PROMPTS TO RECREATE ENTIRE CAMERA CALIBRATION CODEBASE (Model used: Opus Claude Code)

PROMPT 1: PROJECT STRUCTURE & REQUIREMENTS

Create a camera calibration project with the following structure:

- Use OpenCV for camera calibration from chessboard images

- Build an inline Gradio web UI that works in Google Colab and Jupyter notebooks

- Organize code into stateless helper classes using @staticmethod

- Required classes: IO (file operations), Board (chessboard model), Img (image utils), Calib (calibration), Overlay (axes drawing), Render (3D visualization)

- Save calibration results to calibration.json with K matrix, distortion coefficients, extrinsics, and metadata

- Create requirements.txt with: opencv-python-headless>=4.8, numpy>=1.24, matplotlib>=3.7, plotly>=5.20, gradio>=4.0, pillow>=9.0

PROMPT 2: CORE CALIBRATION UTILITIES (calibration_utils.py)

Create calibration_utils.py with these stateless helper classes:

1. IO class: file operations (ensure_dir, list_images, imread_rgb, imwrite_rgb, save_json, load_json)

2. Board class: generate 3D object points for chessboard pattern (inner corners × square size)

3. Img class: image processing (to_gray, find_corners with sub-pixel refinement)

4. Calib class:

- calibrate() method using cv2.calibrateCamera with proper validation

- Return dict with K, D, extrinsics, per_view_errors, valid_paths, metadata

- undistort() method for lens distortion correction

5. Overlay class: project 3D coordinate axes onto calibration images using cv2.projectPoints

6. Render class: create 3D camera pose plots using both Plotly (interactive) and Matplotlib (static)

Include comprehensive docstrings and error handling. Use type hints throughout.

PROMPT 3: GRADIO WEB INTERFACE (app_gradio.py)

Create app_gradio.py with inline Gradio interface featuring:

1. File uploader for .jpg/.jpeg/.png images, saves to /content/images/ (Colab) or ./images/

2. Calibration parameters: inner corners (cols/rows), square size in meters

3. "Test detection" button to validate corner detection without full calibration

4. "Run Calibration" button that saves results to calibration.json

5. Visualization controls: max overlay images, Plotly vs Matplotlib toggle, axis scale, corner drawing

6. Three output areas:

- 3D camera pose plot (interactive/static)

- Gallery of sample images with projected 3D axes

- Before/after undistortion preview (side-by-side)

7. Auto-detect Colab vs local environment

8. Use gr.Blocks with Soft theme, collapsible sections, proper error handling

PROMPT 4: CONSOLIDATED RUNNER SCRIPT (camera_calibration.py)

Create camera_calibration.py that consolidates the entire notebook workflow into a single function:

1. Environment detection (Colab, Jupyter, terminal)

2. Automatic dependency installation via subprocess

3. Google Drive mounting for Colab

4. Import all calibration utilities

5. Launch Gradio interface with appropriate settings for each environment

6. Include run_calibration_pipeline() function that can be imported or run directly

7. Handle errors gracefully with informative messages

8. Make it executable as both script and importable module

PROMPT 5: COMPREHENSIVE README.md

Create README.md with the following sections and content:

TITLE: Camera Calibration (OpenCV + Gradio)

OVERVIEW SECTION:

Write a brief description explaining this is an inline Gradio UI for camera calibration with 3D visualizations that works in Google Colab and Jupyter notebooks.

QUICK USAGE SECTIONS (create separate sections for each):

1. Google Colab Usage - show commands with !git clone, %cd, and import statements

2. Jupyter Notebook Local Usage - same commands as Colab

3. Desktop Terminal Usage - regular git clone and python script execution

4. Manual Setup Option - include virtual environment setup with venv

FEATURES SECTION:

List key features including file upload for chessboard images (minimum 15 recommended), OpenCV calibration pipeline, interactive 3D camera pose visualization using both Plotly and Matplotlib, sample images with projected world coordinate axes, and before/after undistortion preview.

FILES DESCRIPTION SECTION:

Create a bulleted list describing each Python file and its purpose, including calibration_utils.py for core helpers, app_gradio.py for the web interface, camera_calibration.py for consolidated execution, requirements.txt for dependencies, and the Jupyter notebook.

OUTPUT SECTION:

Describe what calibration.json contains including K matrix (camera intrinsics), distortion coefficients, extrinsic parameters, per-view errors, image metadata, and valid image paths. Also mention the three types of visualizations generated.

NOTES SECTION:

Include practical tips for good calibration results such as using sharp well-lit images, varying camera distance and angles, explaining that pattern size refers to inner corners, and suggestions for handling poor calibration results.

PROMPT 6: NOTEBOOK INTEGRATION & TESTING

Create the integration pieces:

1. Update calibration.ipynb to import camera_calibration and run the pipeline function
2. Ensure environment detection works correctly for Colab vs Jupyter vs terminal
3. Test that all visualization components work in notebook environments
4. Verify the consolidated script handles dependency installation properly
5. Add inline execution cells that demonstrate the workflow
6. Include error handling for common issues (no images, detection failures, etc.)
7. Make sure the Gradio interface appears inline in notebooks with proper sharing settings

Add any final touches like making sure file paths work across different environments and that the entire pipeline can be executed with minimal setup.