Project 4: Calibration and Augmented Reality

by Yixiang Xie for CS5330 CVPR

Project Description

This is the fourth project for the class CS5300 Pattern Recognition & Computer Vision. The project is about how to calibrate a camera and use the calibration to display virtual objects in a video stream. The result is a program that detects a target and place a virtual object in the scene relative to the target correctly, even when the camera or the target moves or orients.

The project tasks can be divided into three parts.

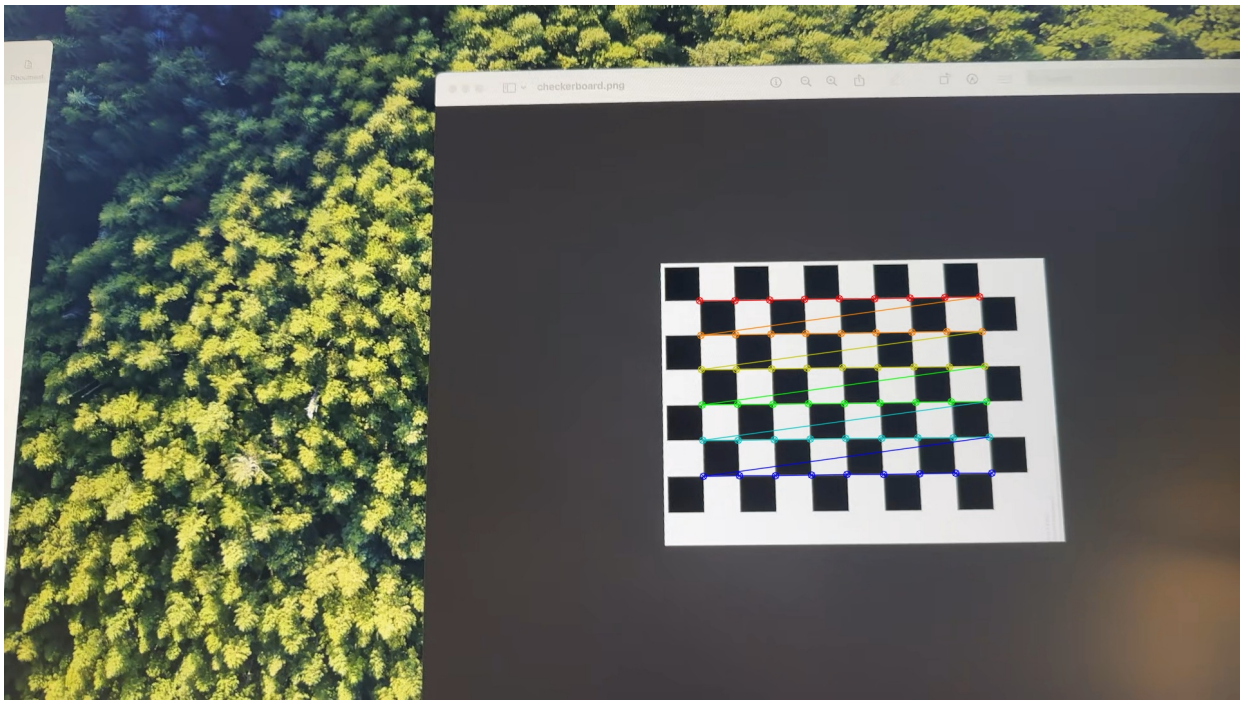
In part one, build a program that can detect a chessboard, extract the corners from chessboard and draw the corners on it. Users can select frames for calibration and run a calibration for the camera. Store the camera matrix and distortion coefficients into a file.

In part two, build a program that can detect a chessboard and calculate the board's pose using the corner locations and the camera calibrations. Create a virtual object and display that object relative to the chessboard in the video. The object should stay in the right position and orientation when camera moves around.

In part three, build a program that shows where the robust features (e.g. Harris corners, SIFT features or SURF features) are in the video stream.

Required Images

1. Calibration image with highlighted chessboard corners



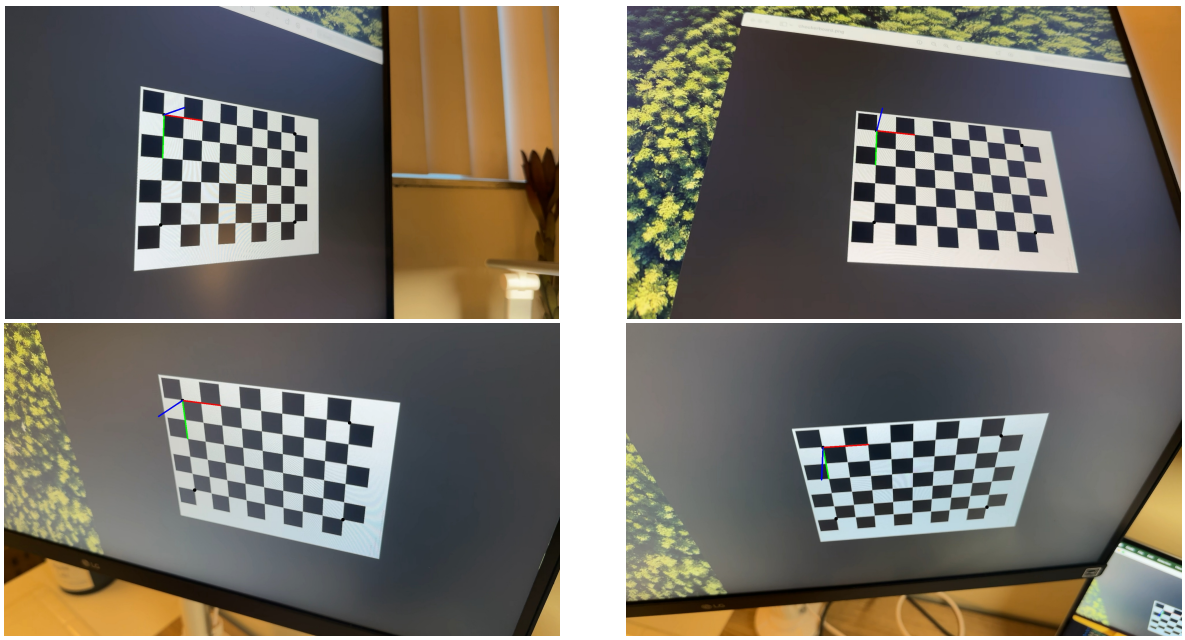
Calibration image with highlighted chessboard corners

2. Calibration re-projection error estimate

In my program, each time the user presses “s” with the chessboard in the frame, the program stores the corner coordinates and the 3D positions of the corners in world coordinates into two lists. It also saves the corresponding calibration image with highlighted chessboard corners into files. When the user has selected at least 5 calibration frames, the program automatically calibrates the camera. This produces a camera matrix and a list of distortion coefficients. The program stores these calibrations into a file. When user adds a new calibration image, the calibration is re-run and the results will overwrite the file.
Because I’m using cell phone cameras, I used the flag CALIB_FIX_ASPECT_RATIO to indicate that the pixels are assumed to be square. I also used the flag CALIB_RATIONAL_MODEL to take radial distortion into consideration as well.

# of frames used to calibrate	Re-projection error estimate
5	0.276528
6	0.267843
7	0.432080
8	0.266091
9	0.268188
10	0.261305

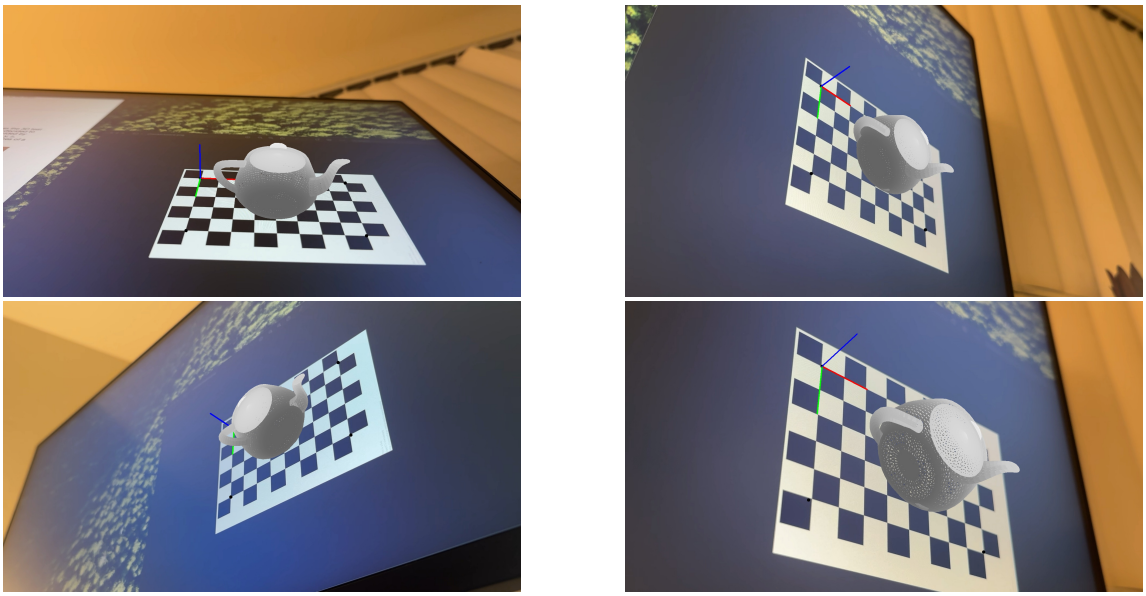
3. Project outside corners and 3D axes at the origin



Four outside corners of the chessboard displayed as black circles;
3D axes displayed as 3 color lines at the origin

4. Project a virtual object

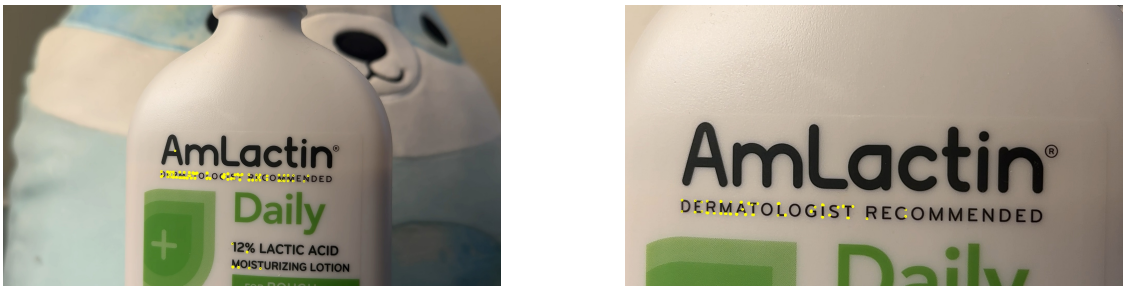
Previously when I was taking a computer graphics class, they used Utah teapot as the 3D test model. It is a standard reference object in the computer graphics community. So I decided to use the same model as the virtual object for task 6. The model data I used was provided by Stanford Computer Graphics Laboratory. For each line in the data file, it is either v, x, y, z, which is the x, y, z coordinates of a vertex, or f, v1,v2,v3, which is the three vertex indices of a face. I need to process the data a bit to allow better display in the scene.



Virtual object displayed on the chessboard

5. Detect robust features

I picked Harris corners to display for task 7. As shown in the results, there are many features being detected in the frame. In order to use them as the basis for putting augmented reality into the image, we need to map them to their 3D positions in the world coordinates. This requires the features to be stable and presenting a predictable order. To achieve this, we need to first pick an easy target for detection, such as squares. This is easier to figure out the corresponding 3D point coordinates. Then for each point, we need to use non-maxima suppression to filter out one feature for it. Then, we can get the target poses using solvePNP and we can get the 2D coordinates to display virtual objects into the image using projectPoints.



Harris corners on the target displayed as yellow circles

Extensions

1. Four robust features comparison

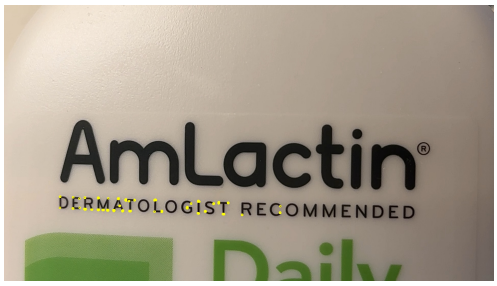
For task 7, besides Harris corners, I also implemented SIFT features, SURF features and Shi-Tomasi Corners. During the detection, Harris corners and SIFT features are pretty slow, while Shi-Tomasi Corners and SURF features are pretty fast. Harris corners and Shi-Tomasi Corners didn't yield too many noises, while SIFT features and SURF features detected a lot more features in the scene, some being noises.

Harris Corners displayed as yellow circles

Shi-Tomasi Corners displayed as blue circles

SIFT features displayed as green circles

SUFT features displayed as red circles



Harris Corners displayed as yellow circles

Shi-Tomasi Corners displayed as blue circles

SIFT features displayed as green circles

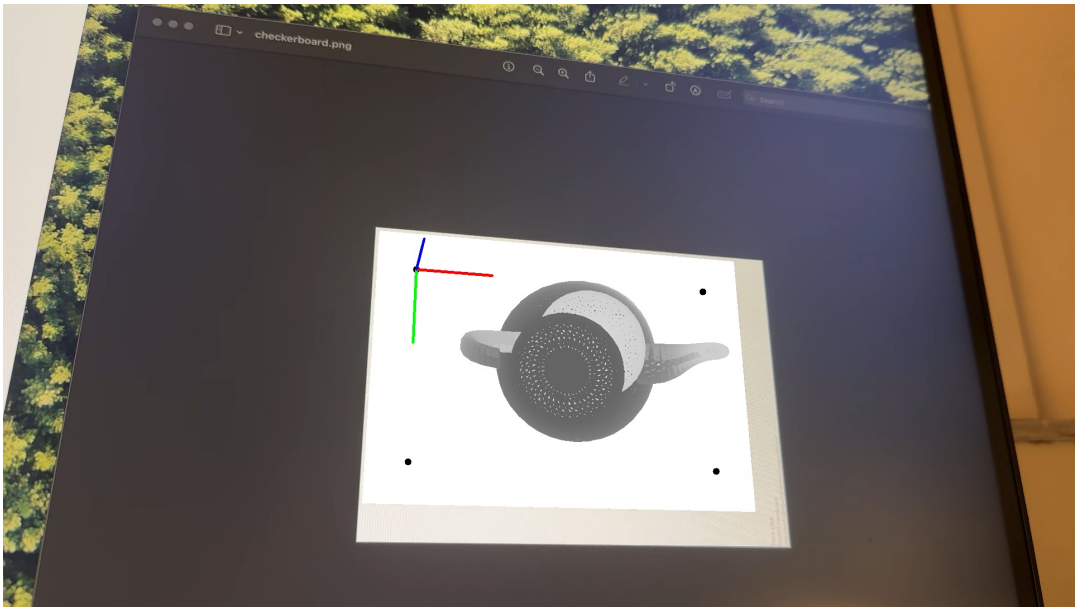
SUFT features displayed as red circles

2. Creative and complex virtual object and scene

As shown in task 6, I used Utah teapot as my virtual object choice. This object is especially helpful for visualization because it is asymmetrical in two dimensions. As shown in task 7, I try to detect features on characters. Some of the letters are good targets for detection, such as letter M and T.

3. Masking the target

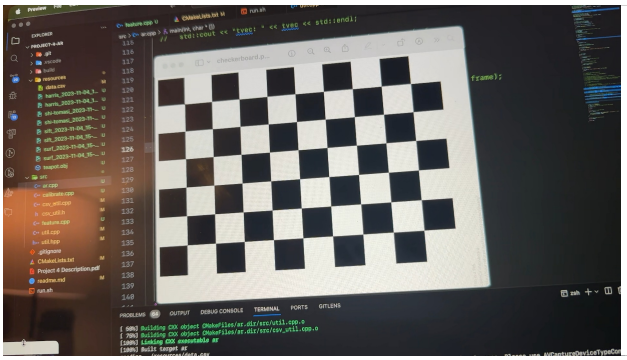
In order to make the target not look like the target itself, I tried putting a mask on the chessboard. The result is as the following picture.



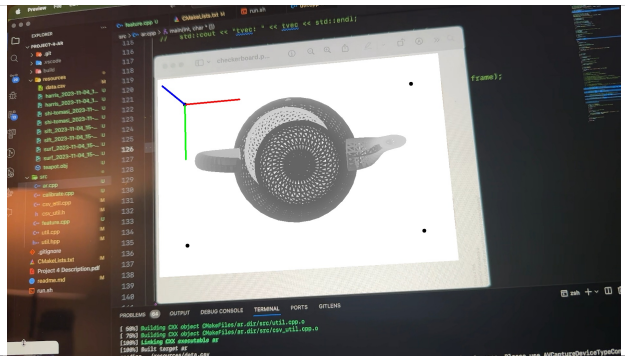
The chessboard is hidden underneath a white mask

4. Insert virtual object into static images of chessboard

For task 6, I also enable the program to receive a static image with the chessboard in it, and insert the teapot into the scene. The result is as following picture.



The static image

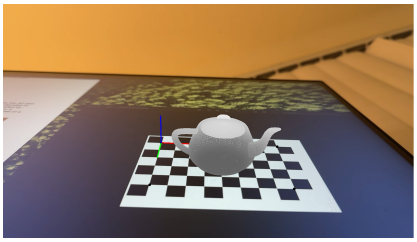


The static image with inserted virtual object

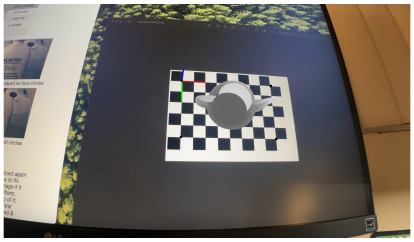
5. Compare different cameras for calibrations and results

I tested out three different cameras to compare their calibrations and quality of the results. The three cameras are iPhone rear facing camera 1X, iPhone rear facing camera 0.5X, MacBook Air camera. Comparing them with 10 frames used to calibrate. The result shows that iPhone rear facing camera 1X has the best re-projection error estimate. iPhone rear facing camera 0.5X has the worst, likely due to the radial distortion being too much.

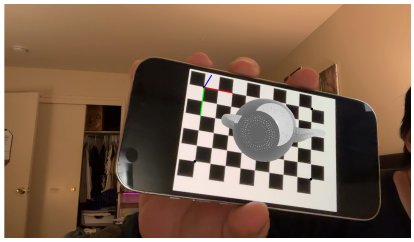
Device	Re-projection error estimate
iPhone rear facing camera 1X	0.261305
iPhone rear facing camera 0.5X	1.965138
MacBook Air camera	0.562737



iPhone rear facing camera 1X



iPhone rear facing camera 0.5X



MacBook Air camera

Reflections

In this project, I learned about how to calibrate a camera and get to develop an augmented reality application by using the calibration to display virtual objects in the video stream. This give me a better understand of camera matrix and distortion coefficients, and how to evaluate the calibration with re-projection error estimate. I also learn how to transform a coordinate from 3D world into 2D image. I also get to explore some feature detection and compare different ways of doing that, such as Harris corners, SIFT and SURF features.

Acknowledgments

Professor Bruce Maxwell, for the the live coding.
Stanford Computer Graphics Laboratory, for the Utah teapot model. Link: https://graphics.stanford.edu/courses/cs148-10-summer/as3/code/as3/teapot.obj