adjust the placements. It returns true if the corners are correctly extracted and false otherwise.
Using the current frame and the corners vector, the extractChessboardCorners method is called in the main body of the code. If the method returns true, a duplicate of the frame has the corners painted using the drawChessboardCorners function (in this case, frame2 is the clone of the original frame).

Task 2: Select Calibration Images

This code defines a function generateChessboard3DPoints that takes a cv::Size object patternSize as input and returns a vector of 3D points that represent the corners of a chessboard. The function iterates over the rows and columns of the chessboard and creates a cv::Vec3f object for each corner, with the first and second elements representing the x and y coordinates, and the third element set to zero.
The initial width and height of the patternSize object are 8 and 6, respectively, to match the dimensions of an 8x6 square chessboard.
To create the 3D points for the chessboard, the generateChessboard3DPoints function is used in the main code. The corners vector and points3D vector are recorded to the imagePoints and objectPoints vectors, respectively, along with the relevant frame in calibration images, if the extractChessboardCorners method locates the chessboard corners in the current frame. If the chessboard corners cannot be located, the console prints a notice.

Figure 1: A calibration image highlighting the found chessboard corners

Task 3: Calibrate the Camera

This function calibrateFrame takes a single frame, a set of object points, and a set of image points, and outputs the camera matrix and distortion coefficients. It uses the calibrateCamera function provided by OpenCV to perform the calibration.

The camera matrix and distortion coefficients are initialized to default values, and then calibrateCamera is called with the provided inputs. rvecs and tvecs are output parameters, but they are not used in this function. If the number of image points is greater than 5, the function calibrateFrame is called with the inputs frame2, objectPoints, imagePoints, cameraMatrix, and distCoeffs. Finally, these camera intrinsics are printed to the console. This data is also printed to a file "instrinsics.xml". This function is used in a loop that processes a series of frames to perform camera calibration.

```
Camera matrix:
[805.9168748889032, 0, 397.4870653859301;
 0, 983.3232543893786, 389.1646275439332;
 0, 0, 1]
Distortion coefficients:
[-0.1051136679508318;
 0.231790928152373;
 0.002534191914379447;
 -0.006988796305199146;
 -0.1447355082453501]
Reprojection error: 0.447073
```

Figure 2: A screenshot of the console showing the camera intrinsics

Task 4: Calculate Current Position of the Camera

The camera matrix and distortion coefficients are loaded from the previous function. The camera matrix describes the relationship between the 3D world coordinates and the 2D image coordinates, while the distortion coefficients describe lens distortion effects.

Then, the solvePnP function is used to estimate the pose of the camera. The solvePnP function takes in a set of 3D object points and their corresponding 2D image points, along with the camera matrix and distortion coefficients. It then estimates the rotation vector (rvec) and translation vector (tvec) that describe the pose of the camera with respect to the object coordinate system.

Finally, the rotation and translation vectors are printed out to the console.

```
Rotation vector: [-0.7573154212110358, 0.1288464625196279, -0.03937366708211625]
Translation vector: [-0.9936997429556345, -2.97059265755637, 14.93484993834929]
```

Figure 3: A screenshot of the console showing Rotation and Translation Vectors

Task 5: Project Outside Corners or 3D Axes

The code starts with a loop that iterates over the four outside points of the chessboard and projects them onto the image plane using camera calibration parameters (rotation vector, translation vector, camera matrix, and distortion coefficients) and passing them into the cv::projectPoints() function. The projected points are then drawn as white circles on the image using the cv::circle() function.

Next, six lines are drawn to connect the projected points and form a quadrilateral using the cv::line() function.

Figure 4: Projection of 3D points of the four corners of the chessboard on the image plane (shown by white dots)

Task 6: Create a Virtual Object

The function octahedronPlot is to plot an octahedron on the frame image using the octaPoints and objectPoints vectors, which contain the 3D coordinates of the octahedron's vertices. The function uses the solvePnP function from OpenCV to determine the rotation and translation vectors needed to project the 3D octahedron onto the 2D image plane. It then uses the projectPoints function to project the 3D points onto the 2D image plane and draws lines connecting the projected points to form the octahedron shape. The function draws circles at each of the projected points for indication.

In the main function, if the size of imagePoints (number of calibration images) is greater than 5, the current frame is calibrated and and the octahedronPlot function is executed for the obtained camera matrix, distCoeffs.

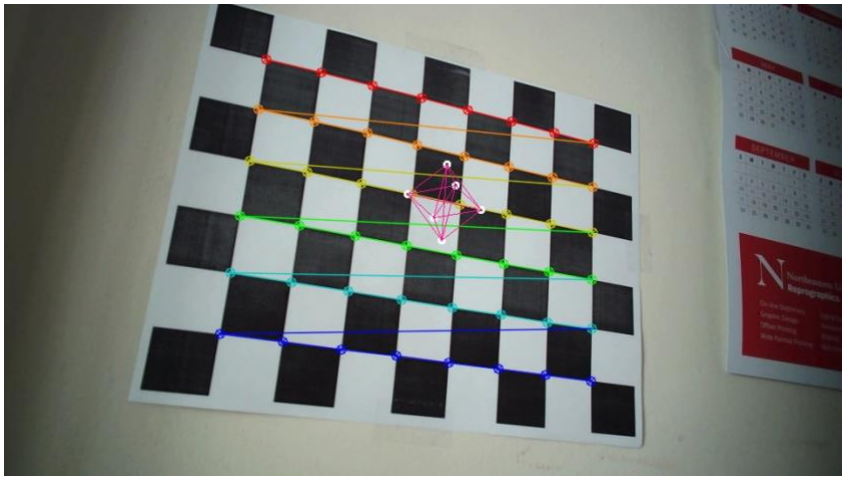

Figure 5: An Octahedron placed as a virtual object on top of the chessboard

Task 7: Detect Robust Features

This task detects Harris corners using a different pattern. The pattern chosen is a combination of QR codes arranged symmetrically. If the user continuously presses the key 'h', the program captures images from a camera or a video file and uses the Harris corner detection algorithm to detect corners. In order to reduce processing time, the captured video frame is first resized to half its original size. The cvtColor() function is then used to convert the frame to grayscale. With specified parameters, including block size, aperture size, and Harris parameter k, the Harris corner detection algorithm is applied to the grayscale frame using cornerHarris(). Corner detection output is then normalized and converted to grayscale using the normalize() and convertScaleAbs() functions, respectively. In the final step, circles are drawn around the detected corners using the circle() function, and the input image and the output of the corner detection are displayed separately using namedWindow() and imshow().

After the program detects the corners of a pattern, augmented reality can be added to the image. By placing a virtual object over the pattern, the object appears to be attached to it. The position and orientation of the virtual object can be determined by using the detected corner points as reference points.

To achieve this, the program needs to compute the transformation matrix between the detected corners and the corresponding points in the virtual object space. This transformation can be computed using the OpenCV function findHomography(), which estimates a perspective transformation between two planes given their corresponding points. The virtual object can then be rendered onto the image using the computed transformation matrix.

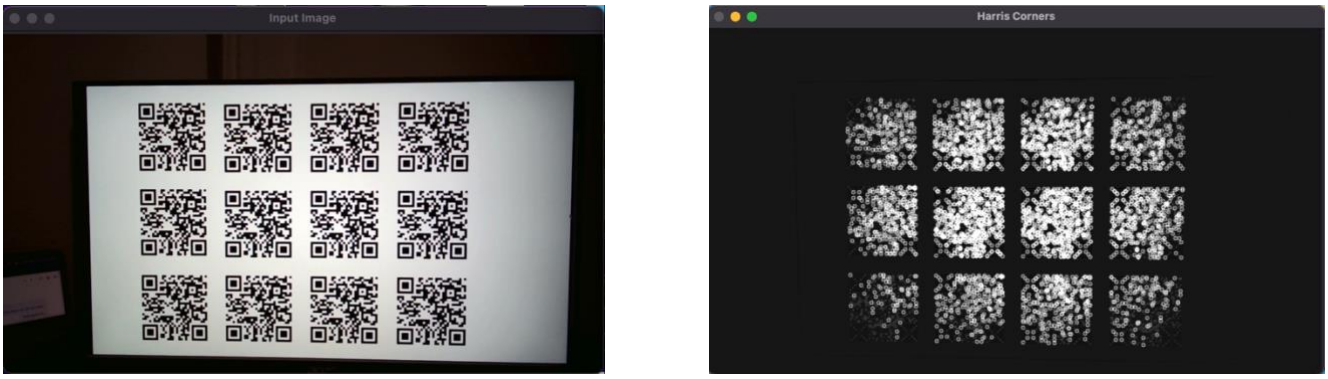

Figure 6: Harris Corners being detected on a QR Code pattern

Extensions

Three extensions have been added to this project. Three different cameras were used for camera calibration. A new image is placed on the target and blended with the target image. The SURF features are used to detect the corners in the image on a pattern of our choice.

Extension 1: Comparing Calibrations of Different Cameras

Three different cameras were considered for comparing the calibration results and quality. The first one is the iPhone camera with a camera resolution of 4032 x 3024. For a chessboard the calibration process was performed, and we obtained a reprojection error of 1.89754.

Then the camera of a MacBook was considered with a resolution of 1920 x 1080 and a reprojection error of 0.90212

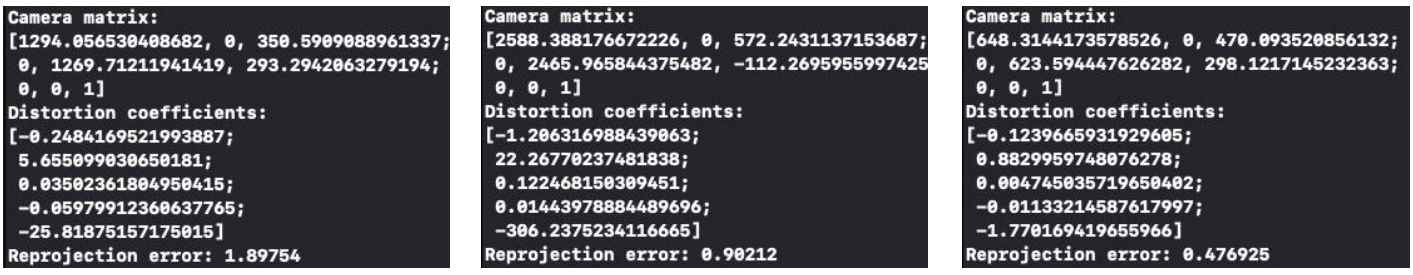

Figure 7: (a) iPhone Intrinsics (b) Laptop Webcam (c) External Webcam

Finally, an external webcam was used with a resolution of 1920 x 1080 and the reprojection error obtained was 0.476925.

Extension 2: SURF Features Detection

As an extension, the SURF features of a different pattern of QR codes was obtained. When the key "s" is pressed continuously, the SURF features are detected for the current frame in the video.

It first converts the input image to grayscale, then creates a SURF detector object with a specified threshold value. The detector is used to detect keypoints in the image, which are then stored in a vector.

The keypoints are then drawn onto the image, and the resulting image is displayed in a window.

Figure 8: SURF features being detected on a QR Code pattern

Extension 3: Target Modification

First, we calculate the minimum and maximum x and y coordinates of the chessboard corners and use them to define the destination (dstpoints) and source (srcpoints) points for perspective transformation.

The findHomography function calculates the homography matrix (h) to map the source points to the destination points. Then, the warpPerspective function warps the dst image using the homography matrix h and the size of the destination image (dstpic.size()), and stores the result in dstpic.

Finally, the code checks the value of the flag variable and, if it is true, replaces the non-black pixels of the input image (frame2) with the corresponding pixels in dstpic, effectively blending the input and warped images.

Figure 9: Image overlay on the Target

Project Learnings and Insights

This project is a computer vision application that extracts and displays the corners of a chessboard pattern in a video frame, performs camera calibration, estimates the pose of the camera, and plots a virtual object on the

image frame. The program also includes an extension to perform perspective transformation and blending of images. We also detect Harris corners for a different pattern, namely QR codes arranged symmetrically, and captures video frames from a camera for processing. We used various OpenCV libraries such as findChessboardCorners, cornerSubPix, calibrateCamera, solvePnP, projectPoints, and warpPerspective. Overall, this project demonstrates how computer vision techniques can be applied to extract information from video frames and perform various tasks such as camera calibration and image transformation. The various applications of these techniques in the Augmented Reality domain has also been explored and implemented giving an overall understanding.

Acknowledgements and Resources
We would like to acknowledge Professor Bruce Maxwell and all the Teaching Assistants for their valuable insights and support throughout the project.

- OpenCV Tutorials: https://docs.opencv.org/4.5.1/index.html
- Camera Calibration: https://learnopencv.com/camera-calibration-using-opencv/
- Transformation and Reprojection: https://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point/mathematics-computing-2d-coordinates-of-3d-points.html
- Calibration and 3D Object creation: https://github.com/Aniruddha-Tapas/Computer-Vision/blob/master/7.%20Camera%20Calibration%20and%203D%20Reconstruction.ipynb