Project 4: Calibration and Augmented Reality

Samara Holmes
Northeastern University

I. Introduction

This project aims to describe how to calibrate a camera using OpenCV and to display how to use the calibration to generate virtual objects within the scene. The result is a program that can detect a target and place an object in a scene relative to that target's movement. Additionally, features were added such as detecting multiple robust features, hiding the chessboard target, image distortion correction, and a point cloud using DepthAnything.

II. Methodology

For this project, we have the following tasks:

1) Detect and extract target corners
2) Save and select calibration images
3) Run camera calibration
4) Calculate the current position of the camera
5) Project outside corners or 3D axes
6) Create a virtual object
7) Detect robust features
8) Hide the target
9) Image distortion correction
10) DepthAnything point cloud?

To detect target corners I take the live video feed, greyscale it, then use cv::findChessboardCorners() and cv::drawChessboardCorners() to detect the corners. To save and select a calibration image, I run my detectCorners() function on the video feed, and I also run a function saveCorners() and saveCalibrationImage(), which saves the corner set and calibration image to a file and a folder, respectively. Once I have saved at least 5 calibration images, I begin camera calibration.

To run the camera calibration, I make sure I've saved enough images. If I've saved enough images, I load the corner sets that I've saved into the corners.txt file. Since it is a 6x9 chessboard, there are 54 points per set. Using the cv::calibrateCamera() function, I can get the reprojection error, calibration matrix, and distortion coefficients.

To calculate the current position of the camera, I use cv::solvePNP() to get the rotation and translation vectors. To project the outside corners, I use the calibration matrix, distortion coefficients, and the rotation and translation vectors to project the corners of the target in the image frame (see Figure 5).

To create a virtual object, I use the same parameters but generate a 3D object on top of the target (see Figure 6). The object I generated is a square bipyramid (which is essentially two pyramids placed on top of each other, with the bases being the connecting face).

Instead of detecting one set of features, I did an extension and detected multiple features, including Harris corners, ORB features, and SIFT features. These can be seen in Figures 7, 8, and 9.

Harris corners are determined by looking at the intensity changes in small patches within an image. From there, based on the eigenvalues of the matrix, it can be determined whether the surface is an edge, a corner, or flat. ORB (Oriented FAST and Rotated BRIEF) is a feature detection and description algorithm that provides scale and rotation invariance along with computational efficiency. It relies on FAST (Features from Accelerated Segment Test) for keypoint detection and can be implemented using cv::ORB(). SIFT (Scale Invariant Feature Transform) is another feature detection and description algorithm that can be implemented using cv::SIFT(). It detects scale invariant keypoints and encodes gradient orientations around the keypoints, making it highly descriptive and robust but also computationally heavy.

As an extension, I hid the chessboard target. Once I detect the target and get the outside corners, I can create a bounding box relative to those corners and cover it. I also hid the target using an image. First, I save an image of the current state of the scene without the chessboard target. Then, once the target is detected, I can get the outside corners, remove that patch, and replace the patch with a patch from the saved image.

III. Results

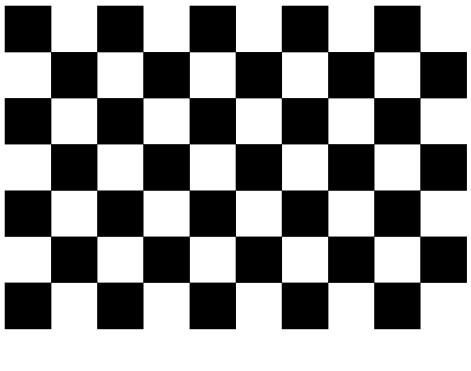

Figure 1. Chessboard target (9x6)

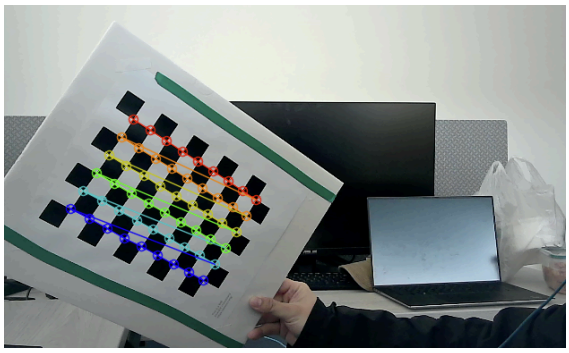

Figure 2. Sample calibration image

```
Camera Matrix:
[765.8573857576605, 0, 262.4036048520036;
 0, 752.6667739363252, 225.3995050749719;
 0, 0, 1]
```

Figure 3. Calibration matrix

```
error reported by cv::calibrateCamera: 0.752389
```

Figure 4. Reprojection error

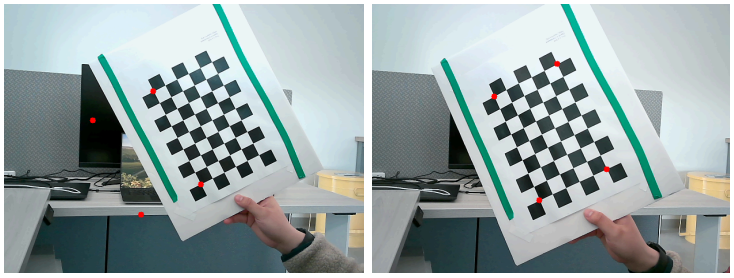

Figure 5. Re-projected points shown on live feed

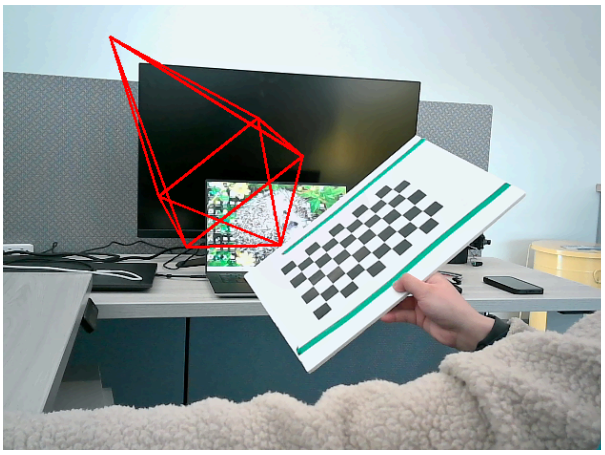

Figure 6. Virtual object projected above target

Figure 7. Harris corner detection

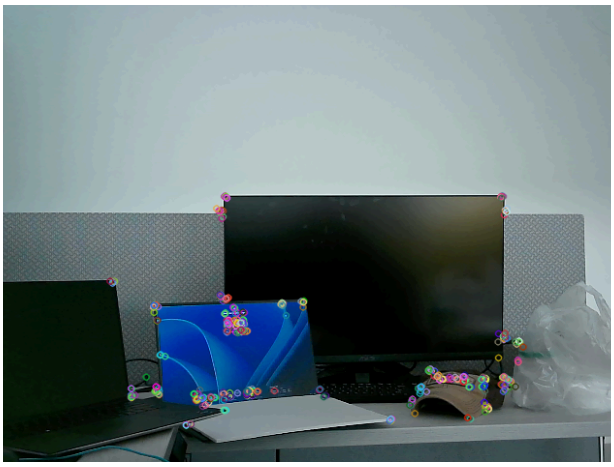

Figure 8. ORB feature detection

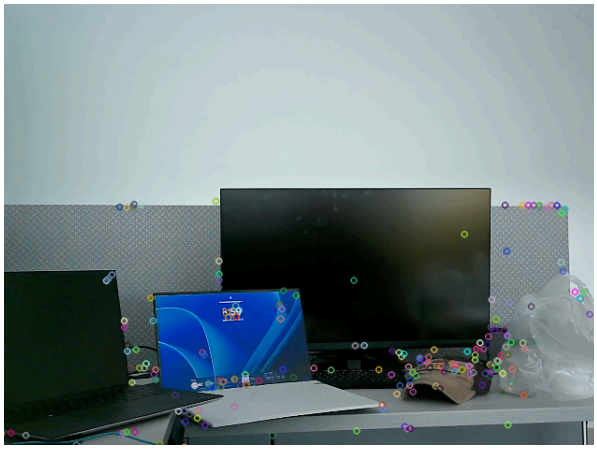

Figure 9. SIFT feature detection

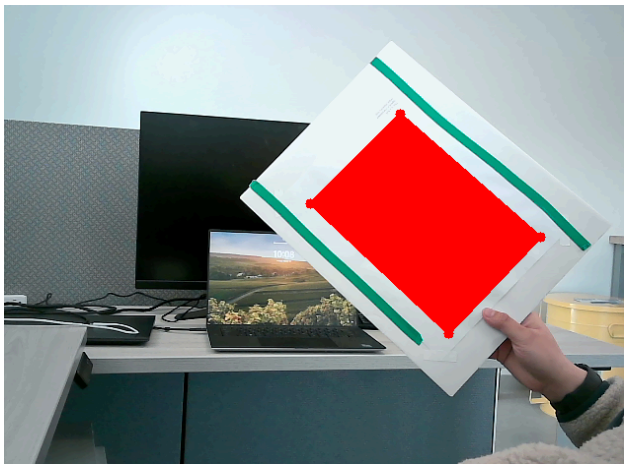

Figure 10. Hide the target

Figure 11. Hide the target using another image

Figure 12. Distortion correction

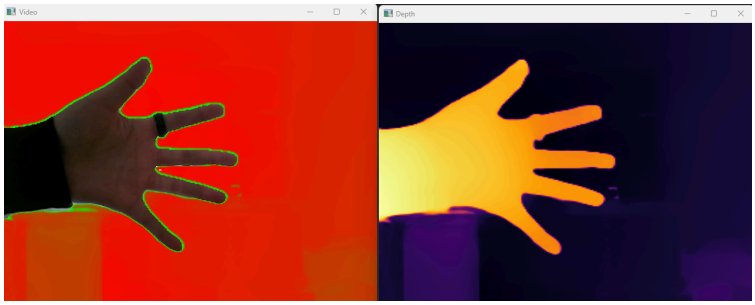

Figure 13. DepthAnything thresholding

IV. Discussion

Figure 1 shows the corners detected on my chosen chessboard target. One challenge of using a chessboard as a target is that there must be no deformation of the chessboard when calibrating. To mitigate errors, I placed the chessboard on a posterboard (see Figure 2). Another challenge I encountered was reflections and poor lighting, making it difficult to discern the corners.

With the calibration parameters (see Figures 3 and 4), I was able to estimate the camera's pose relative to the target. Translation X decreases as I move right to left in the image frame. Translation Y decreases as I move the target up to down in the image frame. Translation Z increases as I move the target further from the camera. This is expected behavior when working in the camera coordinate system. Figure 4 shows that my reprojection error is less than a pixel, which is sufficient to continue.

In Figure 5, I project the 3D points corresponding to at least four corners of the target onto the image plane in real time as the target or camera moves around. My initial attempt is shown on the left, where the points are shown to be going in the opposite direction. To solve this, I needed to negate two of the values, which resulted in the right image.

Figure 6 shows my virtual object hovering over the target. I created a square bipyramid object. Some thought needed to be put in to ensure that the second pyramid was facing the correct direction and on top of the first pyramid.

What I observed when doing the robust feature detection is that ORB (Oriented FAST and Rotated BRIEF) feature detection is fast. SIFT (Scale Invariant Feature Transform) feature detection detected a few more features than ORB but performed slower, as expected.

Hiding the target was an interesting feature to tackle. I started easy by just drawing the box within the projected points (see Figure 10). Then, I expanded upon this implementation by allowing the user to save background images. The user can save an image and then use a patch from that to be the replacement patch when hiding the target. This attempts to make the target invisible. Figure 11 shows my first attempt on the left, where I took the saved background image and sized it into the box. The image on the right is the proper implementation, where I took from the same patch on the saved background image.

Lastly, I wanted to generate a point cloud from DepthAnything, however, I was unable to do so. Instead, I was able to run DepthAnything (with appropriate reductions and upscaling for efficiency), and then, from the resulting depth map, threshold out things that seemed to be further away from the camera. See Figure 13.

V. Reflection

Developing a camera calibration system allowed me to learn more about going from a 3D to an image space. The system identifies a target (in this case, a chessboard), saves the corners to a file, and obtains calibration features that are later used in camera pose estimation. Modeling a virtual object and points relative to the targets' position was the main objective. The largest challenge I faced was in saving the corners to a file and storing them. This was because I needed to not only save the corners for each calibration image but also generate the 3D points associated with them, and store them within the application.

VI. Appendix

See the video demonstration here.

How to run the application:

1) Open the application
2) If images/corner sets are already saved, press '1' to get the calibration matrix and distortion coefficients
    a) If they aren't saved, press 's' to save a set, then press 'a' to reset the key selection, and repeat until a sufficient number of images/corner sets have been saved
3) Press '2' to start printing out rotation and translation matrices
4) Press '3' to project the outer corners
5) Press '4' to hide the target with a red box
6) Press '5' to display the virtual object above the target
7) Press '6' to hide the target using a pre-saved background image
    a) Press 'n' to save a background image
8) Press '7' to run depth anything
9) Press 'h' for Harris corner detection
10) Press 'o' for Orb feature detection
11) Press 'f' for SIFT feature detection

VII. References & Acknowledgments

https://docs.opencv.org/4.x/d4/d94/tutorial_camera_calibration.html
https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a
https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html
OpenCV: Camera calibration With OpenCV
https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html
https://stackoverflow.com/questions/22722772/how-to-use-sift-in-opencv