CS 5330 Project-4

Saideep Arikontham

Title: Calibration and Augmented Reality

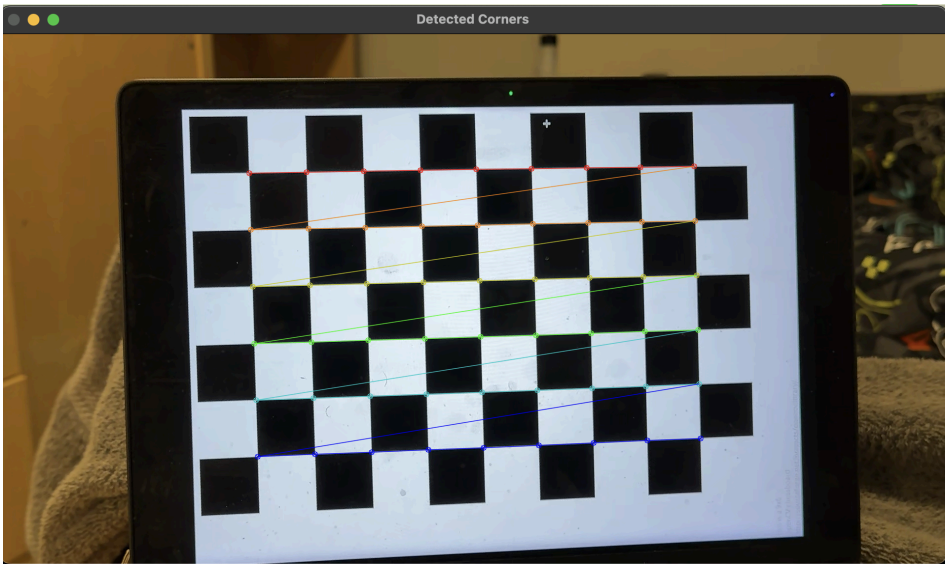
Project Overview

This project focuses on camera calibration and augmented reality (AR) using OpenCV. The primary objective is to detect a calibration target, calibrate the camera based on extracted corner points, and use the calibrated parameters to overlay virtual objects onto real-world scenes. The calibration process involves detecting a checkerboard pattern, computing camera parameters, and estimating the camera's position in the 3D world using the checkerboard corners.

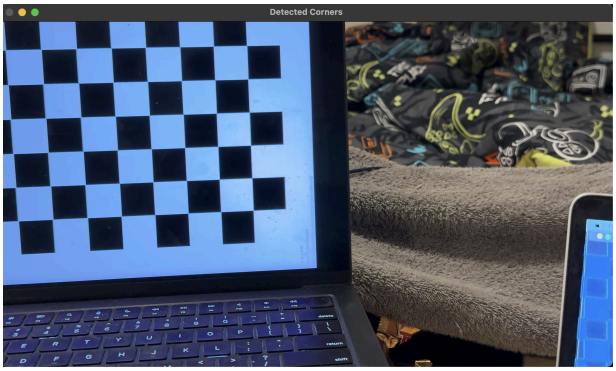
Once the calibration is complete, virtual objects such as a pyramid, house, and letter 'S' are projected onto the detected checkerboard, ensuring that these objects remain correctly positioned and oriented as the camera moves. Additionally, I have implemented the functionality to replace the checkerboard pattern with an image so that the target checkerboard pattern is replaced with another image. I have also computed and analyzed features such as Harris corners and SURF and tried to understand their usage.All these have been done with a live video stream using OpenCV.

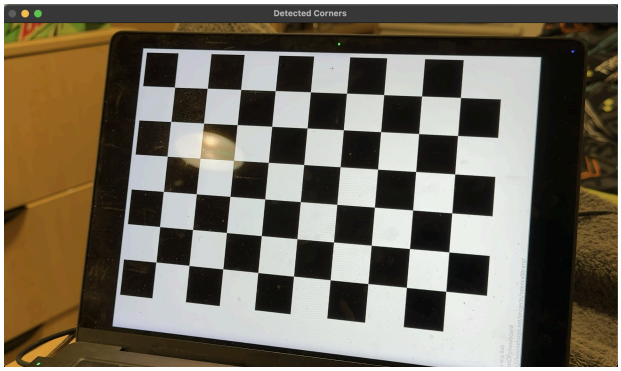
Directed Tasks and required images

1. **Detecting and extract target corners (Req. result 1):** I have chosen the target to be checkerboard. I used the functions findChessboardCorners and drawChessBoardCorners to detect and draw the checkerboard target corners. I am using a laptop screen to display this object and work with it. Below is a reference image which shows that the corners are being identified by the system.

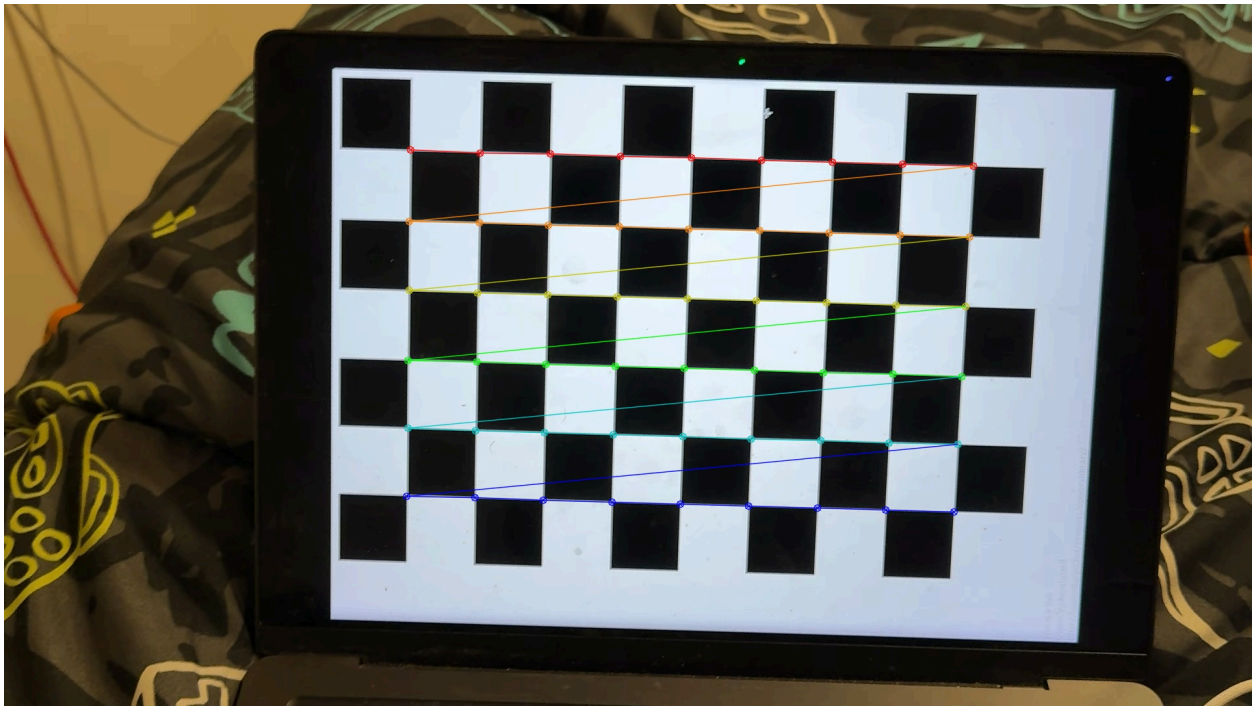


- We can see that the corners are being highlighted properly from top left to bottom right corners properly.
- If either of the corners goes out of picture, the corners cannot be found. Another issue is glare as I am using a laptop screen to display my target. Glare on one of the corners results in the target not being located properly. Below are examples for the two cases:





2. **Select Calibration Images (Req. result 2):** For this step, I implemented functionality that allows the user to manually select images for calibration. When the user presses the 's' key, the system captures the current frame and stores the detected checkerboard corners in a corner_list. I generated a point set for the pattern size 9x6 that represents the corresponding 3D world coordinates of these corners, assuming a unit square measurement system. The point_list stores the same number of point_set points that are present in corner_list. When the user clicks “s”, the current frames corners are detected and the respective coordinates are stored. I also save the image where corners are drawn as a calibration image for reference. Below image is saved as calibration_image_2.jpg in “images” folder.



3. **Calibrate the camera (Required result 3)**: Once enough images (at least five) were collected for calibration, I implemented a system that allows the user to run the calibration process. When the user presses 'c', the program uses the OpenCV calibrateCamera function to compute the camera's intrinsic and extrinsic parameters. This function takes in the point_list (3D world coordinates), corner_list (2D image coordinates), and the image size, and outputs the camera matrix, dist_coeffs, rotation vectors, and translation vector. I have to save the calibration images and calibrate the camera in the same program run. I could have implemented it in such a way that I directly read the above saved calibrated images and use those without the need for any dependency but I thought that would disrupt the real-time inference. Below are the results that I obtained.

- The Camera matrix, distortion coefficients and Reprojection error printed are displayed below. I have used my phone camera as a primary source for video stream and as indicated got a reprojection error of ~2.6.

```
Initial Camera Matrix:
[1680.427564143085, 0, 870.3076802144284;
 0, 1680.427564143085, 534.5313597050341;
 0, 0, 1]
Distortion Coefficients:
[0.05789357811156783, -0.1412011786964448, 0.004530187303798629, -0.01322424019408188, 0.3904267548893471]
Reprojection Error: 2.672111
```

- The below are the calibration_results.csv contents (camera matrix and distortion coefficients) that I am storing for further analysis in later tasks.

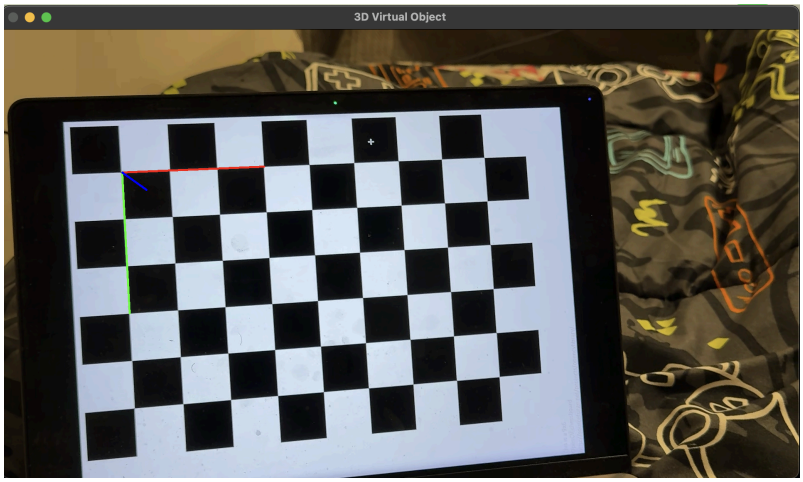
```
1680.427564,0.000000,870.307680,0.000000,1680.427564,534.531360,0.000000,0.000000,1.000000,
0.057894,-0.141201,0.004530,-0.013224,0.390427,
```

4. **Calculate current position of camera (Req. result 4):** I enhanced my current program for this task. I read the calibration_results.csv file contents from above to start with this task. I detect the target, detect its corners and use OpenCV's solvePNP function to get the camera pose relative to translation and rotation. When I move from left to right, the X axis value of the translation vector from X axis is changing while the rotation vector is almost the same. Moving from down to up changes Y axis value and moving forward or backward is changing Z axis values. When I tilt the camera side-ways, the Z axis component of the rotation vector is changing. This indicates that the translation vector values and rotation vector values are reflecting the action that is being performed. The program prints output as shown below. The below are the results right before I tilted the camera sideways.

```
Translation Vector (tvec): [-2.795345636231696, -0.2507235563884029, 23.59994532988149]
Rotation Vector (rvec): [2.592091713125938, 0.3811251540388375, -0.7918689050854927]
Translation Vector (tvec): [-2.338830186556043, -0.53735946367968, 23.94058890658092]
Rotation Vector (rvec): [2.552073454142986, 0.3867068037907397, -0.8357873828111868]
Translation Vector (tvec): [-1.776250859601882, -0.5194677857609792, 24.24513386452026]
Rotation Vector (rvec): [2.537621098624284, 0.3942529426052422, -0.8597409191381873]
Translation Vector (tvec): [-1.52285186350675, -0.5530167942893162, 24.70309232497667]
Rotation Vector (rvec): [2.536021158039148, 0.4053164992664969, -0.8920850968707064]
Translation Vector (tvec): [-1.263243164429887, -1.213755381289451, 24.918732234344]
Rotation Vector (rvec): [2.497166628640495, 0.4072345746018559, -0.942926091323024]
Translation Vector (tvec): [-0.3906094430272669, -1.188668031335986, 25.37530244020263]
Rotation Vector (rvec): [2.435296121890693, 0.4019152191623155, -0.9955906059695079]
```

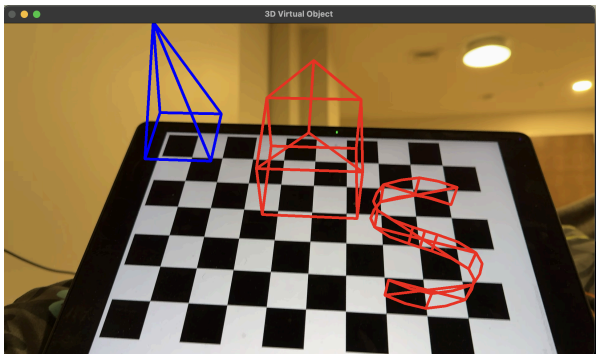
5. **Project outside corners or 3D-Axis (Req. result 5):** For this task, I am displaying the projected axis in 3D which is attached to the origin. The user can press "p" to project the 3D axis on the corners and get the below output. We can see the axis moving with respect to the moving camera. Below is the reference link for the video displaying projected axis and interaction with it.

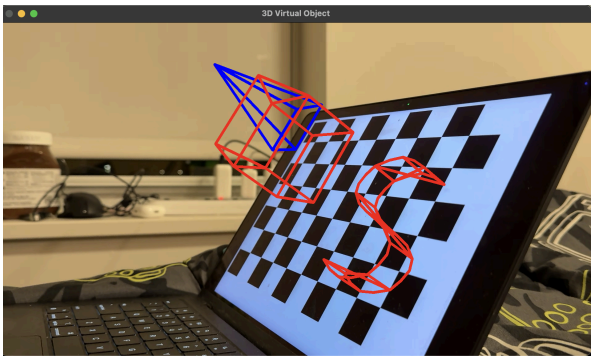
https://drive.google.com/file/d/1RiJsdgwiLAtEuEGT9uX-2zzIQ3oFWfcz/view?usp=drive_link



- We can also see that the axis is being perfectly displayed on the origin point which is the top left corner.

6. **Creating virtual objects (required result 6):** Now that we have all the required work in place, I can create virtual objects. The point set values are the actual corner points in the 3D world. We can now use these values as reference and build virtual objects in place. The user can press “t” to display 3 virtual objects. The first object would be a simple pyramid. The second object would be a house object and the third object is a letter “S”. I have used ChatGPT’s help to create this version of the letter “S”.

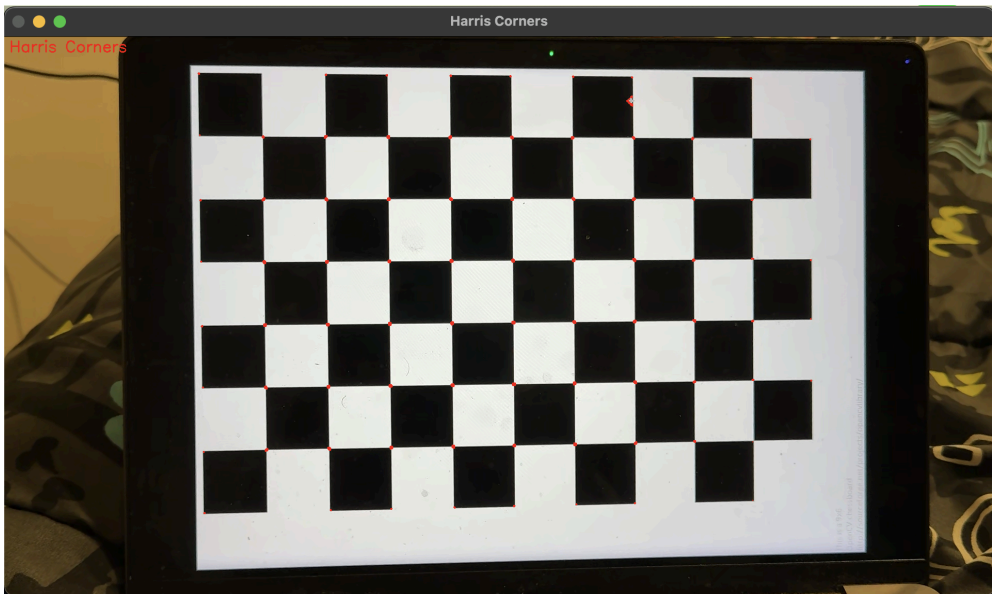


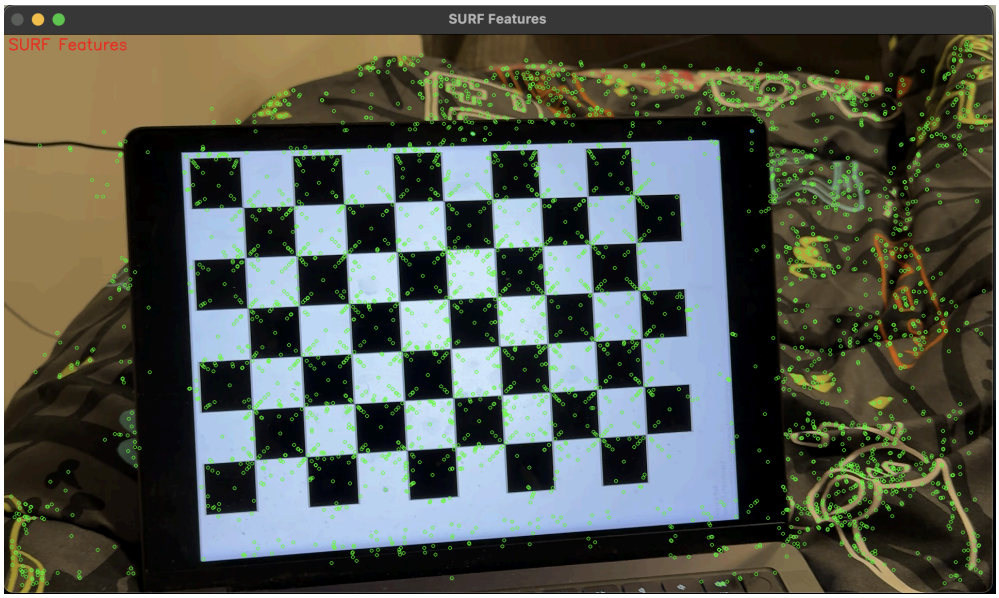


- I first chose the coordinates for the vertices and then connected the edges for a simple pyramid image. But for complex images. I had to work step by step to lay the bottom layer first and then the top layers. Simply, Work from lowest value of Z axis coordinate to highest value of Z coordinate. This made connecting the edges easier as well.
- For this version of “S”, I use GPT’s help as I wanted to implement something more complex, now that I understood how the vertices and edges are connected.
- I then selected an offset value x and y which I was adding to X and Y axis values of the object points. This helped in moving the object on the board.

- The user can press “t” to display these objects.

7. **Detecting Robust Features (Req. result 7):** I created a new code file implementing these features (feature.cpp). The user can press “h” to get Harris corners on the video stream and “s” for SURF features on the video stream. Below is one image for each of these features.



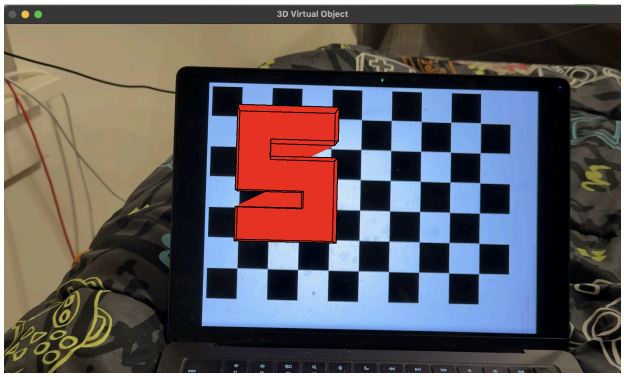


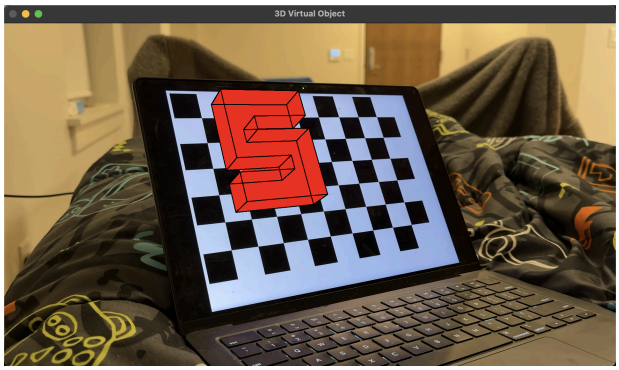
- Harris corners successfully identifies different corners on the checkerboard, something similar to findChessBoardCorners.

- But for SURF features, we can see that there is a distinctive pattern at the edges and there are very rich features values here with SURF values. The background features are random and can be considered noise here.
- We can also use these values to do object detection, camera pose estimation and more.

Extensions

1. **Objects with faces and their own color:** Instead of just displaying the object, I created a system to color the object faces with one color so that the object appears to be a solid object. The user can press “f” to display the object. I have built another “S” object on my own, this time with red face color. Below is the sample output image of the same.





- This output is exactly not what I expected but was not able to correct it due to the time constraints.

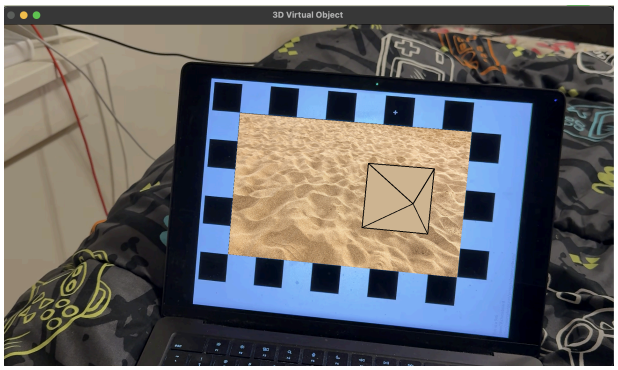
2. **Making the target not look like a target anymore:**

a. We replace the detected checkerboard pattern with an overlay image. I used a sand image as my overlay image. First, we project the 3D checkerboard points onto the 2D image using projectPoints, mapping real-world coordinates to the camera view. It then extracts the four outer corners of the checkerboard and defines corresponding corners in the overlay image. A homography transformation is computed to warp the overlay image onto the detected checkerboard area. The warped image is then blended into the original frame, effectively replacing the checkerboard with the overlay.

- The user can press “r” to display this. Additionally, I also display a colored pyramid object as well.

b. This pyramid object also moves within the overlaid sand image. This object bounces between the walls and always lies between the 9x6 checkerboard corner region (which is

being replaced with our own image). The goal was to make it look like there is a pyramid in sand.



- The image actually appears to be moving outside the board border but this is actually due to the fact that the pyramid is floating (Z axis has a positive value).
- I have replaced those values with 0 to demonstrate that the moving object always stays within the board borders which is being replaced by sand image.

Below is the overview of the entire system showing interactivity with key presses and outputs.

https://drive.google.com/file/d/1B8MRUBZs8Pu27Epee-lSTIBC0y15cYXt/view?usp=sharing

Below is a video to show outputs of tasks such as displaying projected axes, objects and extensions.

https://drive.google.com/file/d/1Vl7TV4worDgafOCeEfI5wXEqlIEn8zGE/view?usp=sharing

- I have also added the functionality to display a message indicating that “No Corners Found” if the system is unable to find corners in the video’s frame.

Reflection

Through this project, I gained deeper insights into camera calibration, pose estimation, and augmented reality techniques. The most challenging aspect was ensuring accurate calibration, as slight errors in corner detection significantly impacted virtual object placement. Implementing pose estimation was an essential learning point, as it provided real-world applicability for AR applications. It was really cool to see how objects are being displayed in AR. Compared to other projects, this has been a bit challenging to understand considering the vast theoretical knowledge required. However, working on this project definitely has improved my understanding for how objects can be projected to a 3D world.

Acknowledgements

The best resources I found for this assignment were ChatGPT and cloud recordings of classes. I used ChatGPT to understand errors that I encountered while working on this project. ChatGPT also helped me understand how few OpenCV functions work internally, different functionalities that I can implement and other ideas that I could use to build a robust system. Building objects was really a complicated task and it was difficult to build some complex objects. It took some time to get used to it but GPT and Claude helped me understand it in a better way. I have also used ChatGPT and Claude to implement the target replacing extension but I made sure to understand what exactly was happening there. I added a functionality of moving objects from project 1 that I implemented in this project as well. The actual idea was to replace the target with an image of outer space and put a moving ufo object that stays within the boundaries. But building a UFO object was harder than expected. Working on these projects helps me to understand things in a better way and it's been great so far.