CS 5330 Project 4

Kevin Heleodoro

Northeastern University

March 21, 2024

Abstract

The focus of this assignment was to create a program that would teach the basics of camera calibration and augmented reality. It involved using a checker, ArUco or ChArUco board to calibrate the camera and project a 3D image onto the 2D surface of the board. The assignment wrapped up with finding other ways to detect features in a pattern and brainstorming how a program could use key features as a surface to project 3D images to.

1 Detect and Extract Target Corners

I began the project using an ArUco board as a target for extracting corners due to them being more stable and offering non-symmetric shapes which allows us to track the top left corner in any orientation. According to the OpenCV documentation, "Calibrating using ArUco is much more versatile than using traditional chessboard patterns, since it allows occlusions or partial views." [2] [5] [4].

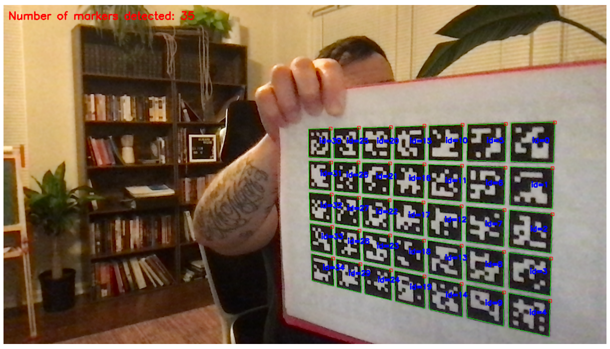


Figure 1: ArUco Marker detection

The ArUco board detection program was able to easily detect the ArUco markers whether they were in a board or scattered format. The issue I ran into was getting the calibration data saved using the ArUco markers. After much trial and error I decided to switch gears and used the Chessboard for this assignment. The program was able to easily recognize the Chessboard corners using OpenCV's built in methods.

2 Select Calibration Images

Using the built-in OpenCV functions I was able to save a series of Chessboard images to be used for the camera calibration. The images used for the calibration consisted of many variations of the Chessboard at different angles and rotations to the camera to provide a full set of features.

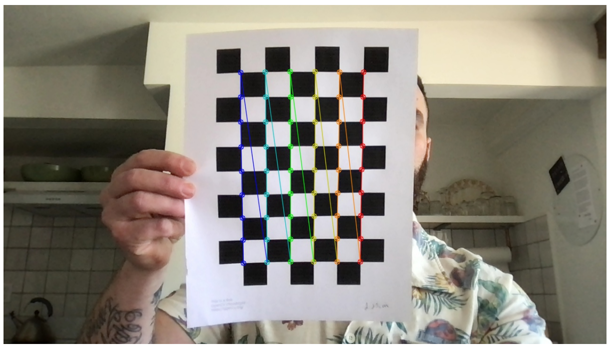


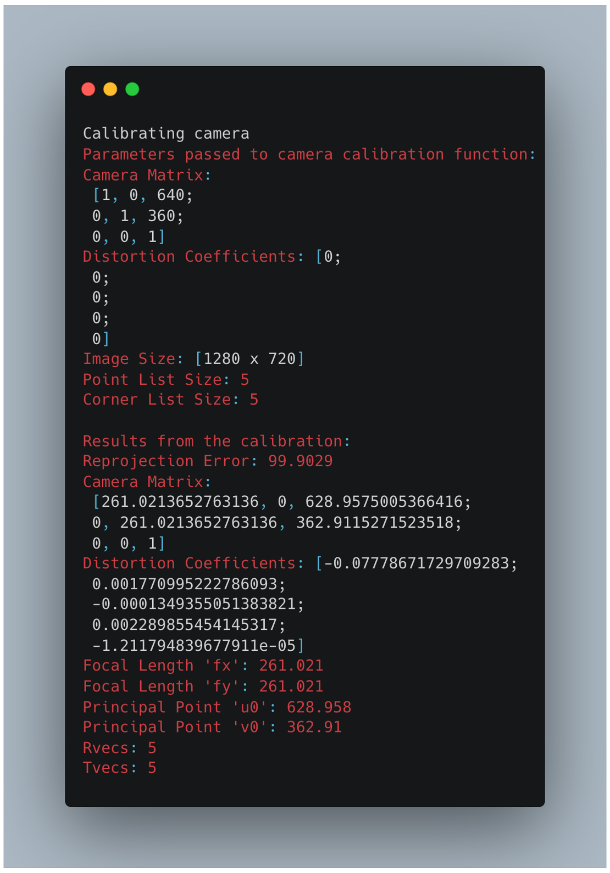
Figure 2: Chessboard corners

3 Calibrate Camera

As I mentioned in the first task, I struggled with using the ArUco boards and associated corners for camera calibration. I was collecting the corners incorrectly at first and not initializing the point set the right way. Originally, I was re-initializing the point set every time we were trying to save a calibration image. This was incorrect because it wasn't providing a consistent base to compare the target corners to. I also was pushing only the first corner instead of all corners detected into the corner set at first.

I was getting error messages during the camera calibration due to an assertion between the object points and the image points. I tried retaking the calibration images without flipping the board completely upside down and instead just taking screenshots at different angles and distances with the first corner maintaining its relative position when compared to the other corners.

I went through many different iterations of trying to calibrate the camera and the attached figure was the lowest error estimate I was able to process.

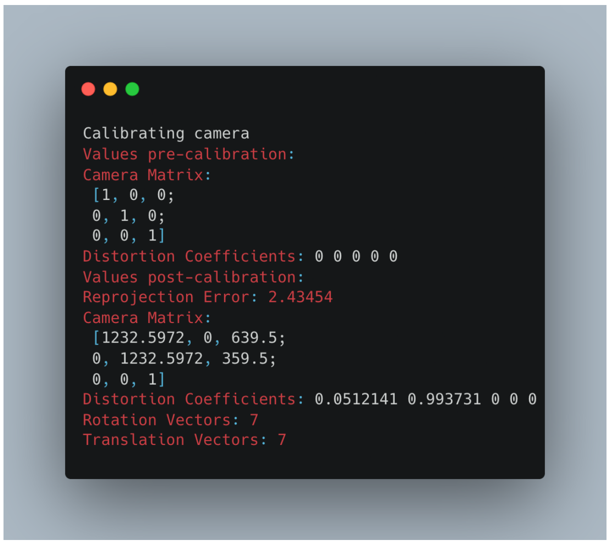


```
Calibrating camera
Parameters passed to camera calibration function:
Camera Matrix:
 [1, 0, 640;
 0, 1, 360;
 0, 0, 1]
Distortion Coefficients: [0;
 0;
 0;
 0;
 0]
Image Size: [1280 x 720]
Point List Size: 5
Corner List Size: 5

Results from the calibration:
Reprojection Error: 99.9029
Camera Matrix:
 [261.0213652763136, 0, 628.9575005366416;
 0, 261.0213652763136, 362.9115271523518;
 0, 0, 1]
Distortion Coefficients: [-0.07778671729709283;
 0.001770995222786093;
 -0.0001349355051383821;
 0.002289855454145317;
 -1.211794839677911e-05]
Focal Length 'fx': 261.021
Focal Length 'fy': 261.021
Principal Point 'u0': 628.958
Principal Point 'v0': 362.91
Rvecs: 5
Tvecs: 5
```

Figure 3: Results from the "best" calibration using the ArUco board

Once I switched to using the Chessboard to calibrate I was able to project much lower error rates. After some tweaking and experimentation I was able to get the error rate down to around 1 pixel. [3]



```
Calibrating camera
Values pre-calibration:
Camera Matrix:
 [1, 0, 0;
 0, 1, 0;
 0, 0, 1]
Distortion Coefficients: 0 0 0 0 0
Values post-calibration:
Reprojection Error: 2.43454
Camera Matrix:
 [1232.5972, 0, 639.5;
 0, 1232.5972, 359.5;
 0, 0, 1]
Distortion Coefficients: 0.0512141 0.993731 0 0 0
Rotation Vectors: 7
Translation Vectors: 7
```

Figure 4: Results for the first calibration using the Chessboard

4 Calculate Current Position of the Camera

Using the $solvePNP()$ function provided by OpenCV I was able to get the translation and rotation vectors in real time for the Chessboard. As I moved the Chessboard around I could see the respective values moving in different directions. For example, rotating the chessboard to the left would cause the rotation vector value to decrease into the negative. Conversely, a rotation to the right would result in the vector increasing. The rotation vector values stayed within the -2 to 2 range.

The translation values directly corresponded into what one would picture as the X,Y values of the camera. Moving the Chessboard to my right would cause the translation value to decrease and the opposite would happen if I moved the Chessboard to my left. The values here corresponded with the relative frame size divided into negative and positive halfs.

5 Project Outside Corners

The projected points were displayed roughly where I expected them to. Depending on the angle of the Chessboard there was a tiny deviation from the exact corner. Nonetheless, the outside corners moved with the Chessboard and remained within their respective corners.

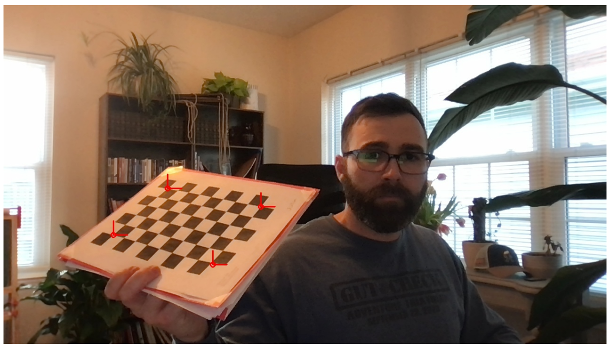


Figure 5: Project Corners

6 Create a Virtual Object

I decided to make an hourglass shape using two pyramids for the 3D object. It made me think about how to connect these points to each other and began to show me how complex these object could be. I plan on circling back to this project in the near future to keep expanding on this task.

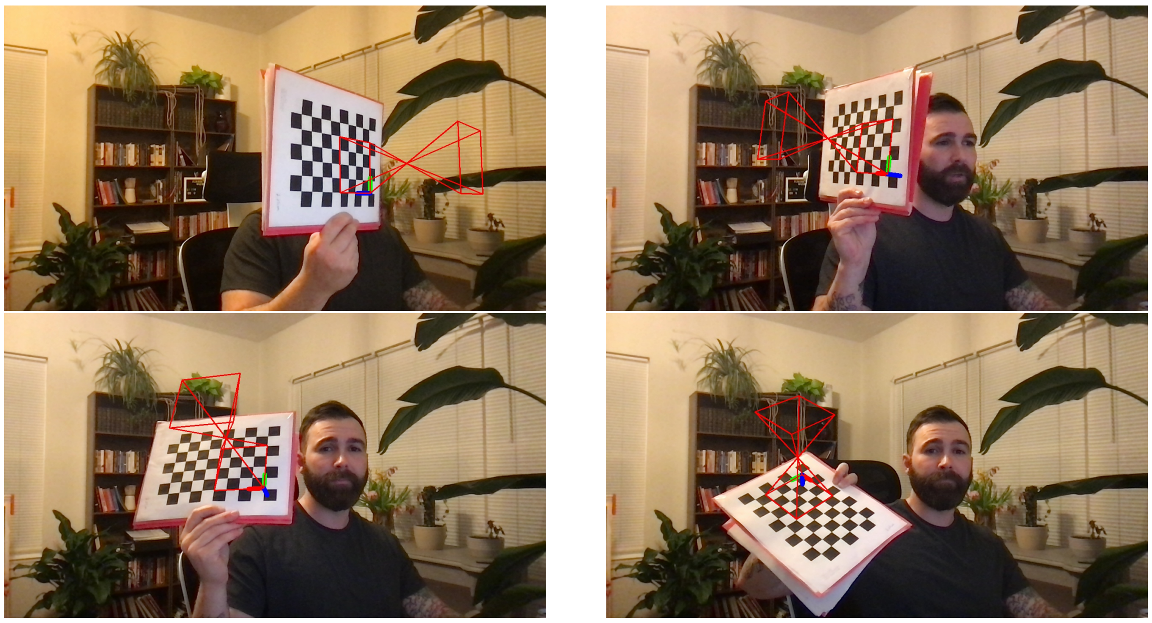
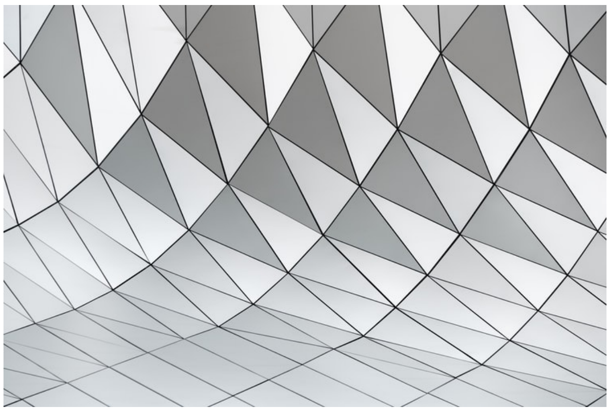


Figure 6: Virtual object at different angles

7 Detect Robust Features

I chose the Harris corners as the robust feature for this task. OpenCV provides some documentation on how to gather these features from an image [6]. I used a pattern I found online that I believed could be interesting due to its symmetry, number of corners, and color gradients. Similar to using the Chessboard corners or ArUco markers, Harris corners could also be used to identify a surface in which a 3D object can be built on. If we take the Harris corner return value we can follow the same logic as we did with the Chessboard corners and convert the data into a vector which can be further improved to provide a trackable area.



[1]

Figure 7: Diamond Pattern

For this task I need to perform further adjustments to get the corners to persist after being detected. Currently they appear to be blinking randomly throughout the pattern. Adjusting the threshold hasn't seemed to improve the retention of these corners.

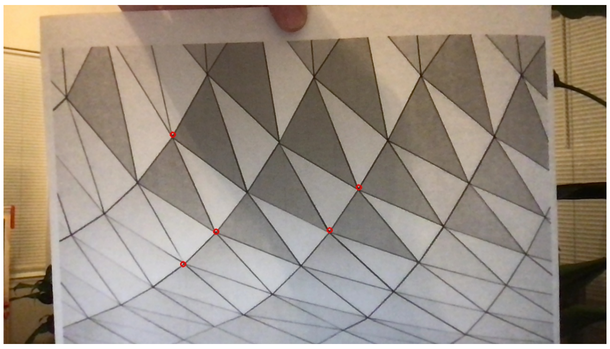


Figure 8: Inconsistent feature detection

8 Reflection

This assignment has piqued my interest in what applications can be built using augmented reality. I spent more time than I should have troubleshooting the ArUco board calibration which is why I was unable to work on any extensions for this assignment. Overall, the main takeaway for me was that if we can detect and extract a consistent set of features, then we are able to augment the surface of that object in a variety of ways.

References

[1] *Corner Detection Using Harris Corner in Python Programming.* https://getpython.wordpress.com/2019/07/10/corner-detection-using-harris-corner-in-python-programming/. Accessed: 2024-03-20. 2019.

[2] Sergio Garrido and Alexander Panov. *Detection of ArUco Boards.* https://docs.opencv.org/4.x/db/da9/tutorial_aruco_board_detection.html. OpenCV version 4.9.0-dev. 2024.

[3] Nicolai Nielsen. *Advanced Camera Calibration Technique with C++ and OpenCV: A Practical Guide.* https://www.youtube.com/watch?v=E5kHUs4npX4. Accessed: 2024-03-20. 2024.

[4] OpenCV Contributors. *OpenCV ArUco Detection Module Source.* https://github.com/opencv/opencv/tree/4.x/modules/objdetect/src/aruco. Accessed: 2024-03-20. 2024.

[5] OpenCV Contributors. *OpenCV Samples Repository.* https://github.com/opencv/opencv/tree/4.x/samples. Accessed: 2024-03-20. 2024.

[6] OpenCV Team. *Harris Corner Detector Tutorial.* https://docs.opencv.org/4.9.0/d4/d7d/tutorial_harris_detector.html. Accessed: 2024-03-20. 2024.