Samuel Lee

CS5330

Project 4

3/18/2024

Project Overview

The primary purpose of this project is to understand how to calibrate a camera and use the calibration data to project objects from 3D to the 2D image plane. This project first explores recognizing corners of a feature or target (for instance by using a chessboard). Once the target is found, we can save and use these images with the detected target to calibrate the camera. During calibration, it is important to have multiple different views of the target.

With enough images and corresponding corner sets, my program takes all the data to calibrate the camera. Calibration returns a refined camera matrix, distortion coefficients, and a reprojection error. Using this data, my program can recognize the same target in different orientations and poses in terms of rotation and translation. Now, with the camera matrix, distortion coefficients, rotation matrix, and translation matrix, the program is now able to project points of the 3D world onto the 2D image plane.

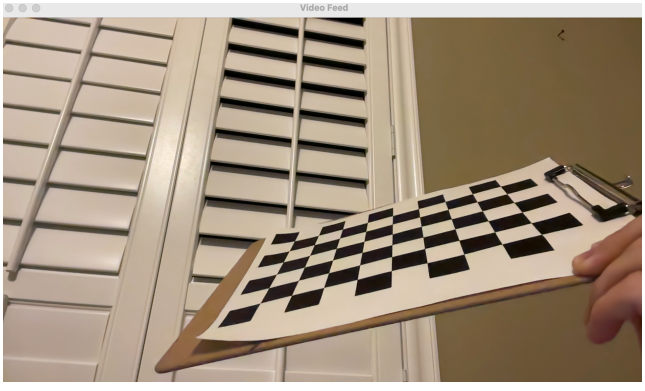
Overall, this project calibrates the camera with example images of the target and uses this calibration data to generate 3D objects in the 2D image plane.

Required Images

Question 1

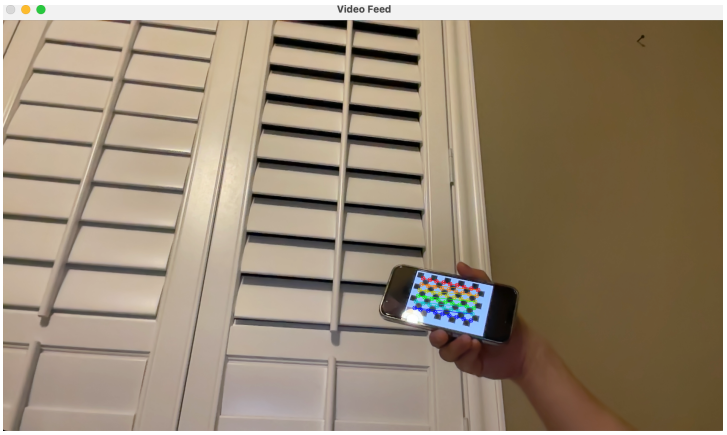
I am using the chessboard (9x6) target.

One limitation I am noticing is with extreme rotations to the chessboard. For instance, when I rotate the chessboard at a certain angle, the program is not recognizing the chessboard.



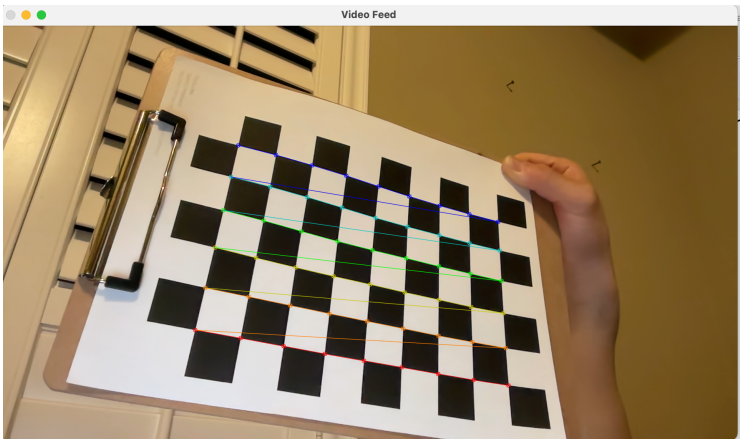
Also, I can see how extreme lighting can prevent the chessboard from being recognized as a target.

I am noticing that size of the chessboard does not really matter as shown below.



Question 2

My program stores the images used for calibration in the directory “calibration_directory” within the project folder.



Question 3

After calibration, my program immediately creates a .yml file and stores the camera matrix and distortion coefficients found in that run. For simplicity, the program overwrites any previous values in the .yml file.

```
Calibrated camera matrix:
[1388.782282182812, 0, 1001.286033264907;
 0, 1388.782282182812, 537.9990529769674;
 0, 0, 1]
Distortion coefficients:
[0.05953409756791424, 0.1012965212375465, -0.004506703757853071, 0.005636420433798742, -0.04856681799252097]
Reprojection error:
1.16769
```

Question 4

```
Rotation:
[1.606122382499663;
 -1.738797537967422;
 1.288589140029616]
Translation:
[-4.427724256405476;
 2.266154313217802;
 12.36094452411252]
Rotation:
[1.595844539565313;
 -1.749584521285176;
 1.304241321066588]
Translation:
[-4.387837903203883;
 2.284472529441086;
 12.33132071676902]
```

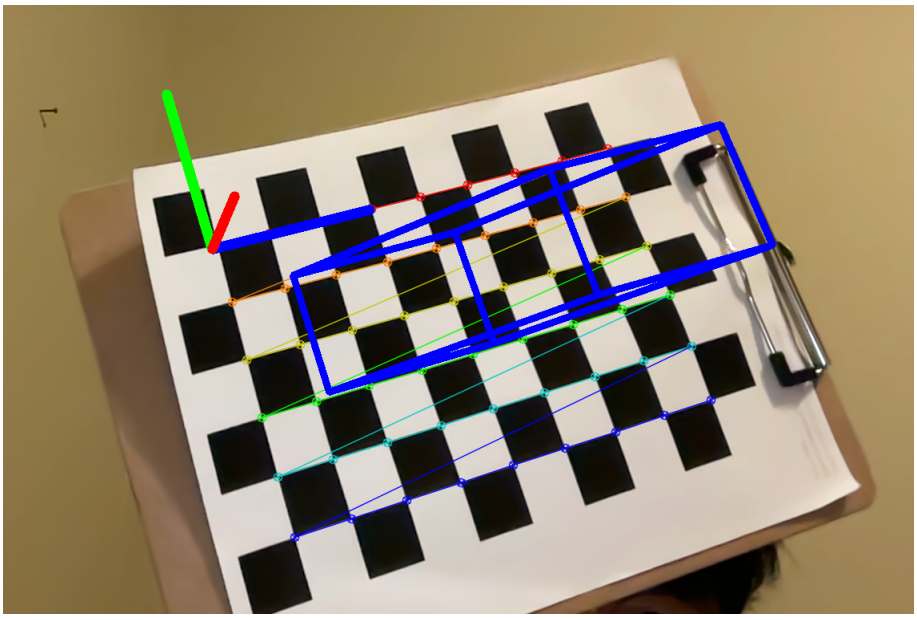
Based on how I set my XYZ coordinates up, I can see that as the target gets closer to the camera, the Z translation value decreases. Also, as I move to the right of the frame, the X value increases. As I move top to bottom, the Y value increases. These changes in the translation matrix makes sense because the direction I move in the corresponding direction changes the value of the corresponding axis vector of the matrix.

For the rotation matrix, when I rotate the chessboard with respect to the x-axis, the corresponding x-axis value for the rotation matrix changes. The same goes for the y-axis and z-axis rotations.

Question 5

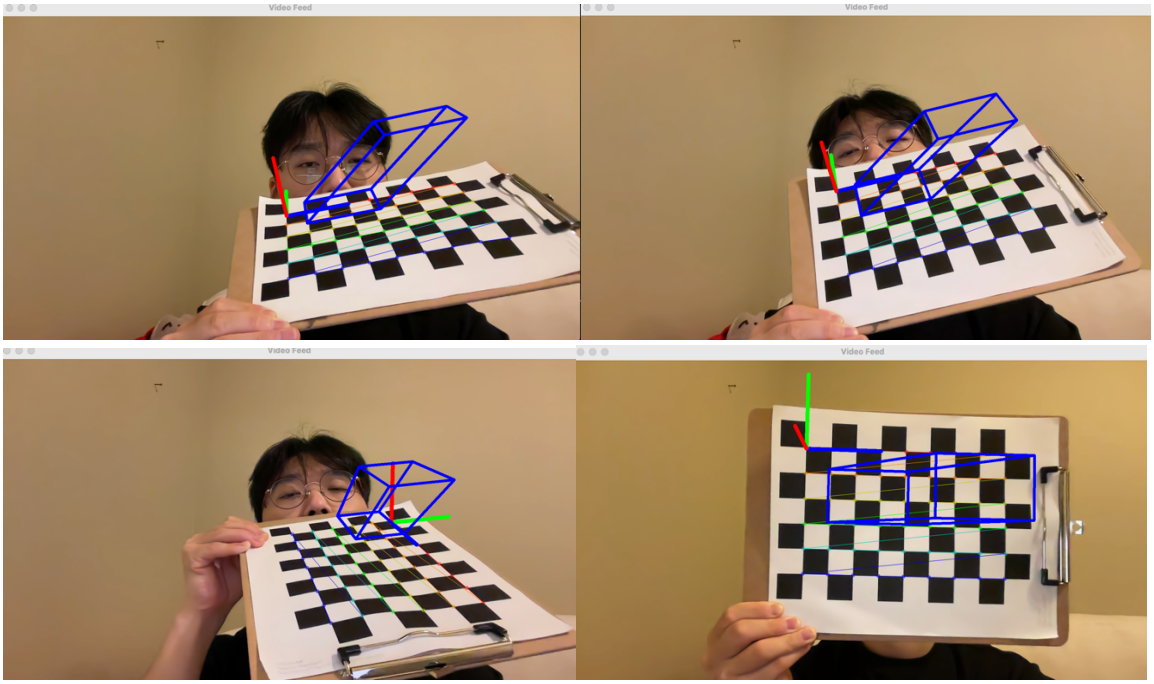
Yes, the reprojected points show up in the expected place. I projected 3D axes on the target attached to the origin. The origin starts 1 square away from the top side and the left side (due to the nature of how chessboard recognition works in OpenCV).

The reprojected points show up in the right places as expected.



Question 6

I created a slanted 3D parallelogram. The top and bottom faces are both rectangles, but their X-coordinates and Z-coordinates differ. From the sides, the shape you can see is not a rectangle but a parallelogram.



Question 7

The feature I picked was Harris corners. I implemented a program “detectFeatures.cpp” that uses Harris corners to detect features depending on certain threshold values and parameters.

I tested detecting features using Harris corners on a aluminum can’s brand logo as shown below.

I experimented with the sobel filter size, block size, aperture parameter, and the threshold for selecting a corner in my program to find the best filters to find the brand logo corners of the can.

Harris corners can detect feature points based on how significantly the intensity changes in all directions. Using these feature points found in a frame and the desired pattern, we can pre-calculate the Harris corners. With this pre-calculated Harris corner data set, we will then match this data set with the Harris corners found in new frames.

Once multiple matching images are found, we can calibrate the camera to get the reprojection error, calibration/camera matrix, translation and rotation matrices, and distortion coefficients. Then, using these data, we can project a 3D object into the 2D image plane.

Extensions

Multiple Chessboard Target Recognition

I made a separate cpp file called “augmentedRealityMultiple.cpp” that projects 3D object points to multiple detected targets in the video feed. To incorporate this, in “detectCorners.cpp,” I added a function called “getMultipleCorners,” which stores all found set of corners in a vector<vector<Point2f>>. Once a corner set is found, the program masks out the corner set and reiterates this until no additional corner sets are found. Using this vector of corner sets, we can project multiple 3D objects if multiple targets are found.

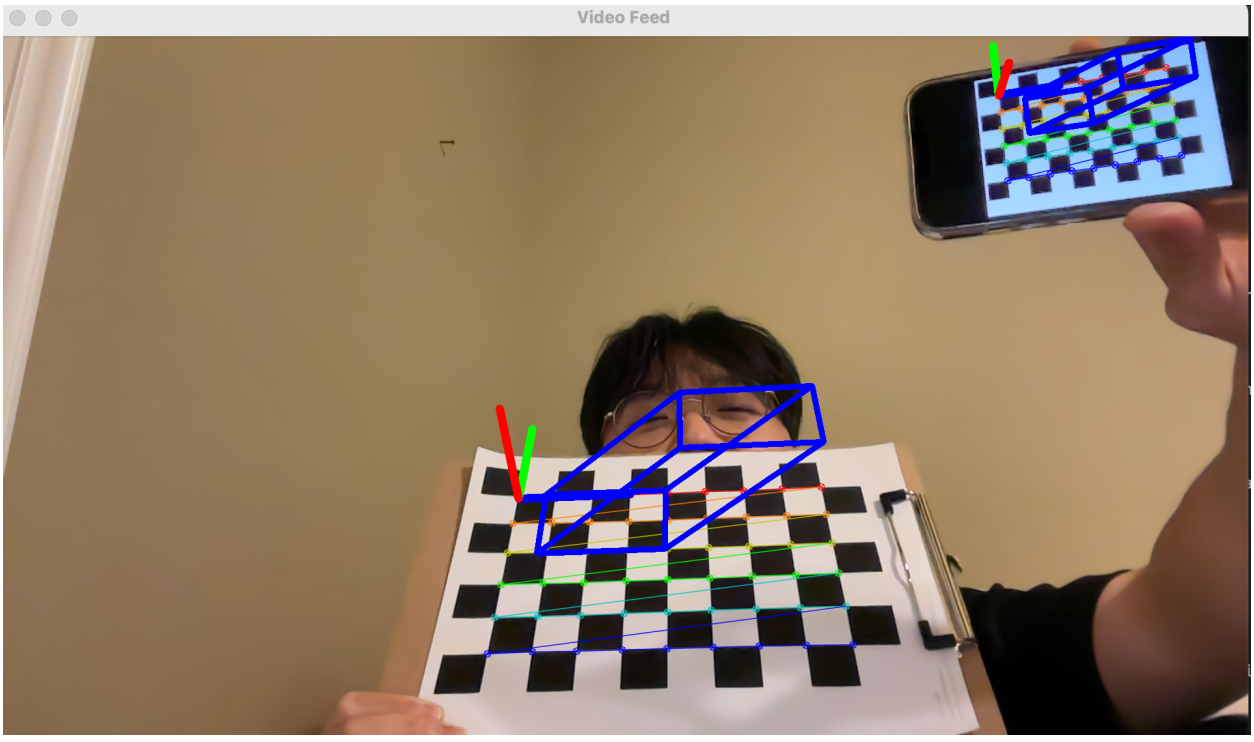
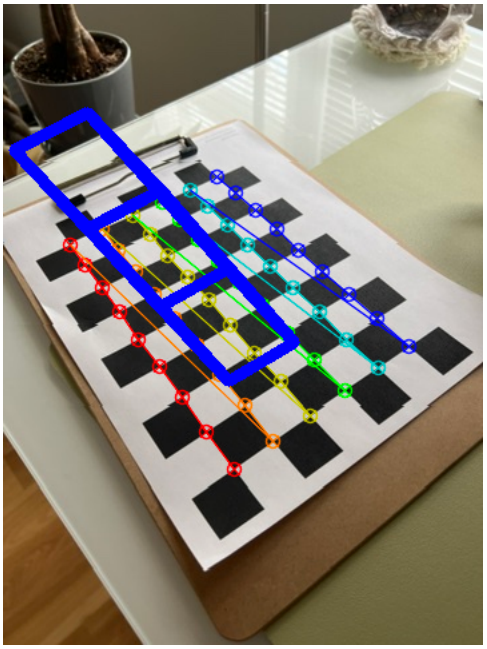


Image and Video Processing

Instead of just using a video feed to project 3D objects onto the current frame, I created a file called “generateAR.cpp” that allows a user to input a file path to a video or image to create an edited version of the video or image that shows the projected 3D object. Using the already existing calibration data in the .yml file, I was able to detect corners and the chessboard target as expected. This extension allowed me to explore VideoWriter in-depth to edit a video frame by frame. I also implemented a file and extension checker to see if the file is a valid video or image file and whether it was a valid file to open in the first place.

A limitation I ran into with video processing, however, is that it takes a long time due to the number of frames per second. Another limitation is that the chessboard is not detected for all the frames consistently. These two limitations, though, are opportunities for improvement in terms of efficiency as well as accuracy for future iterations.



This image has been generated by the program.

```
Calibrated camera matrix:
[1388.782282182812, 0, 1001.286033264907;
 0, 1388.782282182812, 537.9990529769674;
 0, 0, 1]
Distortion coefficients:
[0.05953409756791424, 0.1012965212375465, -0.004506703757853071, 0.005636420433798742, -0.04856681799252097]
Enter the path of the image or video you want to generate the image on (including the extension). Enter 'q' to quit.
../images/test1.jpeg
Image file read successfully.
Rotation:
[2.487279716655915;
 -0.7986045994424683;
 0.02251429175478647]
Translation:
[-36.80069144504269;
 -10.97875456372238;
 55.18353931023404]
Image with AR saved successfully to ../generated_images/test.png
```

Example run of the program shown above.

This is where you can find the processed video example:
https://drive.google.com/file/d/1x46YLrpiistNxpExy-5QiC7zT87VrYAg/view?usp=drive_link

Reflection

This project helped me understand projective geometry and how I can use OpenCV to test this. In class, the concepts I learned were not intuitive and hard to visualize. I understood the theory and studied it repeatedly, but I did not know the practical applications of the material I learned. This project really benefited me in my learning because it bridged the gap between theory and application. The step-by-step instructions of the project were especially helpful for me to understand the concepts by cross-referencing each section to the lecture.

Now, I know that we need a target to anchor from such as a chessboard or anything unique that doesn't vary much from one image to another. Using different views of the target, we can calibrate our camera to get our calibration matrix, reprojection error, and distortion coefficients, which we can use to get our rotation and translation matrices. Using these data points, we can then project 3D objects onto the 2D image plane.

Overall, I think this project was perhaps the most enjoyable and enlightening because I was able to concretely see the results of my program. Looking forward to the next projects, and I hope learning computer vision continues to be fun, exciting, and educational!

Acknowledgement

- Professor Maxwell's lectures – for filling knowledge gaps
- OpenCV documentation – to learn more about calibration and projection
- TAs – to ask questions to whenever stuck
- Stack Overflow – some posts on this website aligned with what I struggled with; used this resource to understand OpenCV functions better
- Past project code – re-used some code to save time