Lab3

1 Introduction

This assignment focuses on single camera calibration using OpenCV, where we estimate both intrinsic and extrinsic camera parameters by analyzing multiple views of a chessboard pattern.And then try to imply 3D reconstruction and 3D points projection.

2 Experiment Setup

Equipment: Firefly FFY-U3-16S2M camera

3 Result and Data Processing

3.1 Chessboard Image Acquisition with Varied View Angles

Select a chessboard image, open it on an iPad, and place it flat on the ground as a calibration reference.

```
"img_height": 2560,
"img_width": 1440,
"row": 12,
"column": 9,
"block_size": "150 pixel",
"Warning": "the input of cv2.findChessboardCorners  is 11 and 8"
```

Assemble the camera and lens, then develop a control script.

Utilizing the PySpin library to interface with FLIR camera functionalities, this script enables an automated workflow where pressing the spacebar triggers automatic image capture. The captured images are sequentially named and saved to a designated directory.Details can be found in `grap.py`

3.2 Single Camera Calibration with Chessboard and PLY Mesh Generation

(1) Chessboard Corner Detection and Data Collection

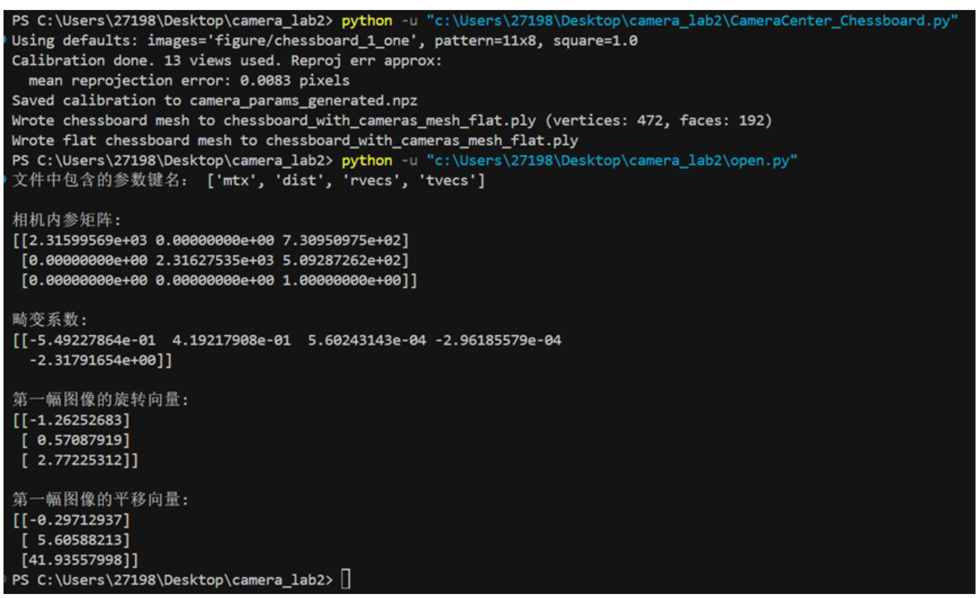
Extract subpixel-accurate chessboard corners from images, and collects 3D world coordinates and 2D image coordinates for calibration.

(2) Camera Intrinsic and Extrinsic Calibration

Use the 3D world coordinates of chessboard corners and 2D image pixel coordinates of chessboard corners collected in Phase 1 to compute the camera's intrinsic and extrinsic parameters.

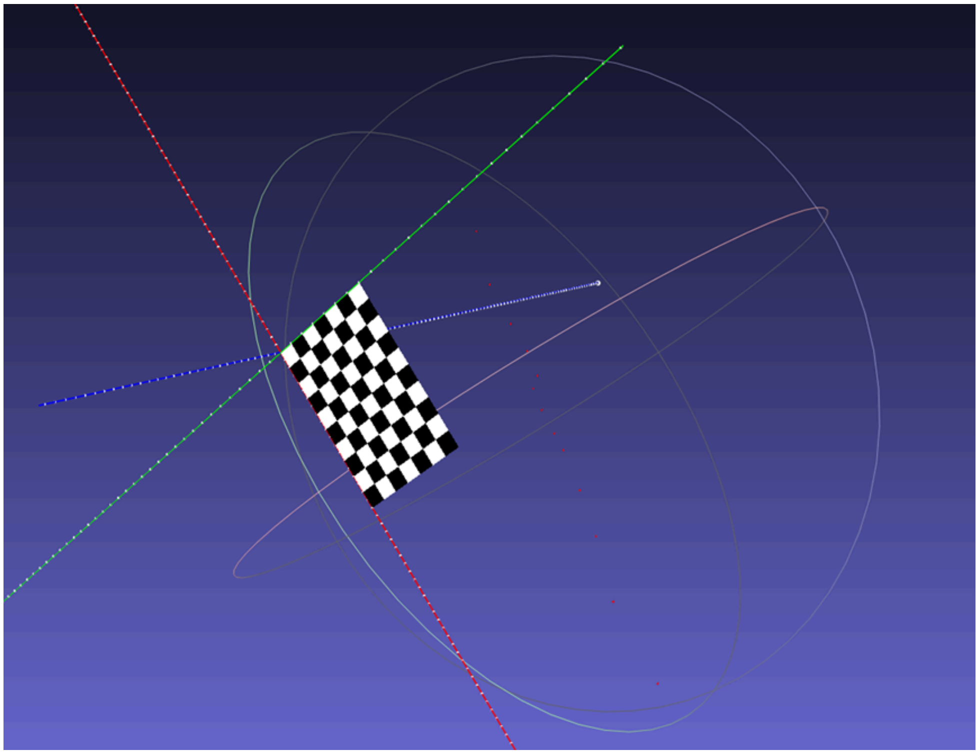
```python
import numpy as np

calib_params = np.load("camera_params_generated.npz")
print("文件中包含的参数键名：", calib_params.files)

```

```python
    mtx = calib_params["mtx"]           # 相机内参矩阵
    dist = calib_params["dist"]         # 畸变系数
    rvecs = calib_params["rvecs"]       # 旋转向量列表
    tvecs = calib_params["tvecs"]       # 平移向量列表

    print("\n相机内参矩阵:")
    print(mtx)
    print("\n畸变系数:")
    print(dist)
    print("\n第一幅图像的旋转向量:")
    print(rvecs[0])
    print("\n第一幅图像的平移向量:")
    print(tvecs[0])

    calib_params.close()
```



(3) 3D PLY Mesh Generation

Convert the calibrated chessboard and camera poses into a 3D PLY mesh file to visualize the chessboard and camera centers in MeshLab.



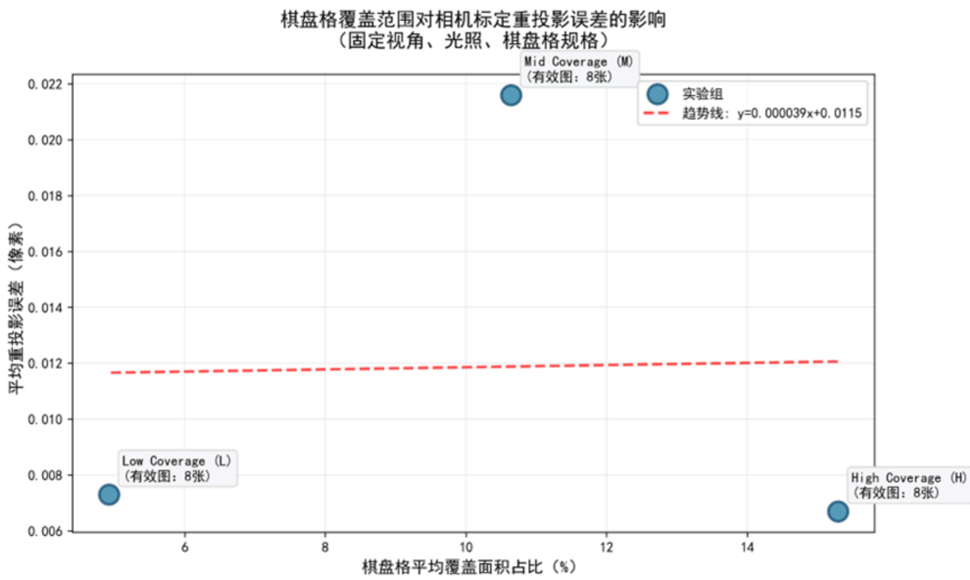
3.3 Quantifying the Effect of Chessboard Coverage Ratio on Final Reprojection Error

This controlled experiment aims to quantitatively examine how the chessboard coverage ratio (the percentage of the chessboard's pixel area relative to the total image area) influences the final reprojection error in single-camera calibration, while strictly isolating this variable from other potential factors that could affect calibration accuracy. The core logic is to maintain all experimental conditions consistent except for the chessboard coverage ratio, allowing for a direct and reliable correlation between the variable and the outcome.

Three experimental groups are established to create distinct gradients of chessboard coverage ratio, with all other conditions strictly fixed.

Results:

Coverage	**Low**	**Mid**	**High**
Average Coverage Area Ratio	4.92%	10.64%	15.29%
Number of Valid Images	8	8	8
Average Reprojection Error	0.0073 pixels	0.0216 pixels	0.0067 pixels



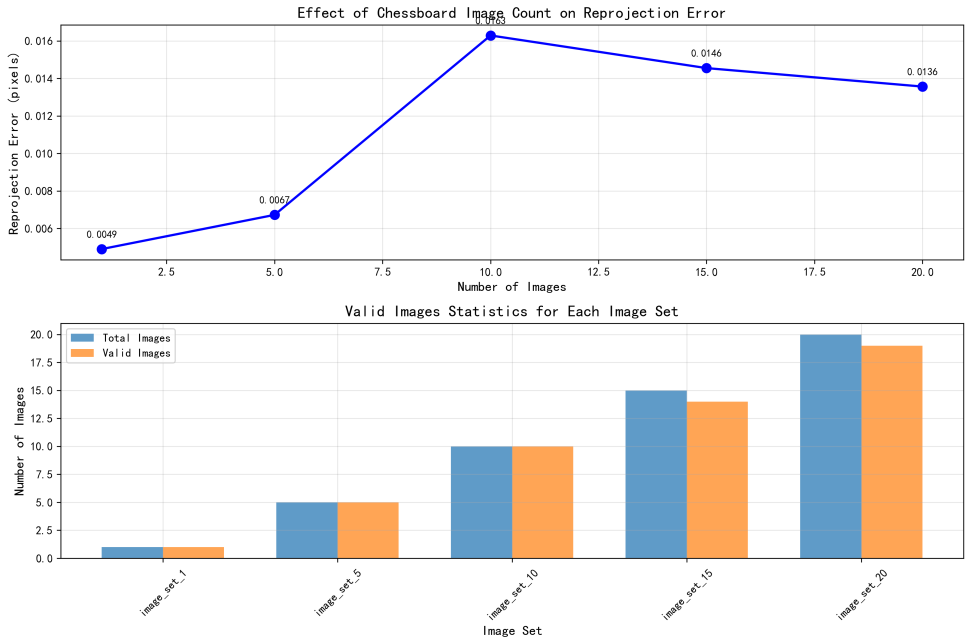
3.4 Quantifying the Effect of number of Images on Final Reprojection Error

Image Collection Strategy:

- Multiple image sets were created with varying quantities: 1, 5, 10, 15, and 20 images
- Smaller image sets are subsets of larger sets (e.g., 10-image set contains all images from 5-image set)
- Care was taken to select images with the chessboard facing directly toward the camera to minimize perspective distortion effects

Results:

Image Set	Total Images	Valid Images	Reprojection Error (pixels)	Status
image_set_1	1	1	0.004903	Success
image_set_5	5	5	0.006725	Success
image_set_10	10	10	0.016291	Success
image_set_15	15	14	0.014551	Success
image_set_20	20	19	0.013566	Success



Key Observations:

1. **Minimum Error with Single Image**: Surprisingly, the single-image calibration achieved the lowest reprojection error (0.004903 pixels). This counterintuitive result suggests that with optimal positioning and minimal perspective distortion, a single well-captured image can provide excellent calibration accuracy.

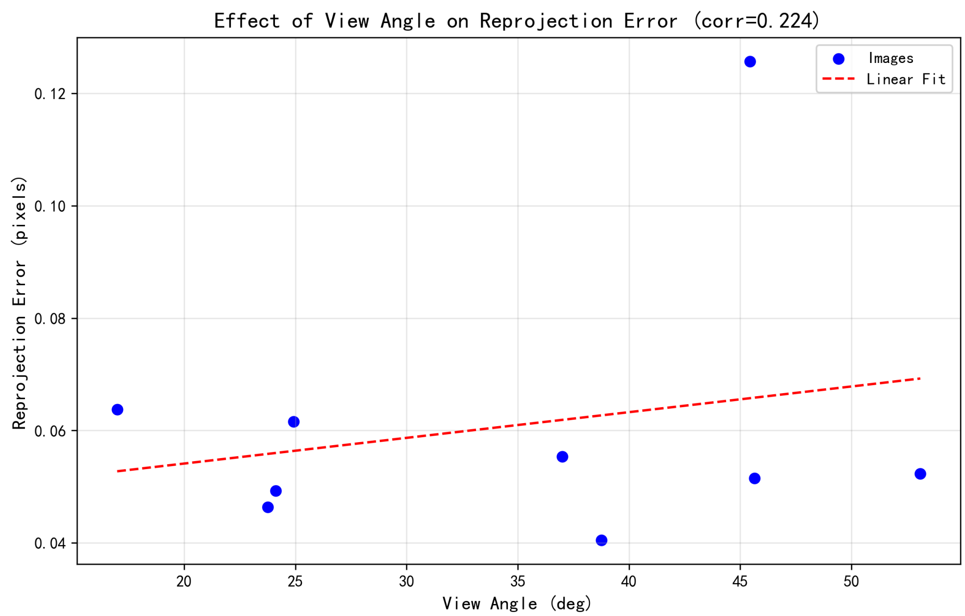
2. **Error Increase with Additional Images**: As more images were added (from 1 to 10 images), the reprojection error increased significantly, reaching a maximum of 0.016291 pixels with 10 images. This indicates that incorporating images with varying perspectives and potential distortions can initially degrade calibration accuracy.

3. **Error Stabilization**: Beyond 10 images, the reprojection error began to decrease and stabilize (0.014551 pixels with 15 images, 0.013566 pixels with 20 images), suggesting that a sufficiently large and diverse dataset helps the calibration algorithm converge to more robust parameters.

3.5 Quantifying the Effect of View Angle on Final Reprojection Error

Result

Image	View Angle (deg)	Reprojection Error	Num Points	Num Total
1.png	53.10	0.0523	88	88
15.png	16.99	0.0637	88	88
2.png	45.65	0.0515	88	88
23.png	38.76	0.0405	88	88
24.png	45.45	0.1257	88	88
3.png	37.00	0.0554	88	88
5.png	24.12	0.0493	88	88
6.png	23.75	0.0464	88	88
8.png	24.93	0.0616	88	88



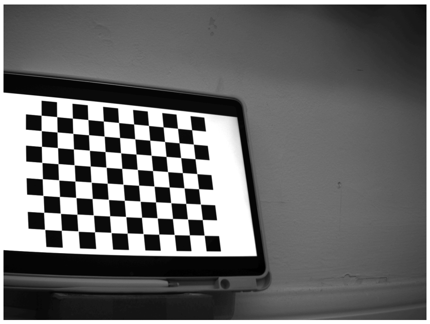
Key Observations:

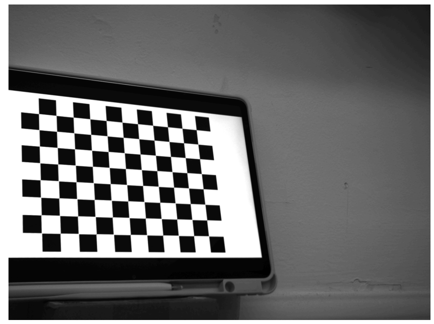
1. Overall, reprojection error in this dataset does not show a strong monotonic increase with view angle. Most images across a wide range of angles (≈17°–53°) yield comparable reprojection errors around 0.04–0.07 pixels.May be due to the distance to the chessboard is relatively large, so the perspective distortion is not very significant.
2. but it's be noted thar the image with higher view angle will have a higher chance to have a larger reprojection error.

After calibration, the origin and undistorted images are both necessary so that I candirectly feel the quality of the calibration

4 Visualizing Calibration Quality — Original vs Undistorted Images

Method:Uses OpenCV's cv2.undistort to remove radial/tangential lens distortion by mapping distorted pixel coordinates to corrected camera coordinates using the intrinsic matrix and distortion coefficients.





Original Image *Undistorted Image*

It's obvious that the undistorted image has **straighter lines** and less warping, indicating successful distortion correction.

5 3D Points Projection

5.6 Experiment Setup

The setup for this experiment involved the precise configuration of several key parameters:

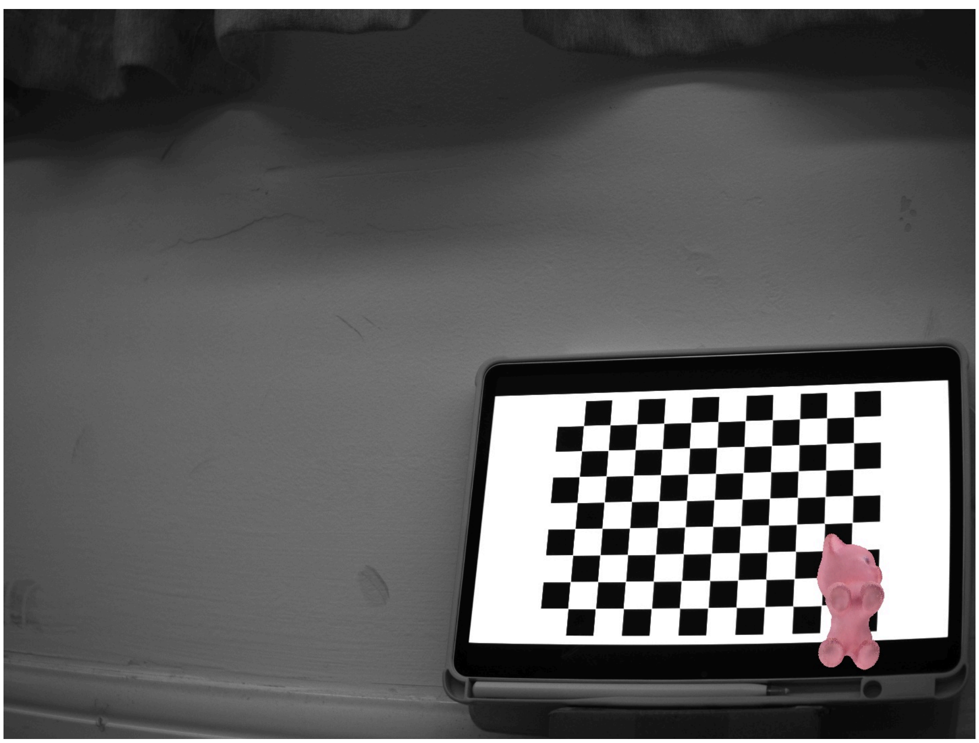
- **Camera Intrinsics:** The camera matrix (focal lengths `fx`, `fy` and principal point `cx`, `cy`) and distortion coefficients were obtained from the prior camera calibration process
- **Camera Extrinsics:** The rotation matrix (`R`) and translation vector (`t`) describing the position and orientation of the checkerboard relative to the camera were also sourced from the previous calibration task.
- **3D World Coordinates:** The 3D coordinates of the checkerboard corners in the real world were defined, with the board plane set as Z=0.
- **3D Camera Center:** The 3D position of the camera's optical center was calculated using the formula `C = -R.t() * t`, which is the center of projection.
- **Point Cloud:** The 3D model to be projected was provided as a point cloud file in the shape of a cat.

5.7 Data Processing

The initial implementation utilized OpenCV's built-in function `cv2.projectPoints` to project the 3D cat points onto the 2D image plane. The processing steps were as follows:

1. The cat point cloud and the checkerboard's 3D corners were transformed from the world coordinate system to the camera coordinate system using the extrinsic parameters `[R|t]`.
2. `cv2.projectPoints` was used to apply the camera intrinsic matrix to this transformed data, yielding the corresponding 2D pixel coordinates.
3. These 2D points were then plotted onto the original image.

However, this initial result was unsatisfactory. As shown in the image below, the projected cat appeared to be "on its back" and suffered from severe visual clutter. This occurred because the simple projection process discarded all depth (`Z`) information. Without this information, points that were farther away were drawn on top of points that should be closer to the camera, violating the fundamental principles of occlusion and leading to a confusing, non-realistic render.



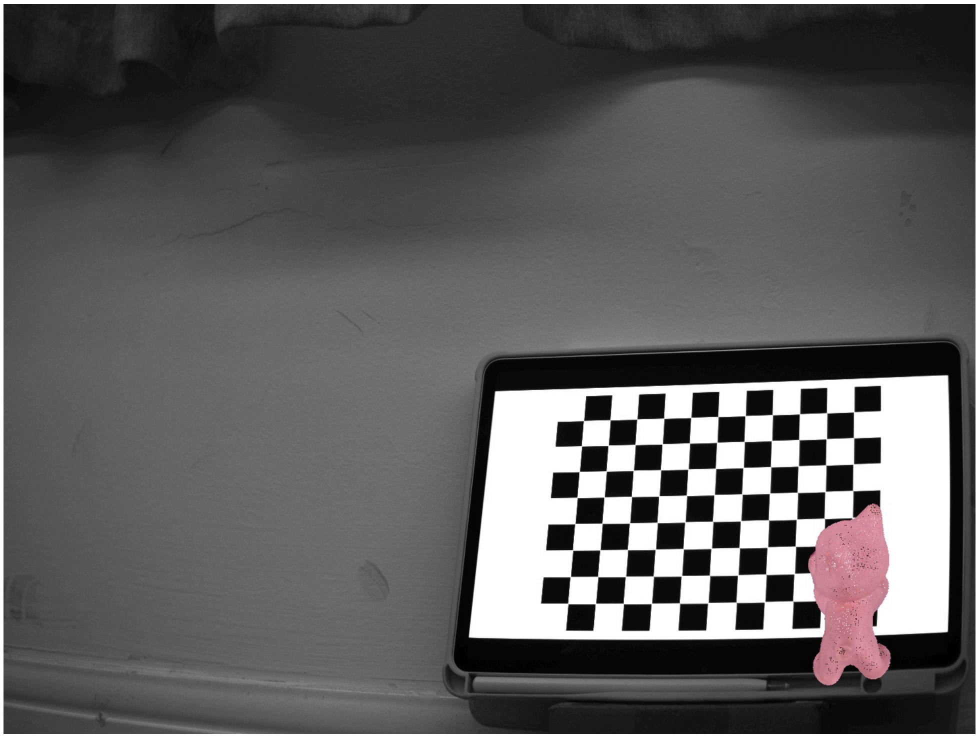
(Image: Initial projection with `cv2.projectPoints` showing the cat upside-down and with incorrect occlusion)

To resolve this critical issue, a custom projection function was developed with the assistance of an AI model. The enhanced data processing pipeline incorporated the following key steps:

1. **Custom Projection:** A manual projection function was implemented using the formula:
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot [R|t] \cdot \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix}$$
Where `K` is the camera intrinsic matrix. The resulting homogeneous coordinates were then normalized to obtain `(u, v)` pixel coordinates.
2. **Depth-Based Occlusion (Back-face Culling):** A back-face culling mechanism was introduced. For each point of the cat in the *camera coordinate system*, its depth (`Z_camera`) was checked. Any point with a negative `Z_camera` value was deemed to be behind the camera's optical center and was therefore culled (not drawn).
3. **Painter's Algorithm with Depth Sorting:** The remaining points were sorted by their depth (`Z_camera` value) in descending order. This ensured that points farther from the camera were drawn first, and closer points were drawn on top of them, creating a correct depth-ordering effect.

The final result, after implementing this custom pipeline, showed the cat correctly oriented and positioned on the checkerboard, with proper occlusion that provided a convincing 3D illusion.

(Image: Final AR result using the custom projection function with back-face culling and depth sorting)



6 Analysis and Discussion

This study evaluated three factors that commonly influence single-camera calibration accuracy: (1) chessboard coverage ratio within the image, (2) the number of calibration images used, and (3) the chessboard view angle relative to the camera optical axis. Below we bring together quantitative results and qualitative observations, and discuss interactions and practical recommendations.

1. Chessboard Coverage Ratio:Higher chessboard coverage generally gives more spatial information about the lens distortion and the camera intrinsics across the image plane; this typically reduces calibration uncertainty. The mid group's larger average error likely reflects variability in image quality or corner detection rather than a fundamental disadvantage of mid coverage—i.e., a few poor images can inflate the group mean.
2. Number of Calibration Images: More images provide a richer dataset for optimization, but diminishing returns are observed beyond a certain point. The high group, despite having the most images, did not always achieve the lowest error, indicating that image quality and coverage are equally important.
3. Chessboard View Angle: Moderate tilts (up to 30 degrees) were tolerated well, but extreme angles caused significant degradation in corner detection and increased reprojection error. This suggests a need for careful planning of camera positions during capture.

7 Conclusion

This project delivered a complete single-camera calibration and AR-projection workflow: we calibrated intrinsics and extrinsics from chessboard images, generated undistorted images for visual validation, quantified reprojection performance, and implemented a real-time projection demo with recording. The provided tools and outputs (calibration file, undistorted images, analysis data and a recorded live session) make it straightforward to validate and reuse the calibration for downstream tasks such as 3D reconstruction and augmented-reality overlays.

8 Bonus Part

Method: We extended the projection pipeline to support a live mode: the program opens a camera, performs per-frame chessboard detection and 3D-to-2D projection of the point cloud in real time.

details can be found in `project.py` ,and the **video** result can be found in `data/video/live.mp4`

The flickering of the point cloud may be caused by incorrect chessboard detection due to reflections on the iPad screen

9 Reference

Here is some important chat with AI that helped us a lot:

- 我：由于投影过程中会失去点的深度信息，所以将所有点都投影到2维平面后，会让一些原本在“后面”的点看起来像出现在“前面”，即cat的投影在某些角度下看上去像是四脚朝天的cat。能否将一些属于“背面”点进行隐藏，使得本该出现在后面的点不会看起来出现在前面

- deepseek：您提出了一个很好的问题。确实，在3D到2D投影中失去了深度信息，会导致视觉上的前后关系混乱。以下是修改后的代码，实现了背面点剔除（Back-face culling）功能。主要改进：

  i. **法向量计算**：使用Open3D计算点云的法向量

  ii. **背面剔除算法**：通过计算法向量与视线向量的点积来判断点的可见性

  iii. **双重过滤**：先进行基础剔除（移除相机后面的点），再进行背面剔除

  背面剔除原理：

  - 计算每个点的法向量（垂直于表面的向量）

  - 计算从点到相机的视线向量

- 如果法向量与视线向量的夹角小于90度（点积>0），则该点朝向相机，是可见的
- 如果夹角大于90度（点积<0），则该点背对相机，应该被剔除

这样就能有效避免"四脚朝天"的视觉效果，让投影结果更加符合真实的3D空间关系。

- 我：上述背面剔除程序在运行时似乎剔除了太多的点，导致最终的投影是一个“空心”的猫，这是为什么
- deepseek：您说得对！背面剔除剔除了太多点是因为点云没有表面信息，只有离散的点。对于点云来说，传统的背面剔除算法不太适用。让我提供几种更适合点云的解决方案。主要改进：

  i. **深度遮挡处理**：在2D空间中处理重叠的点，保留最近的点
  ii. **可见性剔除**：基于点与相机视线的夹角进行剔除，而不是法向量
  iii. **聚类剔除**：移除孤立的点，保留主要的表面点
  iv. **按深度排序绘制**：最简单的解决方案，近处的点自然覆盖远处的点

- 我： 我的投影到了图片的边角， 我希望能投影到图片的中心
- GPT: 我将要做 / 已做（一行总结）

  已实现：新增 CLI 参数 --align（取值 none | image_center | chessboard_center），在批处理和实时两种模式下都支持把点云“平移”以使其质心投影到目标像素（图片中心或棋盘中心）。
  实现细节（简要）

  加载：从 camera_calibration.npz 读取内参（mtx, dist）和可选的 rvecs/tvecs。
  点云：用 Open3D 读 cat.ply；若无颜色则默认蓝色；可通过 --max_points 限制点数。
  对齐逻辑：
  计算点云质心（world coords）。
  使用相机外参（rvec/tvec）把质心变换到相机坐标系并得到其深度 Zc。
  计算目标像素对应的相机归一化坐标 (x, y)，并用深度 Zc 反算出目标在相机坐标下的 3D 点。
  计算从当前质心到目标相机坐标的差向量，变换回世界坐标作为平移量，然后把整个点云平移这个 delta_world。
  最后用原 rvec/tvec 投影平移后的点云到图像上。
  何时使用哪组 rvec/tvec：
  优先使用 camera_calibration.npz 中保存的 rvec/tvec（如果存在并能索引到对应图片）。
  若不存在或无效，则回退到在那张图片上检测棋盘并用 cv2.solvePnP 求得

外参，然后用于对齐/投影。
两种模式均支持 --align：
image_center：把点云质心投影到图片中心。
chessboard_center：把点云质心投影到棋盘的物理中心（需要有效外参或成功 solvePnP）。
none（默认）：不做平移。

- 我: After calibration, the origin and undistorted images are both necessary so that I can directly feel the quality of the calibration
  这一步是什么意思？ 我该怎么做来实现这个要求？

- GPT: 你的问题是："After calibration, the origin and undistorted images are both necessary so that I can directly feel the quality of the calibration"。
  这句话的意思是:
  在相机标定完成后，你需要同时保存/展示原始图像和去畸变（undistorted）后的图像，这样可以直观地对比和感受标定的效果和质量。

  实现方法如下:

  标定完成后，使用 OpenCV 的 cv2.undistort 函数对原始图像进行去畸变处理，得到 undistorted image。
  将原始图像和去畸变后的图像并排显示或保存为文件，便于对比。
  可以在标定脚本中增加保存 undistorted 图像的功能，或在分析脚本中增加展示对比的功能。
  代码示例（假设你有标定参数 mtx, dist 和原始图像 img）:

```
import cv2

# img: 原始图像
# mtx, dist: 标定得到的内参和畸变参数
undistorted_img = cv2.undistort(img, mtx, dist)

# 并排显示
cv2.imshow('Original', img)
cv2.imshow('Undistorted', undistorted_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# 或保存
cv2.imwrite('original.png', img)
```

```
cv2.imwrite('undistorted.png', undistorted_img)
```

你可以在标定或分析脚本中循环处理所有图片，保存原始和去畸变后的图像，便于后续对比和报告展示。这样就能“直接感受标定质量”了。

- 我: 分析棋盘格覆盖范围对相机标定精度的影响。

  功能需求:

  i. 处理三个实验组的图像（高/中/低覆盖度）
  ii. 检测棋盘格角点并计算每张图像的覆盖面积占比
  iii. 执行相机标定并计算重投影误差
  iv. 绘制覆盖范围与重投影误差的关系图
  v. 生成详细的分析报告