# CS5330 Project4 Report

Harshil Bhojwani and Joseph Farrell

March 19, 2024

## Project Description

The focus of this project was to give students exposure to the world of AR and some of its core components. The first part of the project was calibrating the camera in a way to be able to extract the 3d world points and transform it into a 2d camera coordinate system. This enabled us to virtually plot object in the 2d coordinate systems in a similar manner as it would appear in the world coordinate system.

Being in the nascent stage of developing these AR models, a checkerboard pattern was used as a target to extract the world coordinates and convert them to 2d camera coordinate system. The project is divided into 3 main programs, the first one being the camera calibration, being able to extract the 3d world points, detecting the corners, calculating a camera matrix, rotation, and translation matrix to be able to correctly plot objects.

The second part is where we create objects as per the world coordinates. Then we projected those objects virtually on our target (checkerboard).

The third part is to detect robust features. In our case, we choose to detect Harris corners. Harris corners can also be used to in camera calibration.

## Link to Video Demo

This is a link to a short demo video of our project:

CLICK HERE FOR LINK.

## Detect and Extract Target Corners

**Target**: *checkerboard*

**Camera**: *iPhone 12*

A checkerboard is being used for a target image, the system especially using the OpenCv built-in functions are quite prompt with detection of the checkerboard,

ony at times when the camera was overexposing or the target image had some glare, it was at times failing to detecting it.

## Select Calibration Images

Below are example calibration images with checkerboard corners highlighted.


Figure 1: Calibration Image Example (a)


Figure 2: Calibration Image Example (b)

## Calibrate the Camera

$$camera\_matrix = \begin{bmatrix} 2430.23 & 0 & 1106.86 \\ 0 & 2414.32 & 598.524 \\ 0 & 0 & 1 \end{bmatrix}$$

$$reprojection\_error = 2.56583$$

```
----------------------------------------

Distortion Coefficients: Pre Calibration
Empty
----------------------------------------

Camera Matrix: Pre Calibration
1 0 960
0 1 540
0 0 1
----------------------------------------

Distortion Coefficients: Post Calibration
0.356308, -1.77588, 0.0159564, 0.0347482, 3.90079,
----------------------------------------

Camera Matrix: Post Calibration
2430.23 0 1106.86
0 2414.32 598.524
0 0 1
----------------------------------------

Reprojection Error = 2.56583
----------------------------------------
```

Figure 3: Example Terminal Output Post Calibration

## Calculate Current Position of the Camera

$$t\_vec = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \qquad r\_vec = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

As I move the camera to left, but keep everything steady the value of $t_x$ increases. As I move it to the right the value of $t_x$ decreases. This makes sense because in the image as I move the camera to left the target is moving to the right in

the image, and vis versa. The rotation matrix values with left right movement remain largely the same.

Similarly, as I move the camera down $t_y$ decreases and as I move it up $t_y$ increases. This also makes sense because as I move down the target moves up, and vis versa. Again, the rotation matrix values remain largely the same.

The rotation matrix values change when I rotate the target, but the leave the camera stationary.

All of these values changes make sense.

## Project Outside Corners or 3D Axes

Below is the image of 3d Axes representing all 3 axes on checkerboard plotted from the first point $(0, 0, 0)$.

The x-axis end point: $(0, -2, 0)$

The y-axis end point: $(2, 0, 0)$

The z-axis end point: $(0, 0, 2)$

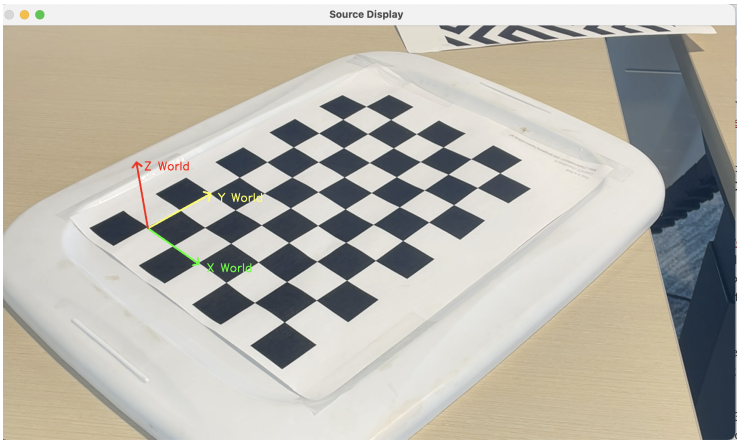


Figure 4: Image Displaying Axes (a)

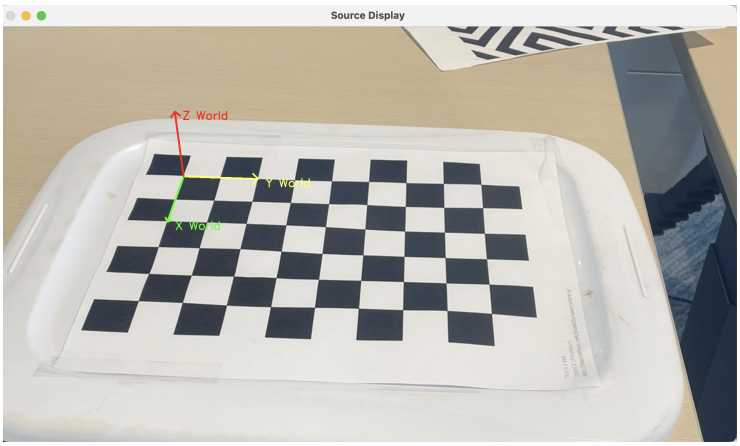


Figure 5: Image Displaying Axes (b)

## Create a Virtual Object

*Object #1*

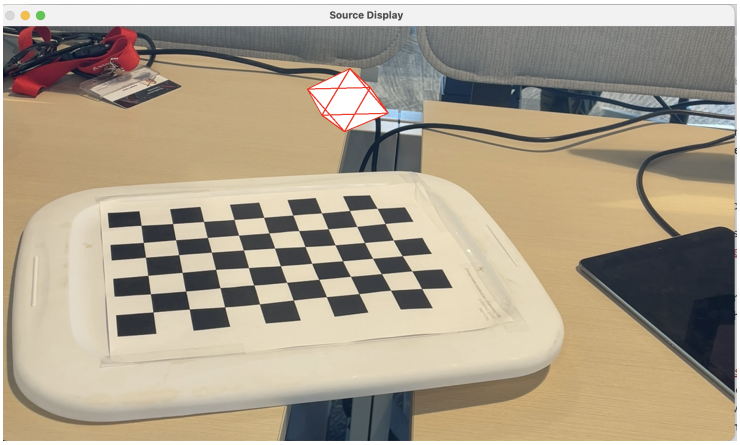


Figure 6: Virtual Object: Diamond (a)

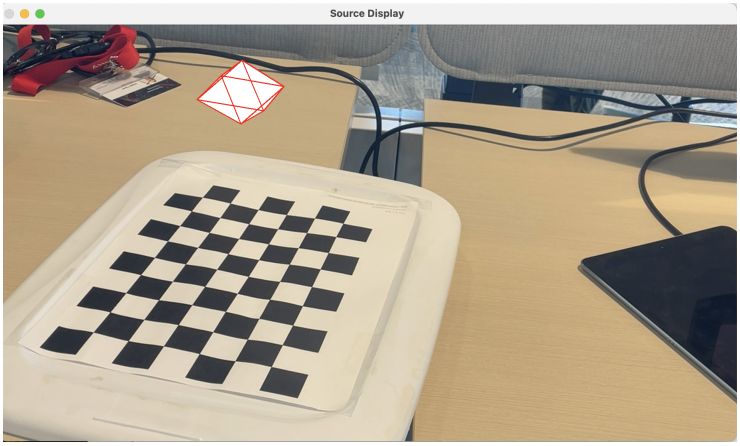


Figure 7: Virtual Object: Diamond (b)

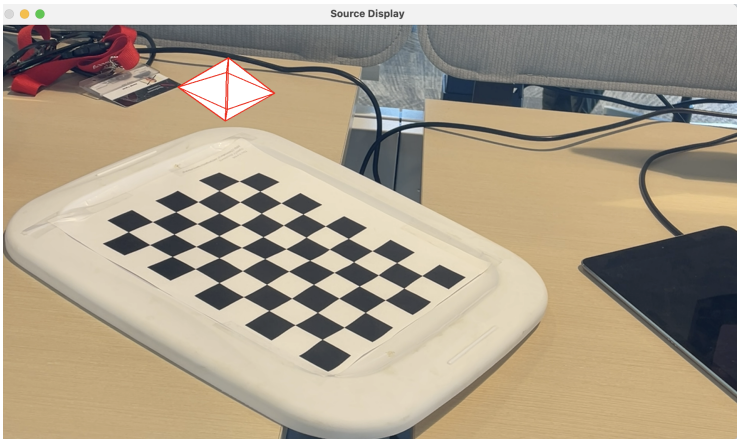


Figure 8: Virtual Object: Diamond (c)

This object is a diamond with the sides poly-filled in white color. The lines are colored in red.

*Object #2*

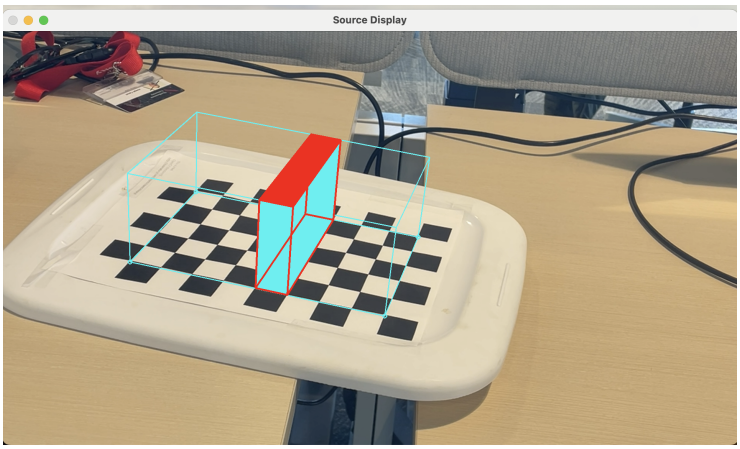


Figure 9: Virtual Object: Rectangles (a)

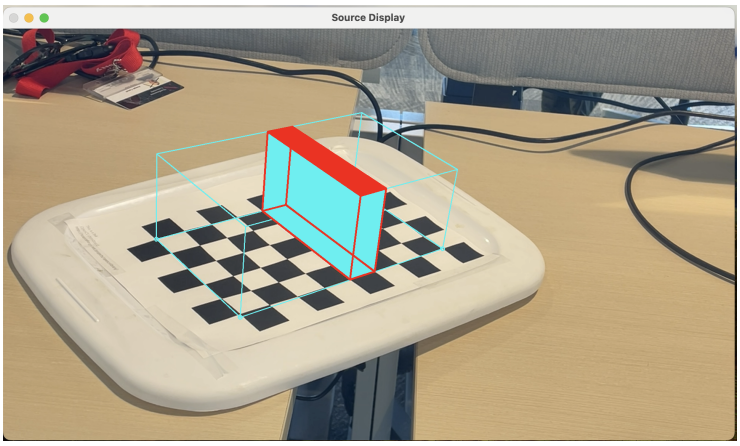


Figure 10: Virtual Object: Rectangles (b)

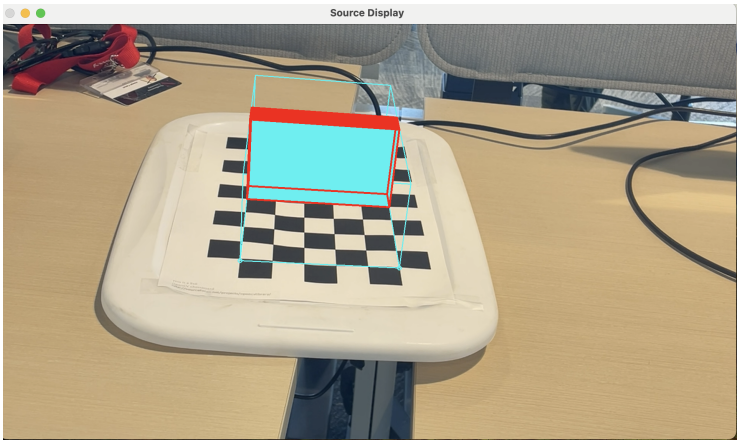


Figure 11: Virtual Object: Rectangles (c)

This shape a is two rectangles, one is poly-filled light blue with red edges and cuts across the target. The other traces the edges of the target and is just the lines.

For both of these objects we attempted (albeit unsuccessfully) to capture the depth using world and image coordinates, distance metrics, and sorting.

## Detect Robust Features

*Feature:* Harris Corners


Figure 12: Harris Corners (a)

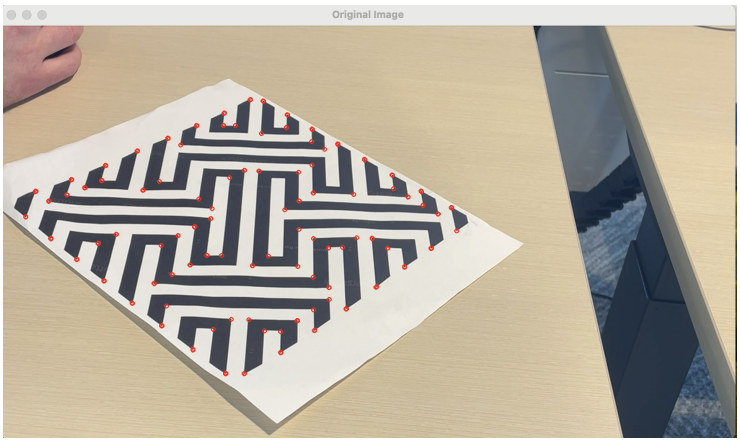


Figure 13: Harris Corners (b)

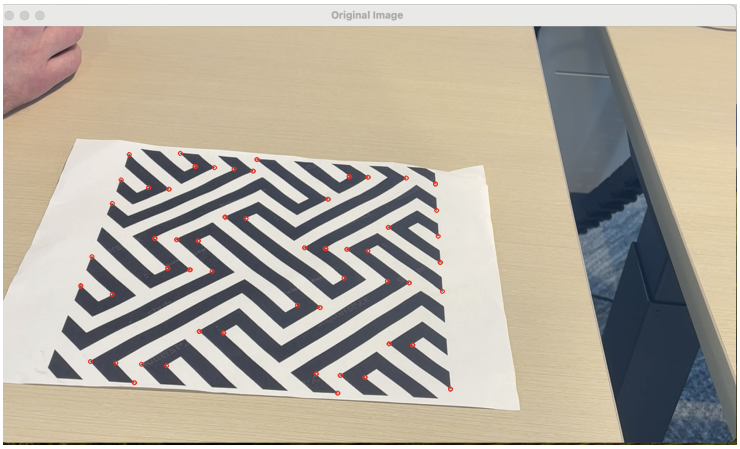


Figure 14: Harris Corners (c)

Above are the images of Harris corners which were detected as robust features.

By definition, it is important to detect robust features such as Harris corners especially when we have a stereo setup or multiple camera setup as something like a Harris corner which helps us extract features with large image value of the image gradients as they remain constant through out the object, and when setting up multiple camera systems to extract those particular features to be able to calibrate and orient the cameras as required.

We can use Harris corners by filtering out the Harris that are very close to each other. Then we can keep $n$ largest. We could use these corners to calibrate our camera, and then project points from world 3d to image 2d.

## Extension

For our extension, we effectively did two extensions in one. We created a more intricate scene while simultaneously making the target disappear. Our virtual scene consists of mountains, a small forest, the Sun, and the bank of a river.

For a curvy river bank we used multiple points and *cv::polyline*.

Again, we spent a long time trying to capture depth and draw our scene in accordance with the depth of each object. This proved unsuccessful under the current framework.

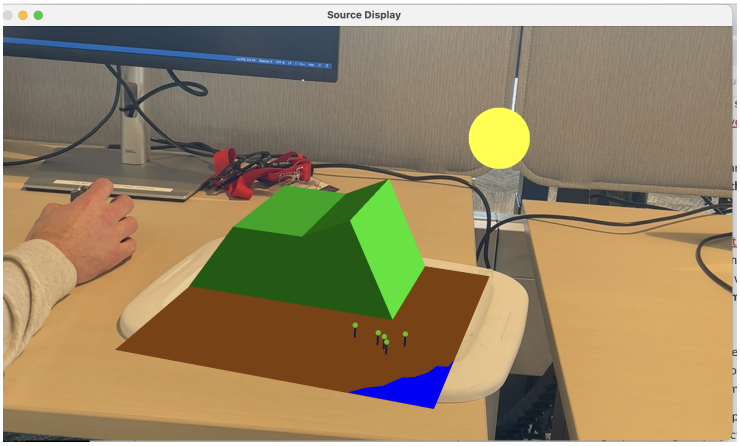


Figure 15: Extension Example (a)

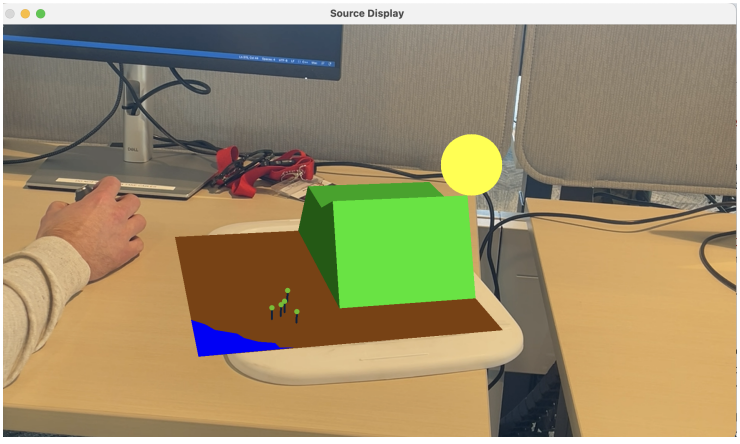


Figure 16: Extension Example (b)

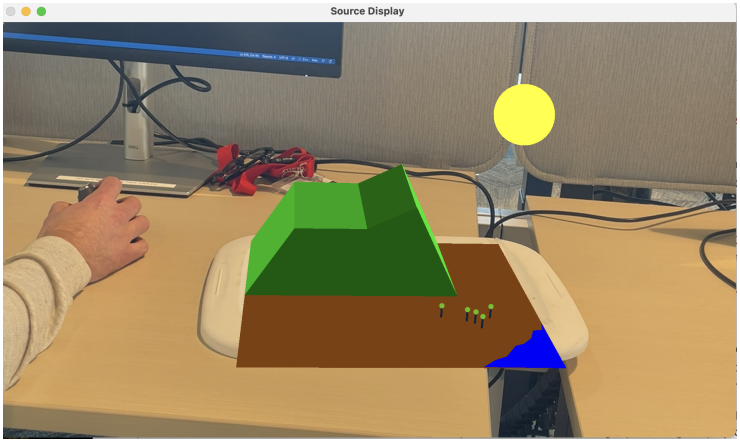


Figure 17: Extension Example (c)

## Reflection

This project gave us a good idea of how the fundamentals of AR work, both from a mathematical standpoint and from a programmable implementation. Opencv has enough built-in functions to be able to take care of most of the math, but it is important to understand what is happening under the hood. As we can get more complex with our implementation, this project will definitely enable us to dive much further in this field where there world coordinates would be from a perspective of a board.

## Acknowledgements

1. OpenCV Documentation. https://docs.opencv.org/4.x/index.html
2. Stack Overflow. https://stackoverflow.com/
3. Geeks for Geeks. https://https://www.geeksforgeeks.org/
4. Bruce Maxwell. CS 5330 Class Materials