Report 4 – Calibration and Augmented Reality

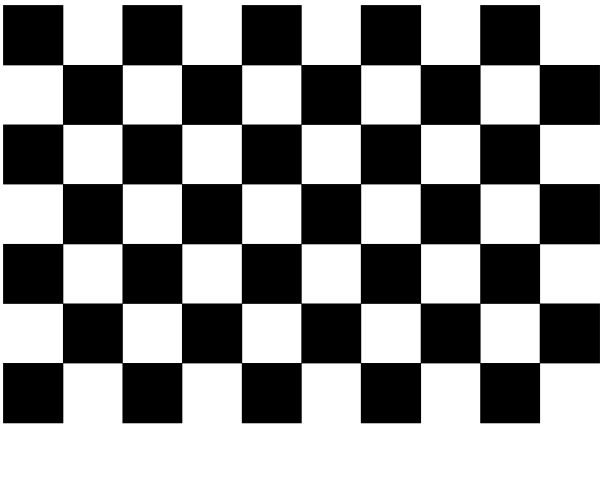
@Yanting Lai

Description

This program creates an augmented reality system using a checkerboard pattern and OpenCV. It operates in three phases: first, it captures multiple views of a 9x6 checkerboard pattern for calibration; second, it uses these images to calculate the camera's internal parameters; finally, it demonstrates AR capabilities by overlaying a virtual pyramid on the checkerboard. The pyramid maintains correct perspective as either the camera or checkerboard moves, demonstrating successful real-time pose estimation. The system provides a practical implementation of camera calibration and basic augmented reality concepts.

Task 1: Detect and Extract Target Corners

> Camera calibration begins with detecting and extracting corners from a calibration target. This project uses a checkerboard pattern with 9x6 internal corners as the calibration target. OpenCV's findChessboardCorners function is utilized to detect and refine corner positions with sub-pixel accuracy. The consistent geometric structure of the checkerboard pattern provides reliable feature points, though its symmetrical nature means we need to pay attention to orientation when establishing the coordinate system.



Target type: 9 x 6 checkerboard pattern for camera calibration and pose estimation
Challenges:

- **Viewing angle sensitivity:** The system struggles to detect corners when the checkerboard is at steep angles.
- Lighting conditions: Strong reflections or shadows on the board surface can cause detection failures; Performance degrades in low-light conditions.
- Motion Handling: It requires relatively stable positioning for reliable corner extraction.

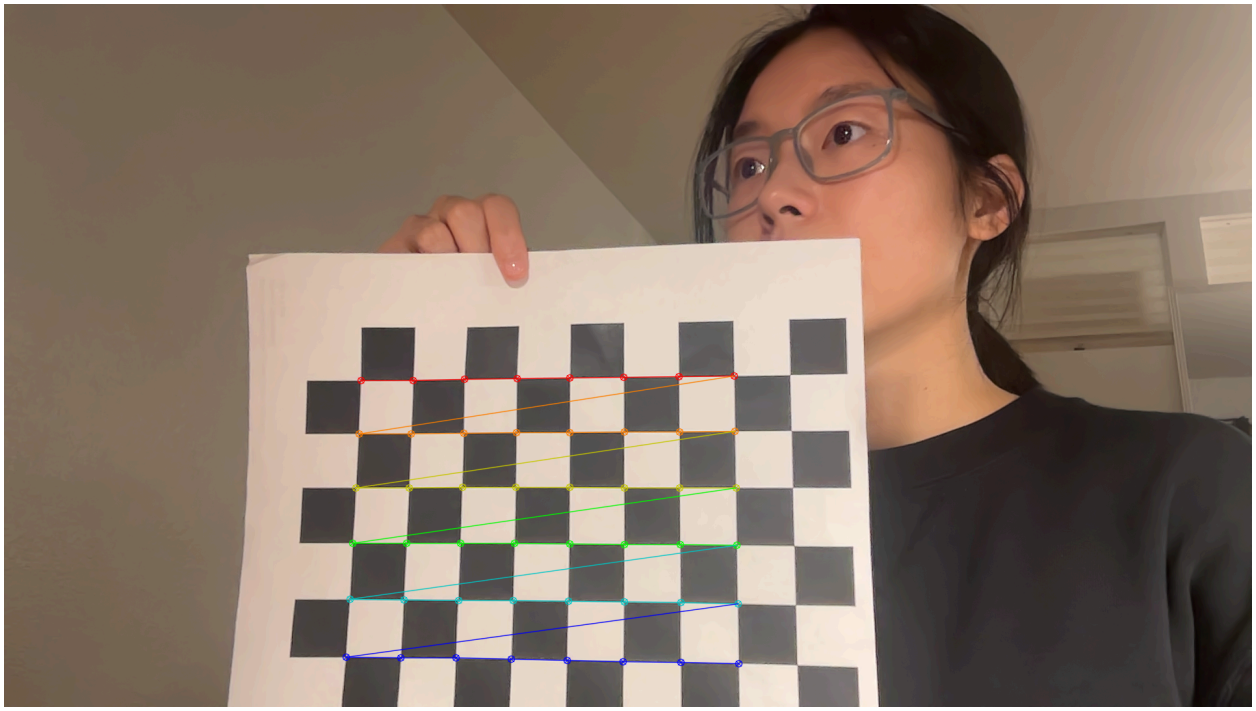
Task 2: Select Calibration Images

> I capture multiple views of the checkerboard target and store both their 2D image coordinates and corresponding 3D world coordinates. This builds my robust dataset for calibration by collecting at least 5 different perspectives of the target, ensuring comprehensive camera parameter estimation.

Summary of frames being used for calibration:

```
Parameter,Value
Number of frames,11
Board width,8
Board height,6
Points per frame,48
Total points,528
```

Example of a frame used for calibration:



Task 3: Calibrate the Camera

> I compute the camera's intrinsic matrix and distortion coefficients using OpenCV's calibrateCamera function. My system minimizes reprojection error to generate accurate camera parameters, aiming for sub-pixel accuracy in the calibration results.

```
Calibration complete!
RMS error: 0.45
Camera matrix:
[1092.284991863491, 0, 1001.364578528664;
 0, 1091.758738196825, 561.4434685189126;
 0, 0, 1]
Distortion coefficients:
[-0.046080077948421 32;
 0.4854021262255315;
 -0.009106869639857 27;
 -0.000127066574194 2877;
 -0.6688373059364306]
```

Task 4: Calculate Current Position of the Camera

> I track the camera's real-time position and orientation relative to the target using solvePnP. This transforms my 2D-3D point correspondences into precise camera pose information, enabling accurate spatial tracking for AR applications.

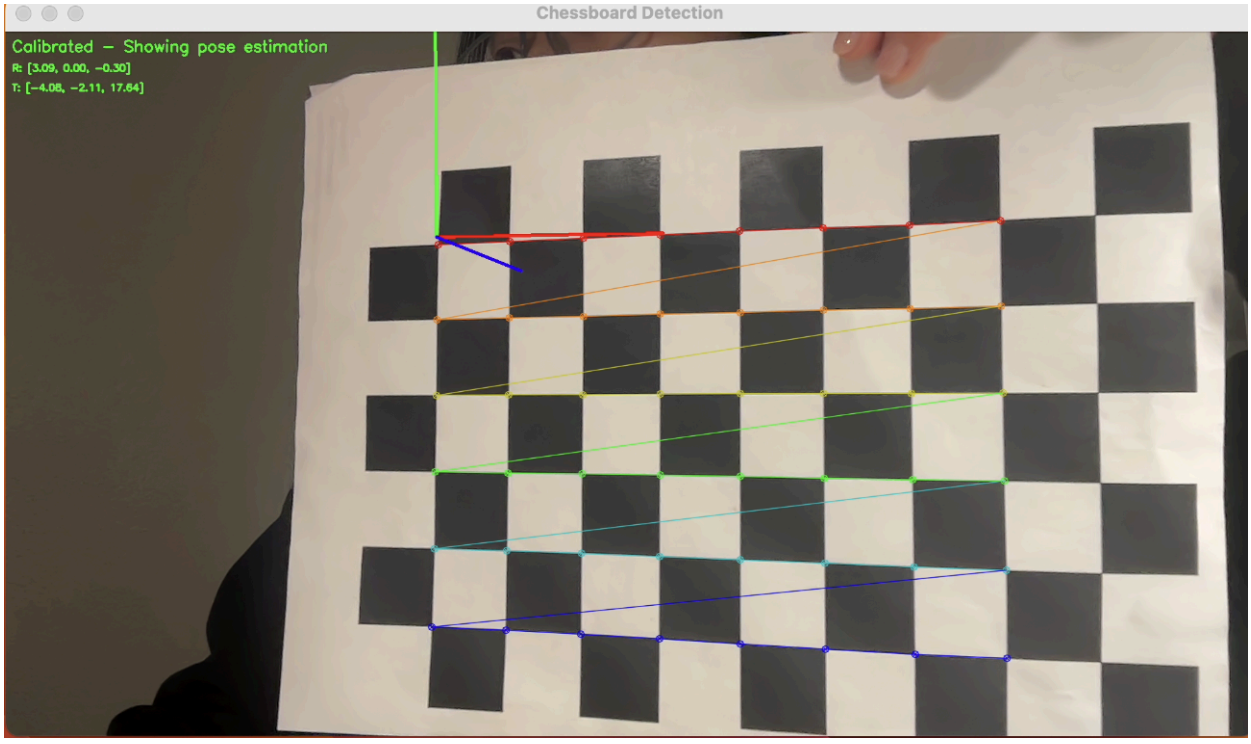
Let’s use three examples of rotation vector and a translation vector to explain how these values changes as the camera moves side to side:

Rotation vector: [3.00, -0.02, -0.16] Translation vector: [-5.53, 0.19, 13.49]
Rotation vector: [3.01, -0.01, -0.14] Translation vector: [-5.54, 0.22, 13.43]
Rotation vector: [3.03, -0.04, -0.20] Translation vector: [-6.33, 1.12, 14.27]

The pose estimation values demonstrate logical changes when moving the camera side to side. In the translation vector, we see significant changes in the x-component (-5.53 to -6.33) reflecting lateral movement, while the z-component remains relatively stable (around 13-14 units) as the distance to the board stays consistent. The rotation vector shows minimal changes (mainly around x-axis at ~3.0 radians) since the camera's orientation remains mostly constant during lateral movement. These values align with the expected behavior of a camera moving sideways relative to the checkerboard.

Task 5:Project Outside Corners or 3D Axes

> I validate my calibration accuracy by projecting 3D reference points onto the image plane. Drawing 3D axes on the target demonstrates my system's ability to maintain correct spatial relationships during camera movement.



Task 6:Create a Virtual Object

> I design and render a 3D virtual object (a pyramid) above the target board. My system projects this object into the video feed, maintaining proper perspective and orientation as the camera moves around the scene, demonstrating practical AR capabilities.

I implemented a virtual pyramid floating above the checkerboard pattern. The object consists of a square base with sides of 2 units (scaled) and rises to an apex 2 units above the base. The pyramid's edges are color coded blue for the base and green for lines connecting to the apex to aid visualization. As shown in the video below, the virtual object maintains proper perspective and position relative to the checkerboard during camera movement**.**

Video demo:
https://drive.google.com/file/d/14xmNXshD4a7BVUosNGi_4qW2CmNQ-dSB/view?usp=sharing

Task 7: Detect Robust Features

> I implement feature detection using Harris corners or SURF features to identify distinctive points in natural scenes. This explores alternative tracking methods that could enhance AR applications through more flexible and robust feature matching.

I implemented **Harris Corner detection** as my feature detection method. My system provides three adjustable parameters for users: **Threshold, Block Size, and Kernel Size**.

Video demo harris corner:

https://drive.google.com/file/d/1EbaAiFpkKOE6GMyr8f4yY9y2jA74h4HI/view?usp=sharing

When adjusting these parameters:

Threshold controls detection sensitivity - higher values detect fewer but stronger corners, while lower values detect more corners but reduce performance.

Block Size affects detection area - smaller sizes capture finer details with more noise, larger sizes reduce noise but miss details.

Kernel Size influences smoothing - smaller sizes preserve details but include noise, larger sizes reduce noise but blur details.

Harris corners could be used for AR by first capturing reference feature points of a target object and establishing their 3D positions. During real-time operation, newly detected corners would be matched with these reference points to compute camera pose using solvePnP. This would allow virtual object projection without requiring artificial patterns like checkerboards, though it would need robust feature matching algorithms to maintain stability.

Extension completed

1.Enable the system to use static images or pre-captured video sequences with targets and demonstrate inserting virtual objects into the scenes

Reflection

1.This assignment is relatively more straightforward than the previous assignments since we can use the opencv functionalities instead of writing our own.
2.The challenging part is with the extension, and I think overall the project breaks down the process of how to create a virtual object in a very simple and easy to understand way.
3.The setup of the camera took some time. I tried using the computer camera as well as using my iphone camera as the input and mapping that to the macbook. The second method would work better since that will ensure my object is not moving while only the camera is moving. If it is the other way around, it would be harder to detect etc. the chessboard corner.

Reference

[1] OpenCV Documentation. "Camera Calibration and 3D Reconstruction." Used for: findChessboardCorners, calibrateCamera, solvePnP, drawChessboardCorners implementation. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[2] OpenCV Documentation. "Harris Corner Detection." Used for: Feature detection implementation and parameter tuning.
https://docs.opencv.org/4.x/d4/d7d/tutorial_harris_detector.html

[3] OpenCV-Python Tutorials. "Camera Calibration and 3D Reconstruction." Used for: Understanding calibration workflow and virtual object projection.
https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html