Atharva Nayak (002322653) **Akshaj Raut (002337519)**

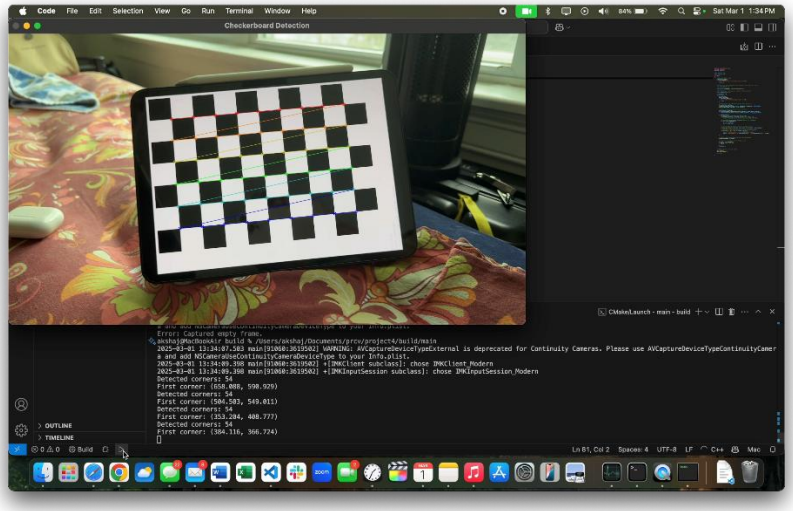
Project 4: Calibration and Augmented Reality

This project focuses on bridging the gap between the physical and digital worlds through camera calibration and augmented reality. The goal is to accurately detect a calibration target—such as a printed or displayed checkerboard pattern—in a live video stream and then use that information to determine both the camera's internal characteristics and its position in space. By establishing a known correspondence between specific points in the target and their real-world positions, the system can effectively estimate how the camera sees the world.

Once calibrated, the system computes the camera's location and orientation relative to the target. This information allows it to overlay a virtual 3D object onto the scene, ensuring that the object remains correctly aligned even as either the camera or the target moves. Additionally, the project investigates the extraction of robust features from the scene, which can be used to enhance the stability and accuracy of the augmented reality experience. Overall, this project provides a practical introduction to the techniques that make dynamic, interactive digital overlays possible in real-world environments.

Task 1: Detect and Extract Target Corners.

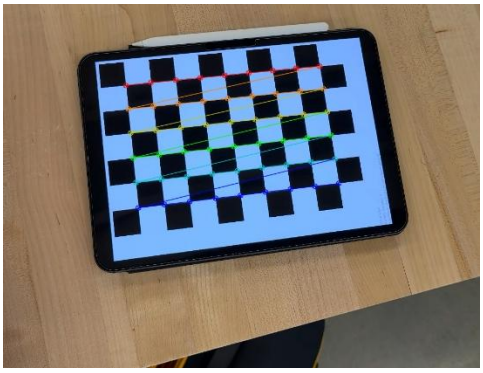
The first step in this project is detecting and extracting target corners from a reference checkerboard pattern using OpenCV functions such as findChessboardCorners. The system captures a live video stream, converts frames to grayscale using cvtColor, and applies corner detection algorithms to locate the internal intersections of the checkerboard. The detected corners are then refined using cornerSubPix to achieve sub-pixel accuracy. Once detected, these points are visualized using drawChessboardCorners and stored for later calibration. The detection process must handle challenges such as lighting variations, motion blur, and perspective distortions, which can affect accuracy. Checkerboards require high-contrast edges and proper alignment to ensure reliable detection. This step ensures accurate corner extraction, forming the foundation for intrinsic camera calibration and real-time pose estimation for augmented reality applications.

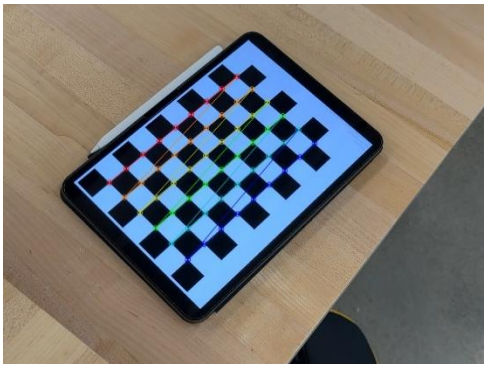


The target used in the project is checkered board and the challenges or limitations and challenges that we observed were difficulty in detecting corners due to light reflection, light variations and shadows.

Task 2: Select Calibration Image

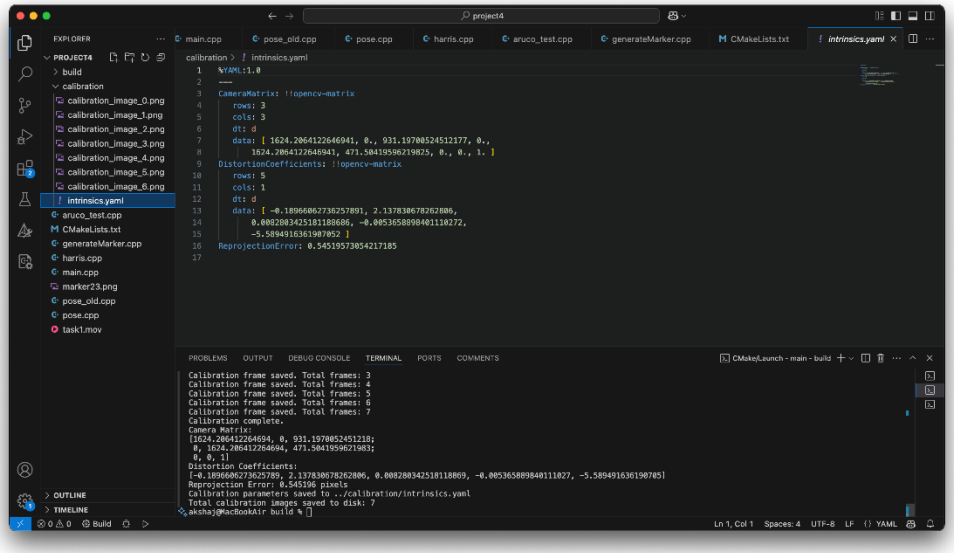
In this step, the system allows the user to save calibration images by pressing 's', storing the most recent checkerboard detection along with its corresponding 3D world points. The detected 2D corners are saved in corner_list, while the 3D world coordinates, stored in point_list, are defined relative to the checkerboard using a unit-square system, where the top-left corner is (0,0,0) and subsequent points increase in the x-direction and decrease in the y-direction. To ensure consistency, each set of 3D points matches the number of detected 2D corners, maintaining a structured mapping for calibration. Only valid detections are saved, preventing errors from incorrectly stored images. A common challenge at this stage is ensuring the correct row-column mapping in the 3D coordinate system. The stored calibration images, with highlighted corners, serve as verification for accurate data collection, forming the foundation for precise camera calibration in the next phase.





Task 3: Calibrate the Camera

In this step, the system performs camera calibration after the user saves at least five valid calibration images by pressing 'c'. The calibrateCamera function is used to compute the camera's intrinsic parameters, taking in the stored point_list (3D world points) and corner_list (2D image points). The camera_matrix is initialized as a 3x3 matrix with the principal point set to the image center, and distortion_coefficients are set to zero initially. The calibration process estimates these parameters and outputs the final re-projection error, which is displayed in the console. If calibration is successful, the computed camera matrix, distortion coefficients, and error values are printed. The system also allows the user to save the intrinsic parameters to a file by pressing 'w'. This step ensures that the camera's internal properties are determined, allowing for accurate 3D-to-2D projections in later tasks.

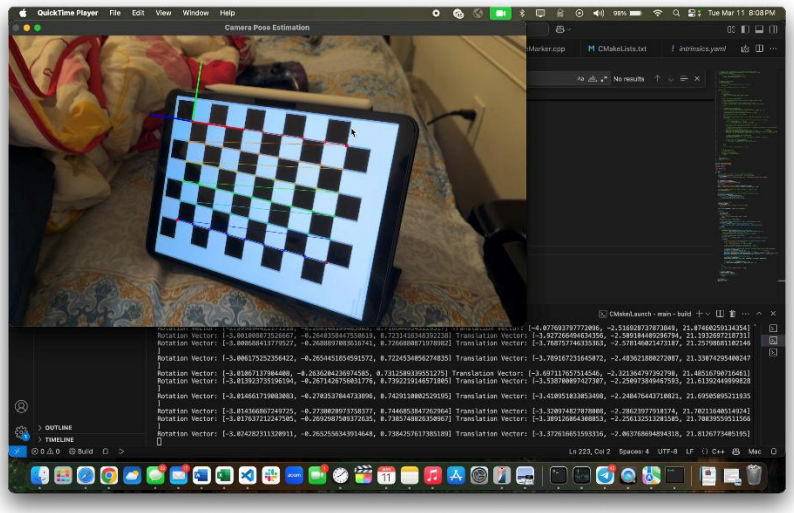


Task 4: Calculate the Current Position of the Camera

The system reads the previously saved camera calibration parameters from a file and starts a video stream to detect the checkerboard target in each frame. If the pattern is found, the detected 2D corner points are used along with the predefined 3D world coordinates to estimate the camera's pose using solvePnP. The computed rotation and translation vectors are printed in real-time as the camera moves. Observing these values, moving the camera side-to-side changes the translation vector accordingly, while tilting or rotating the camera affects the rotation vector. The results align with expected behavior, confirming that the pose estimation accurately tracks the camera's movement relative to the target. A video of demonstration of the changes in the values is recorded and the link to it will be in the README file.

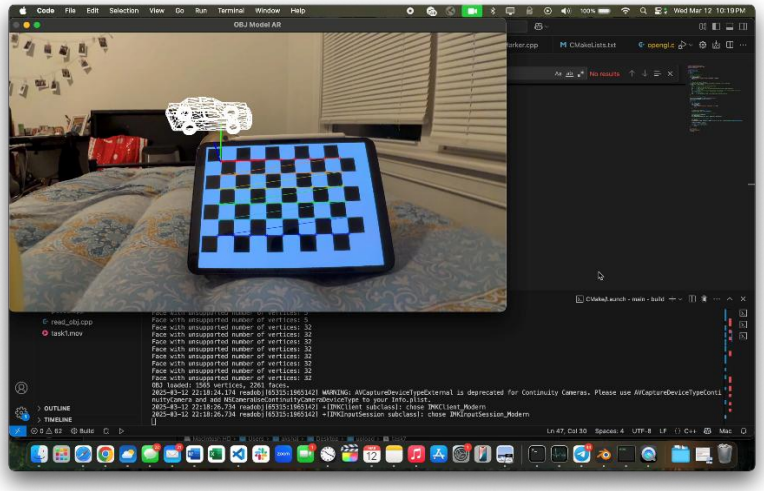
Task 5: Project Outside the Corners or 3D Axes

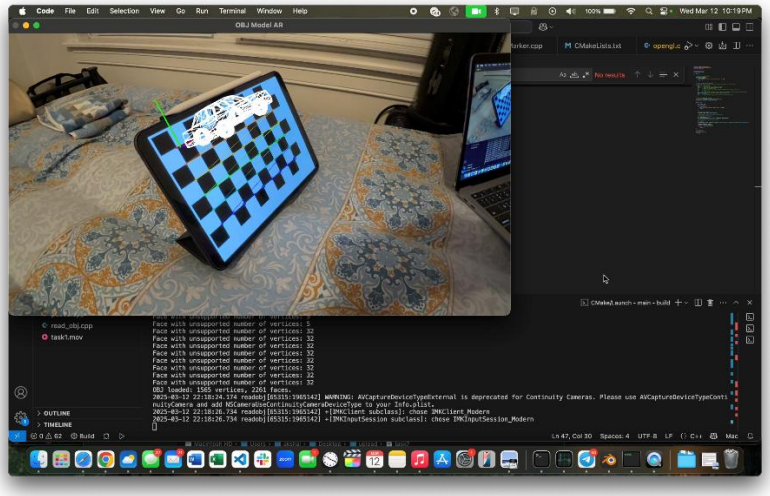
In this step, the system utilizes the previously estimated camera pose to project 3D points onto the image plane in real-time using the projectPoints function. The 3D coordinates of at least four checkerboard corners are transformed into 2D image coordinates based on the computed rotation and translation vectors. Additionally, a set of 3D coordinate axes is rendered at the target's origin, aiding in the verification of correct pose estimation. As the camera or target moves, the projected points and axes dynamically adjust, maintaining proper alignment with the detected checkerboard. Observing the output, the projected points consistently overlay the corresponding real-world locations, confirming the accuracy of the camera pose and projection process. This step validates the system's ability to map 3D objects into the real-world scene, laying the foundation for integrating virtual objects.

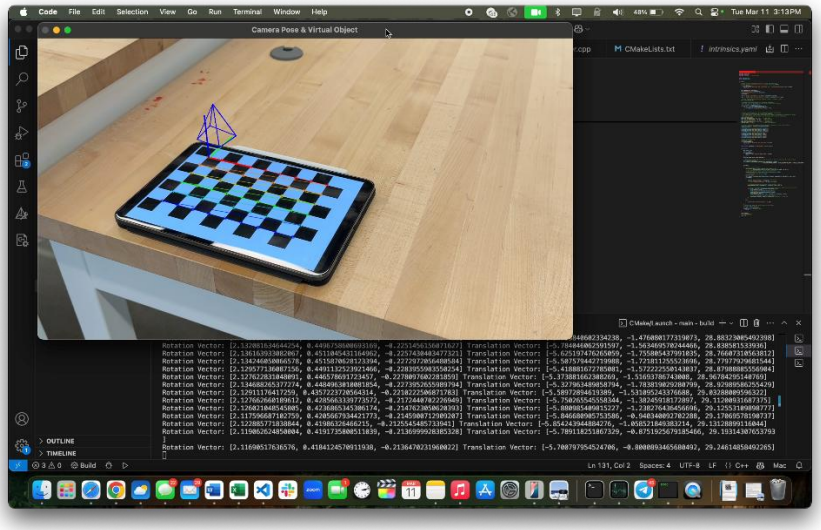


Task 6: Create Virtual Object

In this step, the system constructs a virtual 3D car model in world space and projects it onto the image using projectPoints. The car is defined as a set of 3D points representing key structural components such as the body, roof, and wheels, connected by lines to form a recognizable shape. Positioned above the checkerboard target, the car's 3D coordinates are transformed into 2D image coordinates based on the estimated camera pose. The projected points and lines are dynamically updated as the camera moves, ensuring that the car remains correctly oriented in the scene. The rendered car appears to "float" above the checkerboard, maintaining proper perspective and scale relative to the target. Observing the visualization, the car accurately adjusts to camera movement, confirming that the projection process is functioning correctly. This step successfully integrates a virtual 3D car into the real-world scene, demonstrating real-time augmented reality rendering.

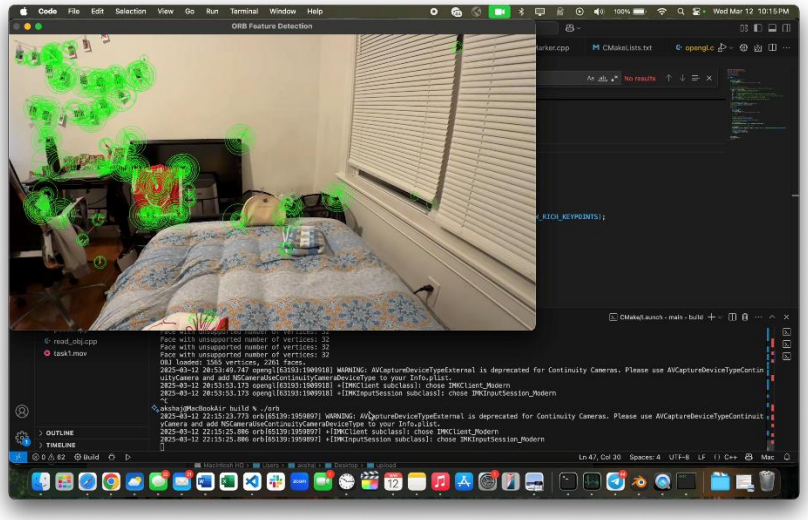






Task 7: Detect Robust Features

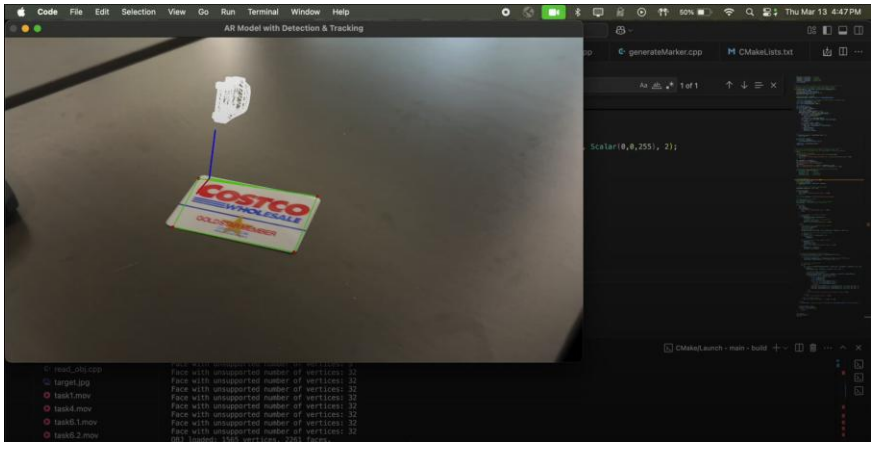
The system employs ORB to detect and track key feature points in a live video stream. It captures frames from the camera using VideoCapture, converts them to grayscale with cvtColor, and processes them using orb->detectAndCompute to extract keypoints and compute their descriptors. The detected keypoints are then visualized on the frame using drawKeypoints, where each point represents a distinctive feature in the scene. The system continuously updates and displays the detected features in real time within a window created using namedWindow and imshow. By adjusting ORB parameters such as the number of features, scale factor, and threshold values, the system optimizes feature detection across different lighting and motion conditions. The extracted keypoints remain stable across frames, making them suitable for augmented reality applications where they can serve as anchor points for tracking virtual objects. The program runs in a loop until the user exits by pressing the ESC key, ensuring continuous real-time feature extraction and visualization.



Extensions

AR System Working of Other Targets

In this extended version of the project, the AR system was adapted to track a custom target—a Costco membership card—using ORB feature detection instead of a checkerboard. The system captures an image of the target, extracts keypoints and descriptors, and stores them for real-time matching against live frames using a Brute Force Matcher (BFMatcher). Once the target is detected, at least four corresponding points are used to compute a homography transformation, aligning the detected object with the reference model. Using solvePnP, the system estimates the target's 3D pose, allowing a virtual 3D car model to be projected onto the scene with projectPoints. As the camera moves, the AR object dynamically updates its position and orientation, ensuring proper alignment and perspective. This implementation demonstrates how augmented reality can function with real-world objects by leveraging feature detection and pose estimation, enabling accurate virtual object placement and interaction in dynamic environments.



Atharva Nayak (002322653) **Akshaj Raut (002337519)**

Reflection

This project provided hands-on experience in camera calibration, pose estimation, and augmented reality, demonstrating how computer vision techniques can be used for real-time virtual object integration. The process began with detecting a checkerboard pattern to extract key points, which were then used for camera calibration to correct distortions and determine intrinsic parameters. This enabled accurate pose estimation, allowing the system to track the camera's position relative to the target and project 3D objects accordingly. A virtual car was successfully placed in the scene, maintaining its correct orientation and position as the camera moved. Additionally, ORB feature detection was implemented to identify stable key points, which could serve as anchors for augmented reality applications. Throughout the project, challenges such as lighting variations, motion blur, and perspective distortions highlighted the importance of fine-tuning detection and calibration settings. Overall, the project successfully integrated multiple computer vision techniques, showcasing their potential for real-world applications such as object tracking, augmented reality, and interactive digital environments.

Acknowledgements

We would like to acknowledge that this project was a collaborative effort between Atharva Nayak and Akshaj Raut, as we worked together to research, develop, and document our work. Through teamwork and problem-solving, we successfully overcame challenges and gained a deeper understanding of calibration and use of augmented reality. We also appreciate the resources provided by the OpenCV and OpenGL communities, which helped us implement our tasks. We would also like to thank the developers and researchers working on ORB feature detection for making it accessible for practical applications.