Report 4: Calibration and Augmented Reality

> Edited by Hui Hu @hui hwoo

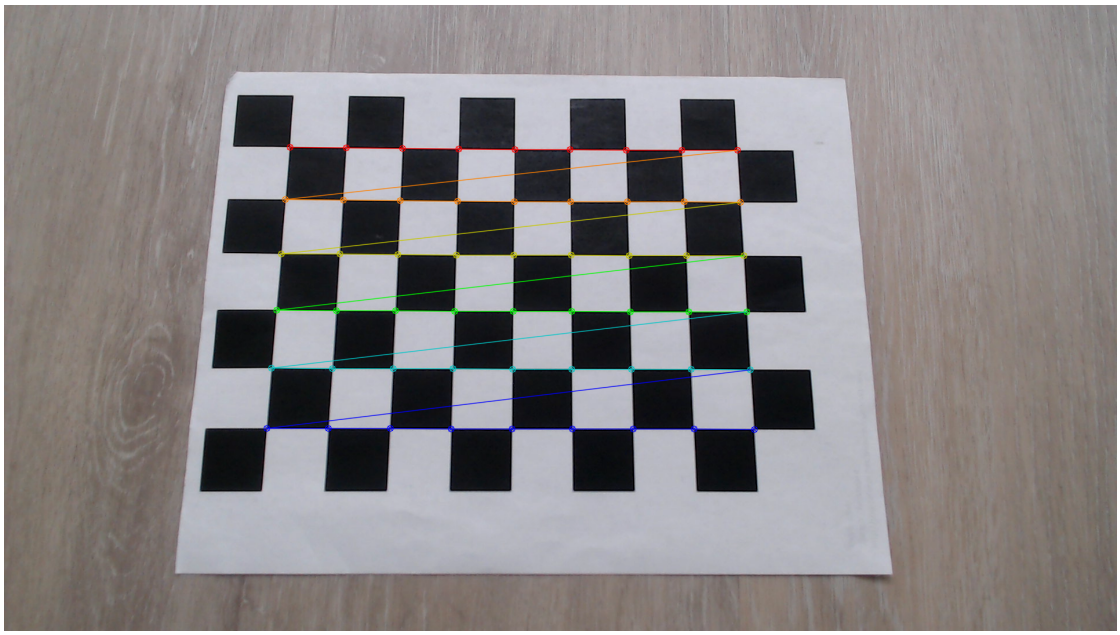
Description

This project involves learning how to calibrate a camera and using the calibration to generate virtual objects in a scene. The project uses a checkerboard pattern as a target, and the tasks involved include detecting and extracting the checkerboard corners, selecting calibration images, calibrating the camera, and generating virtual objects in a scene relative to the checkerboard.

Tasks

Task 1 : Detect and Extract Chessboard Corners

> The task is to develop a system that can detect and extract the corners of a chessboard. The system should draw the chessboard corners when it finds them and print out how many corners it finds, along with the coordinates of the first corner.



Chessboard corners

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

found 54 corners, first corner at (492.821,244.604)
found 54 corners, first corner at (492.848,244.596)
found 54 corners, first corner at (492.844,244.561)
found 54 corners, first corner at (492.845,244.557)
found 54 corners, first corner at (492.868,244.635)
found 54 corners, first corner at (492.89,244.617)
found 54 corners, first corner at (492.922,244.596)
found 54 corners, first corner at (492.82,244.591)
found 54 corners, first corner at (492.879,244.589)
found 54 corners, first corner at (492.906,244.561)
found 54 corners, first corner at (492.905,244.629)
found 54 corners, first corner at (492.868,244.58)
found 54 corners, first corner at (492.861,244.597)
```

Number of corners and the coordinates of first corner

Task 2 : Select Calibration Images

> The task involves letting the user specify a particular image to be used for calibration and saving the corresponding corner locations and 3D world points. The user can type 's' to store the vector of corners found by the last successful call to findChessbordCorners into a corner_list.

The calibration images with highlighted corners in different angles:

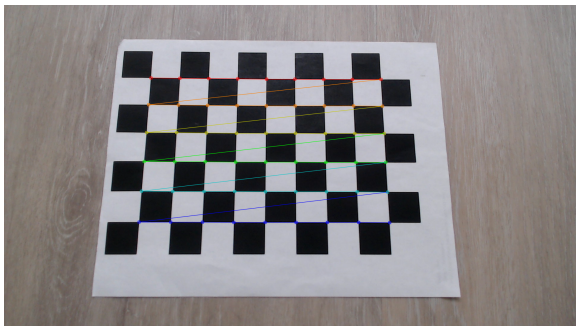


Image 1

Image 2

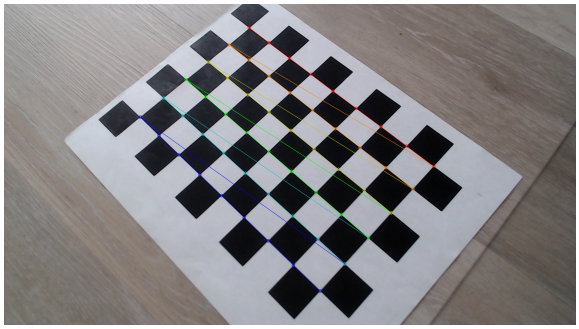


Image 3

Image 4

Task 3 : Calibrate the Camera

> The task involves using the cv::calibrateCamera function to perform camera calibration given a set of calibration frames. The user should be required to select at least five calibration frames before running the calibration. Alternatively, the calibration can be continuously updated each time a new calibration image is added, and the per-pixel error should be displayed to the user after each calibration.

```
PROBLEMS 19   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

Camera matrix before:
[1508.348523652425, 0, 903.3268570635672;
 0, 1508.348523652425, 477.4940811336128;
 0, 0, 1]
Distortion coefficients before:
[0.1147098273165217;
 -0.5832280725801583;
 -0.009442310359543149;
 -0.004977038815036649;
 1.073225014056951]

Camera matrix after:
[1506.344045240945, 0, 903.3757064749266;
 0, 1506.344045240945, 478.8835338924646;
 0, 0, 1]
Distortion coefficients after:
[0.1159254438056125;
 -0.5795664835819834;
 -0.009414828458673656;
 -0.005075287681105642;
 1.033683235912335]
error: 0.486494
```

After adding five calibration frames, following is the results I got:

frame num	**Camera matrix**	**Distortion coefficients**	**Error**

frame num	**Camera matrix**	**Distortion coefficients**	**Error**
6	[1503.53979769491, 0, 906.2285254959652; 0, 1503.53979769491, 481.9058372583199; 0, 0, 1]	[0.1092000109308758; -0.5442535808283032; -0.009276841839472645; -0.004700750635345242; 0.9759116411797532]	0.480738
7	[1502.450007032078, 0, 906.3635771357975; 0, 1502.450007032078, 482.4544016693633; 0, 0, 1]	[0.1103239325208851; -0.553126657241427; -0.009266253303367104; -0.00469676416958887; 0.9917498852471598]	0.48069
8	[1500.672185340607, 0, 906.6379082054754; 0, 1500.672185340607, 483.3568997413904; 0, 0, 1]	[0.110985759228179; -0.558599304792357; -0.009241431762351414; -0.00467949394116226; 1.001157909751761]	0.478772
9	[1501.613355455753, 0, 906.2284165119376; 0, 1501.613355455753, 482.781506403054; 0, 0, 1]	[0.1111871233403712; -0.5599546982585718; -0.009258662681362254; -0.004749396582649825; 1.009235645311438]	0.479023
10	[1501.917203297633, 0, 906.2753667753334; 0, 1501.917203297633, 482.6216879511295; 0, 0, 1]	[0.1110924465629136; -0.5598760056553544; -0.009263625020244829; -0.004734376637540889; 1.010346870618754]	0.47869

1. **Error**

   It's obvious that the error becomes smaller and smaller when we add more frame, and the values of error are all less than a half-pixel.

2. **Camera intrinsic matrix**

   We choose the Camera matrix after adding 10 frames to explain.

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

```
[1501.917203297633,                0, 906.2753667753334;
                 0, 1501.917203297633, 482.6216879511295;
                 0,                0,                1]
```

- According to above result, $f_x = 1501.917203297633$, $f_y = 1501.917203297633$, they are the same value, which means we have gotten really good result.
- On the other hand, $c_x = 906.2753667753334$, $c_y = 482.6216879511295$, and the size of frame is 1920×1080

  $$1920/2 = 860 \text{ and } 1080/2 = 540$$

  c_x, c_y values should be close to the initial estimates of the center of the image, although they are not the same value, after adding more frames, they must be the same.

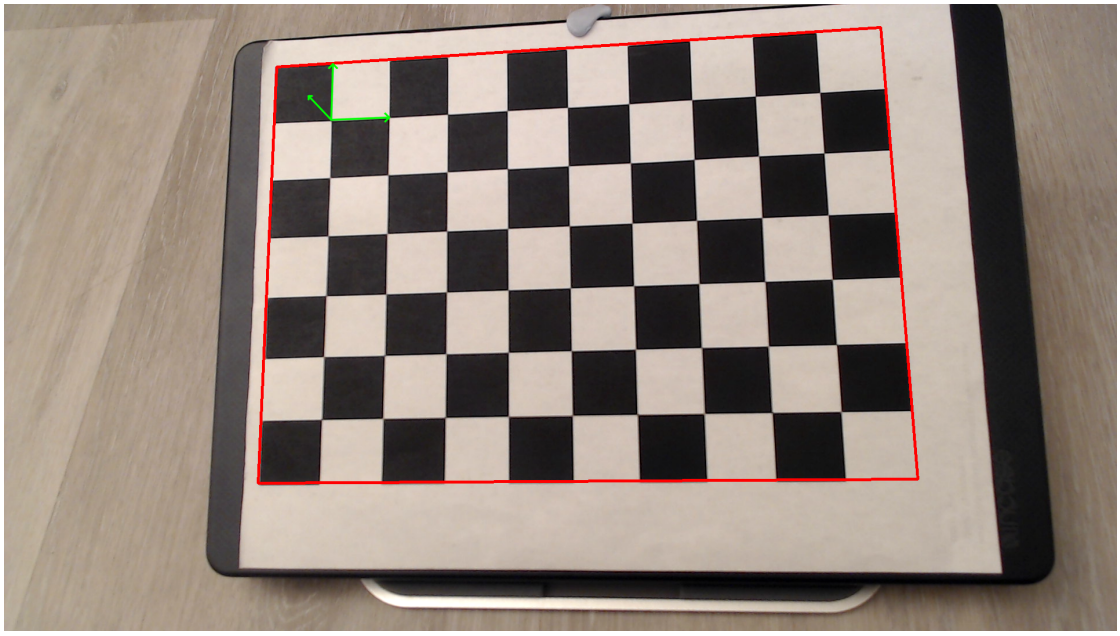
Task 4 : Calculate Current Position of the Camera

> The task involves writing a program that reads camera calibration parameters from a file and then starts a video loop. For each frame in the video, the program should attempt to detect a chessboard. If a chessboard is detected, the program should grab the locations of the corners and use the solvePNP function to calculate the board's pose, including its rotation and translation.

```
PROBLEMS 14   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

Rotation:
[-0.07720899191000584;
 -2.924696017694498;
 0.7719643014693345]

Translation:
[5.126099322896276;
 3.195656216726886;
 13.97324920643885]

Rotation:
[-0.07719775456807054;
 -2.924631226851509;
 0.7718448646494336]

Translation:
[5.12582739179358;
 3.19584175860378;
 13.97380632335016]

Rotation:
[-0.07729724190036746;
 -2.924637956914062;
 0.7717824204938551]
```

Task 5 : Project Outside Corners or 3D Axes

> In this task, the program should use the pose estimated in the previous step to project the 3D points corresponding to the four outside corners of the chessboard onto the image plane in real-time as the chessboard or camera moves around.

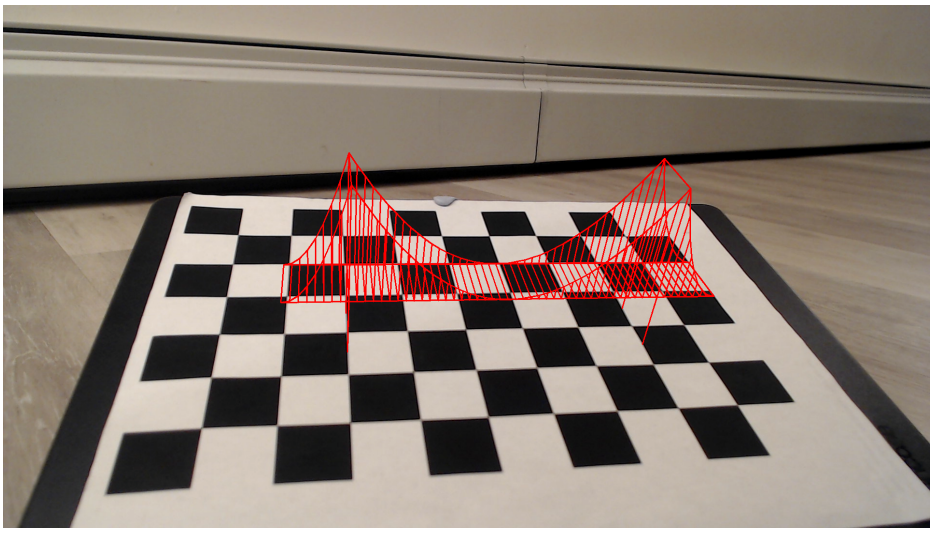


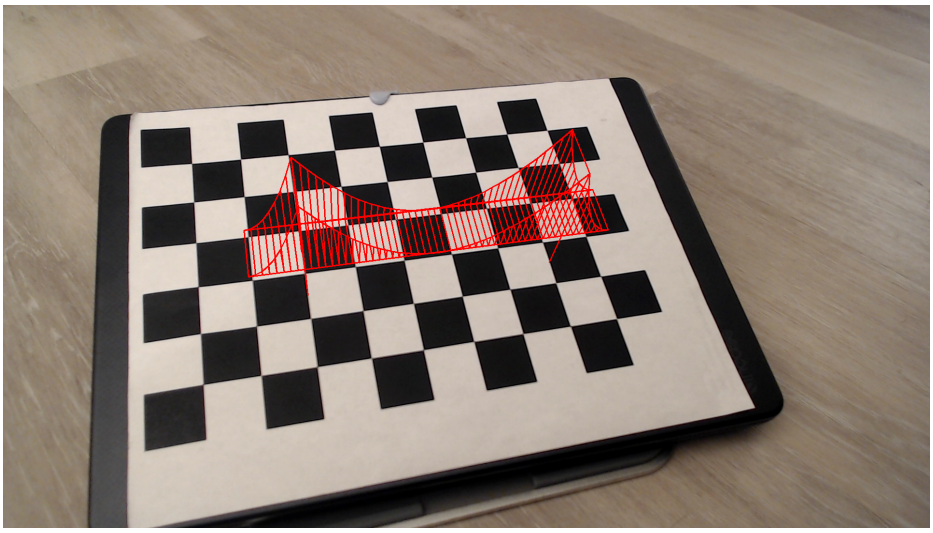
- **Green** part is the 3D axes.
- **Red** part is the outside corners

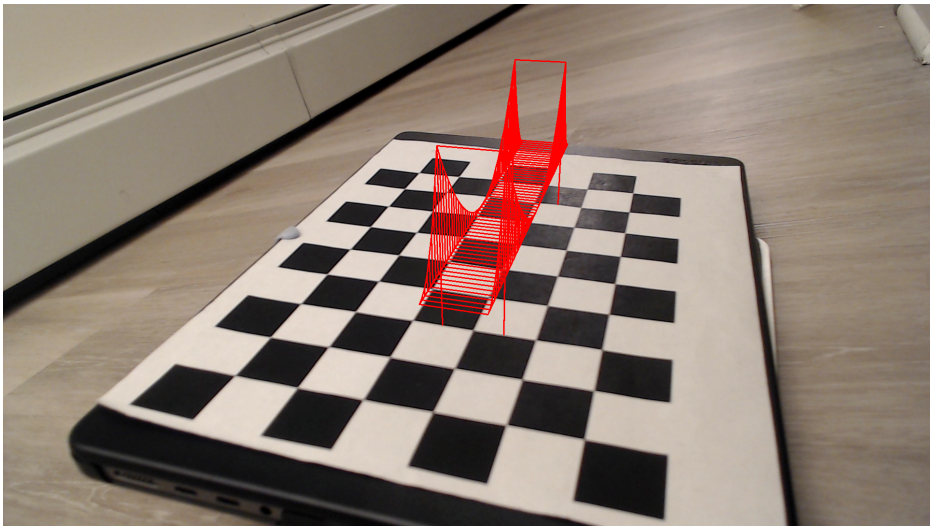
Task 6 : Create a Virtual Object

> In this task, the program should construct a virtual object in 3D world space made out of lines.

Since my campus is in San Francisco, so I choose to reproduct the **Golden Gate Bridge** in this litter chessboard.



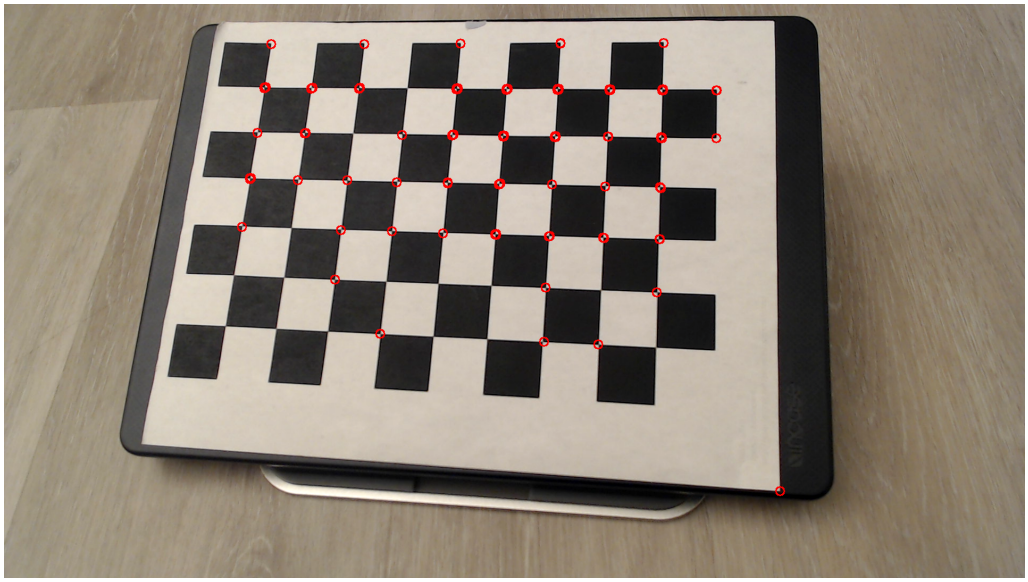




Task 7 : Detect Robust Features

> The task involves writing a program to detect and display features in a video stream using a particular feature detection method such as Harris corners or SURF features. The program should also show where those features appear on a pattern of the user's choice.

- **Image**



- **Video**

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/8f84ff1b-bed0-43b5-8731-caccb449422f/Untitled.qt

The red circles are the corners that I found using Harris corners Features. Once you play the video, you will notice that number of corners is changing all the time, which means the corner we found is not stable.

Advantages:

1. Robustness: The Harris corner detector is robust to noise, which means it can detect corners even in images with a lot of noise.
2. Invariance to image rotation: Harris corners are invariant to image rotation, which means that the algorithm can detect the same corners in images that have been rotated.
3. Localization: The Harris corner detector can accurately localize corners within an image.
4. Speed: The Harris corner detector is relatively fast and can process images in real-time, making it useful for applications such as object tracking and video analysis.

Overall, the Harris corner detection algorithm is a powerful tool for detecting corners in images and can be used in a wide range of applications, including computer vision, robotics, and image processing.

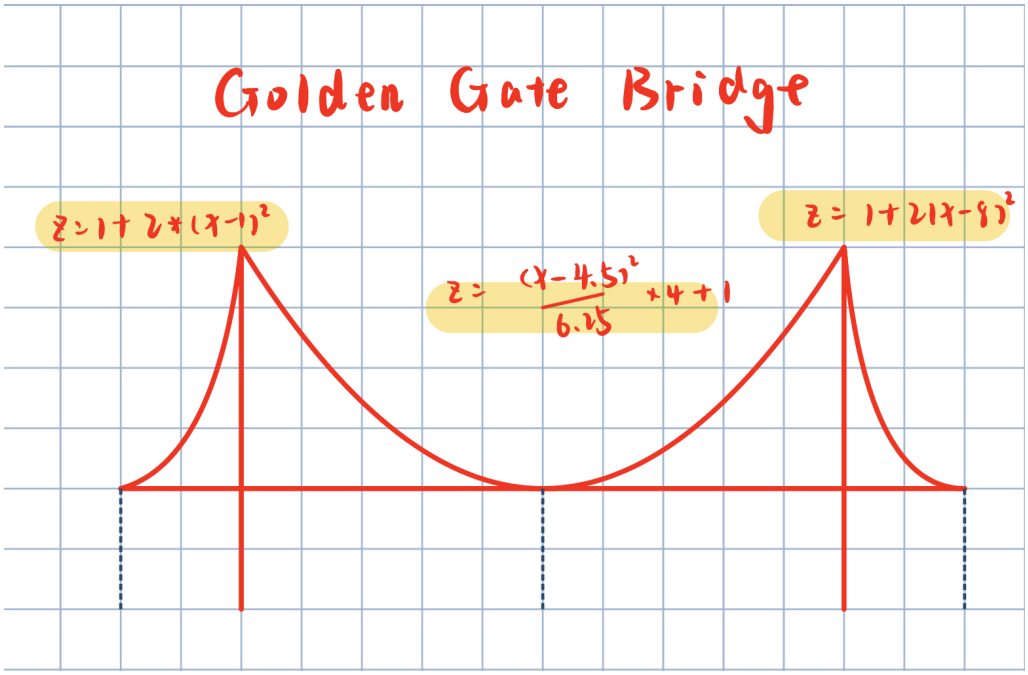
Disadvantages:

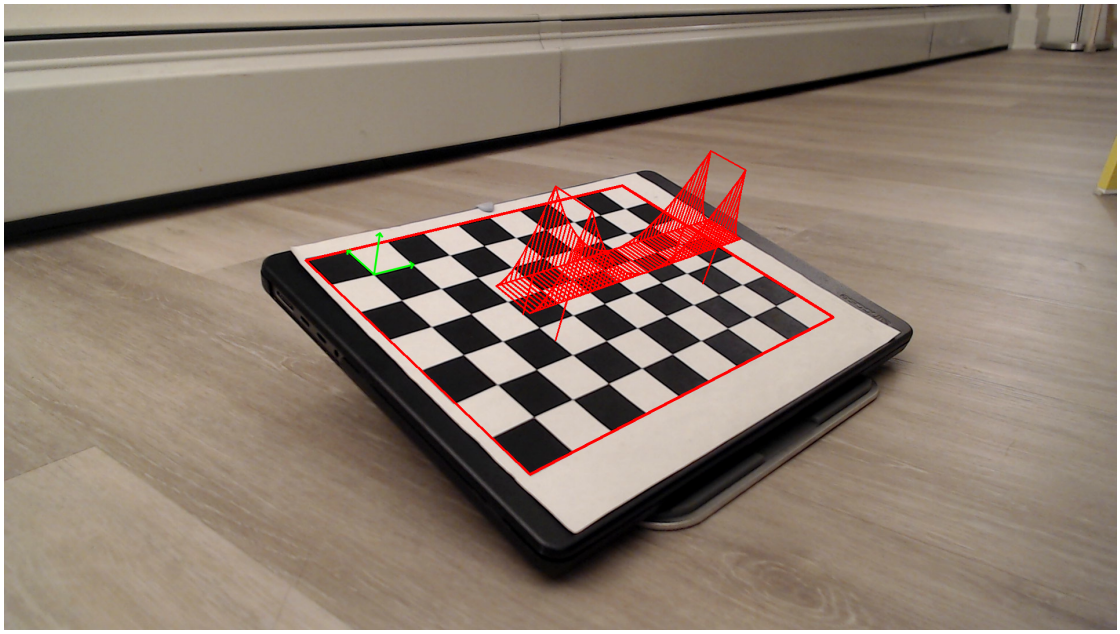
1. ensitivity to image noise: While the Harris corner detector is robust to noise, it can still produce false positives or miss corners when there is too much noise in the image.
2. Sensitivity to changes in lighting conditions: The Harris corner detector is also sensitive to changes in lighting conditions, which can cause it to miss corners or produce false positives.
3. Parameter tuning: The Harris corner detector has several parameters that need to be tuned, such as the window size and the Harris response threshold, which can require some trial and error to get the best results.

Extensions

Extension 1: Golden Gate Bridge

- **Image**



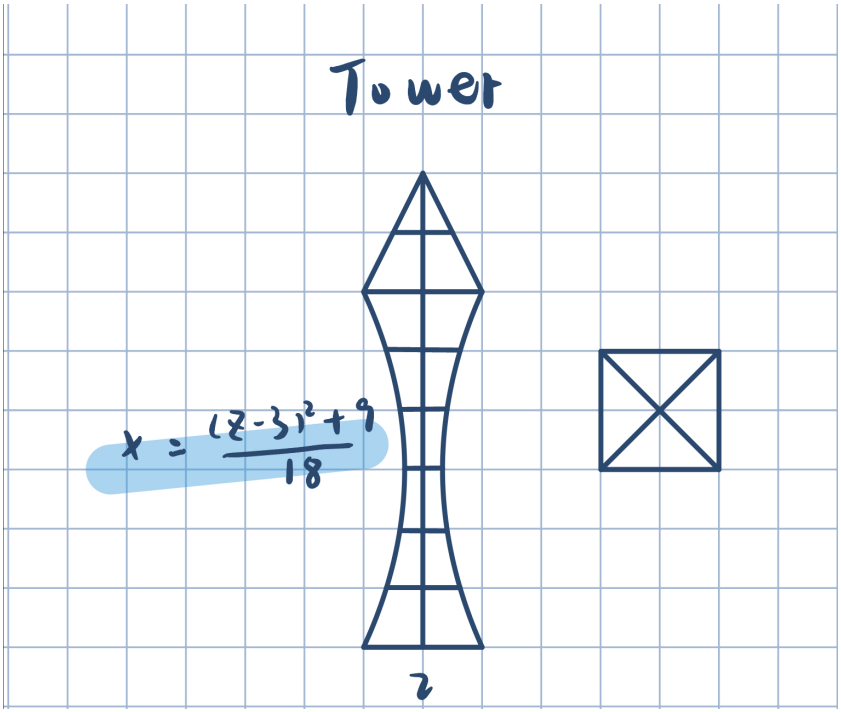


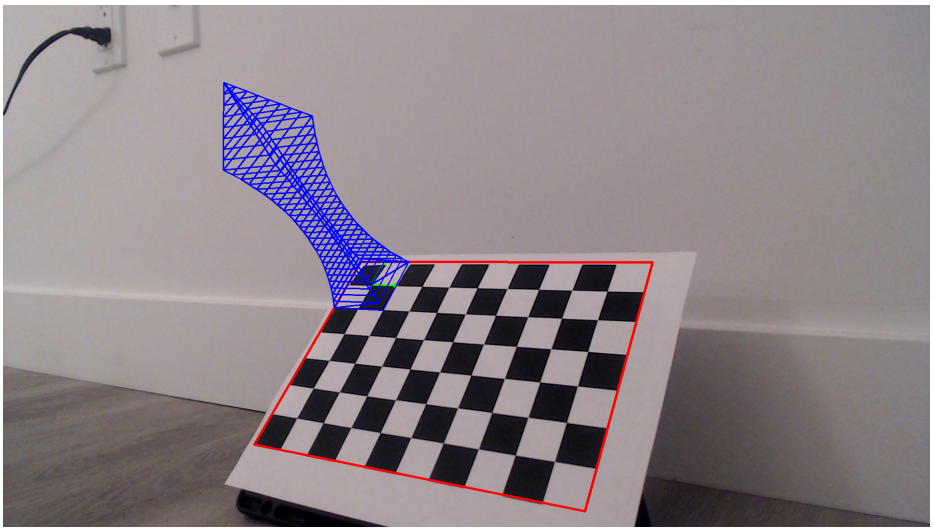
- **Video**

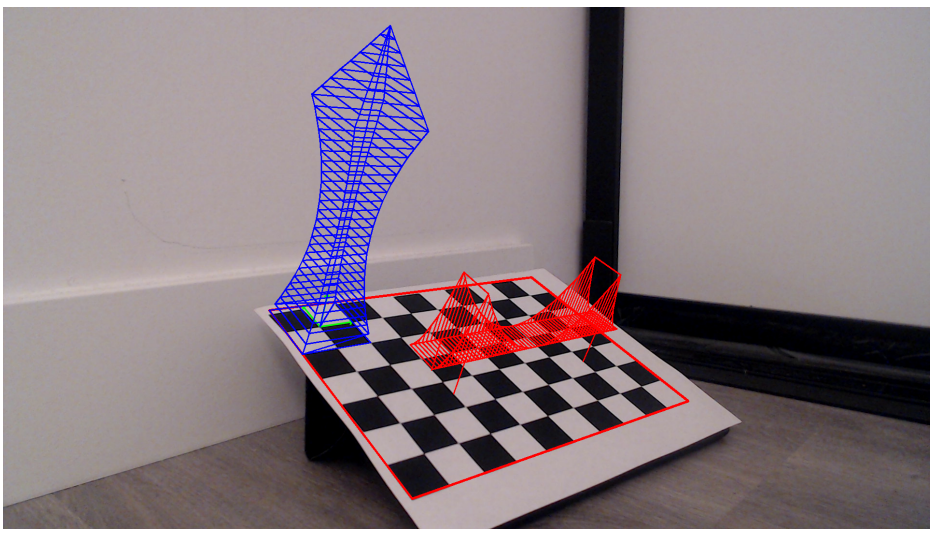
  https://s3-us-west-2.amazonaws.com/secure.notion-static.com/bda0323d-1400-4a4c-bce3-b270d53f1cc1/Untitled.qt

- I projected SF Golden Gate Bridge in the sence, although it’s a simple version, I stiill need to design each line and combine them into one image to display part beauty of this bridge.

Extension 2: Tower


Tower
$x = \frac{(z-3)^2+9}{18}$
z

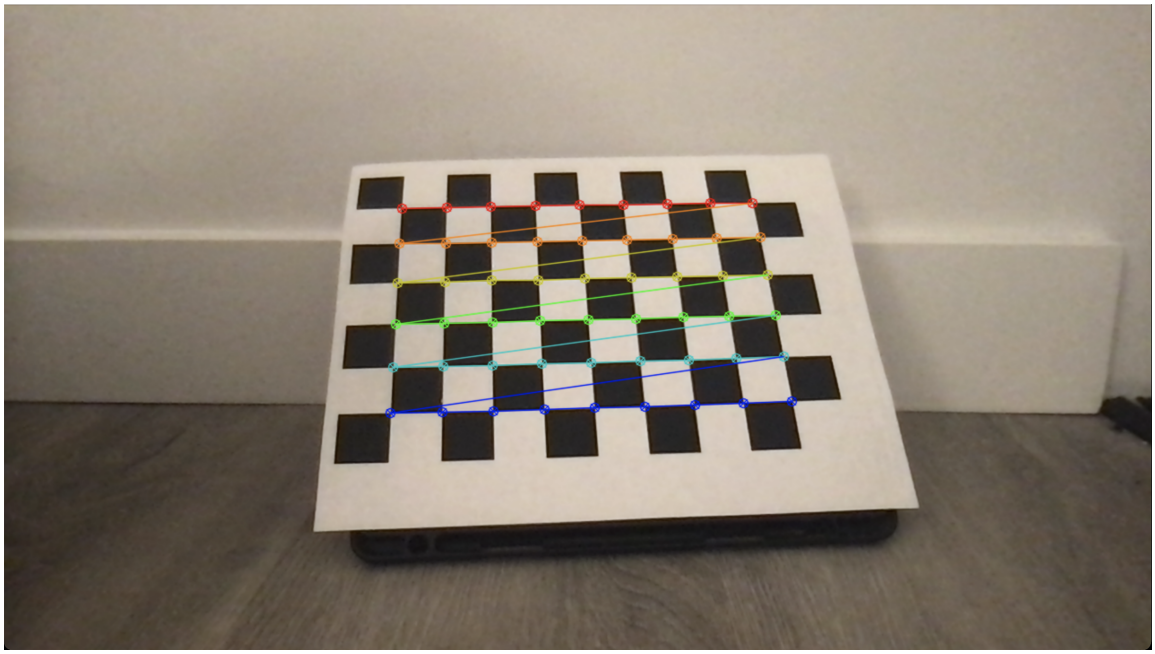




Extension 3: Comparison between different camera

- Image quality

Webcam



Build-in camera

- Error value

Webcam	Build-in camera
0.378862	0.713454

Webcam	Build-in camera
0.379219	0.714794
0.379217	0.716196
0.380374	0.718021
0.37836	0.718767

According to the image quality and error values, we know that different camera do effect the results of calibration, therefore, if we want to more corners (details) of one image, we could choose better camera, rather than choose more complicated and time-consuming algorithms.

Reflection

After completing the tasks related to computer vision and augmented reality, I have gained a deeper understanding of the underlying concepts and techniques involved in creating these systems. I learned how to calibrate a camera, estimate its pose, and project virtual objects onto a real-world image. Additionally, I explored different feature detection methods such as Harris corners and SURF features and learned how they can be used as a basis for augmented reality.

I found that the OpenCV library was extremely helpful in implementing these tasks. It provided me with a wide range of functions and tools that enabled me to perform complex operations with ease. I also appreciated the extensive documentation and examples that were available online, which made it easier to learn and understand the various concepts.

Overall, this project was a great learning experience, and it gave me a deeper understanding of image processing and computer vision.

Acknowledgement

[1] *OpenCV modules*. OpenCV. (n.d.). Retrieved February 10, 2023, from https://docs.opencv.org/4.x/