Computer Vision Project Report

DAI Yucheng, BOUHAI Yasser

December 15, 2024

1. Introduction

In a context where computer vision and image processing are rapidly evolving, the precise mapping of virtual objects onto real-world images represents a complex and essential challenge. This project aligns with this dynamic by focusing on converting three-dimensional coordinates from the real world into two-dimensional image coordinates, with the goal of seamlessly and realistically superimposing 3D objects onto 2D planes.

Using the OpenCV library and image processing algorithms, this project implements several key steps. First, it leverages camera calibration to obtain intrinsic parameters, followed by the computation of homography and projection matrices, enabling accurate transformations between different coordinate systems. Next, a virtual cube is drawn onto the images, demonstrating the projection of 3D objects onto 2D scenes and simulating applications such as augmented reality, robotic navigation, and 3D reconstruction.

This work also relies on advanced color segmentation techniques to extract key points necessary for projection. Despite the successes achieved, challenges remain, particularly in managing color variations caused by lighting and chromatic aberrations. This project proposes potential solutions to improve key point recognition, and provides a solid foundation for developing advanced applications in the field of computer vision.

2. Implementation steps

1. Use the *camera calibrator* module in Matlab to calculate the internal parameter matrix K of the camera.

2. Create an image for recognition using a computer vision system. Use Python and it's libraries to identify and extract key points from the images.

3. Calculate the homography matrix using four sets of corresponding points in the image and the real world. Then, using the camera's intrinsic parameters and the homography matrix, determine the projection matrix $\mathbf{P}$ for converting three-dimensional space to two-dimensional space, as well as the scaling factor α.

4. Use the projection matrix $\mathbf{P}$ and the scaling factor α to create a virtual cube on the original image and display it.

3. Obtaining the Internal Parameter K of the Camera

To obtain the internal parameters of the camera, you must first use software to lock the camera's focal length. Here, **Manuel Camera DSLR (Lite)** is used. Next, prepare a checkerboard pattern like the one shown in Figure 1 and take photos from various angles, ensuring that the angle does not exceed 30°. A minimum of 10 photos is required.

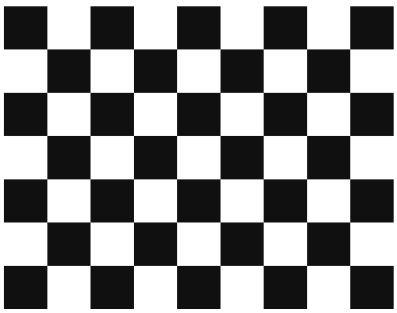

Figure 1: Checkerboard Grid used

The images are then loaded into MATLAB for analysis. It is necessary to discard images with significant errors (as shown in Figure 2, where the mean error in pixels exceeds 0.8, in our case we have an average of 0.68). Afterward, proceed with the analysis and export the camera's intrinsic parameters K.

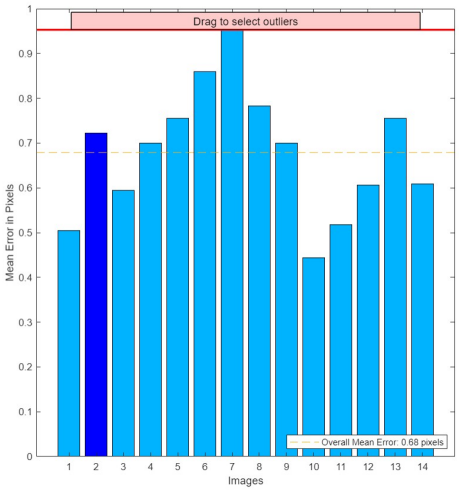

Figure 2: Graph showing the average error per image.

4. Creation of Grid and Steps for Color Detection

To project a 3D cube onto a 2D image, a grid of nine specific colors (red, green, blue, yellow, cyan, magenta, orange, white, and violet) is used as shown in Figure 3 . These colors help detect points of interest in the image, which are essential for calculating the homography matrix and enabling the projection.

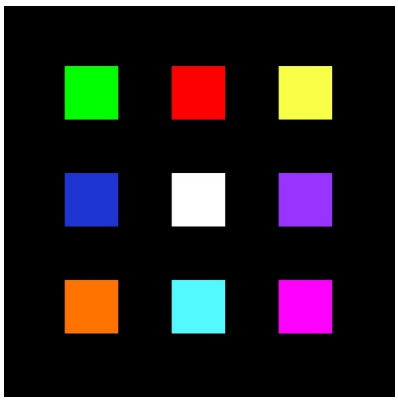

Figure 3: Grid used.

Here are the steps involved:

4.1. Create a Color Mask

Color masks are generated by defining specific ranges in the HSV (Hue, Saturation, Value) color space for each color. These ranges isolate the regions in the image that correspond to the respective colors,we used **OpenCV** methods in Python to check if each color falls within its defined range in the HSV color space. For instance:

- The color red requires two distinct ranges due to the wrapping of the hue circle in HSV space.
- Other colors, like blue, green, and yellow, are isolated using similarly defined ranges.

To demonstrate the process, here is an example that we will use throughout the steps with the color **Blue**:

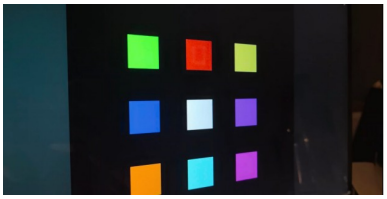

Figure 4: Example image used for color detection.

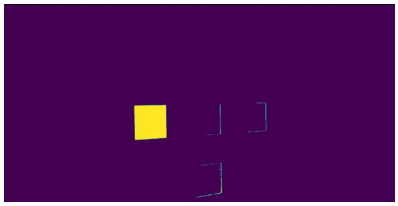

Figure 5: Color mask for detecting regions of interest.

4.2. Apply Morphological Operations

After generating the masks, we utilized the **morphologyEx** from the **openCV** library in Python to perform morphological operations to refine the masks and remove noise:

- **Opening:** Removes small, isolated noise points that might cause false detections.
- **Closing:** Fills small gaps within detected regions, making them more complete and coherent.

Kernel Size Adjustments:

- small kernels like 3×3 were used. However, they caused large regions to break into smaller parts, particularly in challenging detection scenarios.
- Larger kernels, such as 7×7 or 10×10, were employed to maintain the integrity of larger regions while reducing noise.
- Kernels larger than 20×20 were avoided because they significantly increased computational time (especially if we are dealing with videos).

Figure 6: Masks after the morphological operations.

In rare cases ,despite these adjustments, moderately sized irrelevant regions sometimes remained near the edges of the image, as they were too large to be fully removed by the chosen kernel sizes. These issues were addressed in the next step.

4.3. Identify Connected Regions and Calculate Centroids

After refining the masks:

- **Region Selection:** Only the largest connected region is retained.
- **Centroid Extraction:** The centroid of the largest region is calculated and used as key points for subsequent computations.

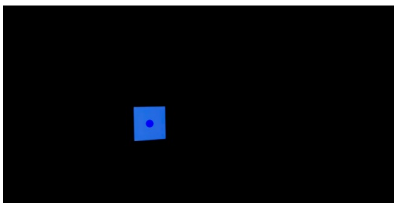

Figure 7: Final region with it's centroid.

We at least need 4 world points that are not collinear but we went for 9 for extra accuracy!

The process is repeated for each of the 9 colors for each frame!

5. Projection Matrix Estimation

The geometric model of a camera is expressed by a linear relationship between a 3D point M and its image point m:

$$m_{\text{image}} = PM_{\text{world}} \tag{1}$$

The goal here is to find a way to estimate P (the projection matrix), assuming that the point M and its image point m are known.

5.1. Homography Matrix Calculation

To compute P, we start by calculating the homography matrix H by fixing the coordinate on the z-axis in M_{world} to 1. From Equation (1), we can deduce:

$$m_{\text{image}} \times HM_{\text{world}} = 0_3$$

Next, we express m_{image} in the following form:

$$m_{\text{image}} = \begin{bmatrix} 0 & -1 & v_m \\ 1 & 0 & -u_m \\ -v_m & u_m & 0 \end{bmatrix}$$

where u_m and v_m are the coordinates of m_{image}.

For the projection matrix, we use the following notation:

$$H = \begin{bmatrix} H_1 & H_2 & H_3 \end{bmatrix}$$

Here, H_i represents the i-th row of H. By combining the formulas above, we obtain:

$$\begin{bmatrix} 0 & -1 & v_m \\ 1 & 0 & -u_m \\ -v_m & u_m & 0 \end{bmatrix} \begin{bmatrix} M^t H^1 \\ M^t H^2 \\ M^t H^3 \end{bmatrix} = 0_3$$

5.2. Construction of Matrix A

To estimate H, the number of unknowns and the number of equations must be linearly independent. The matrix H is 3×3, but we can normalize the last element to 1. Thus, H has 8 unknowns. Consequently, at least four sets of corresponding points between M_{world} and m_{image} are required to construct matrix A, given by:

$$A = \begin{bmatrix} 0^t & -M^t & v_m M^t \\ M^t & 0^t & -u_m M^t \\ -v_m M^t & u_m M^t & 0^t \end{bmatrix}$$

5.3. Scaling Factor and Rotation

Once H is estimated, we need to calculate the scaling factor α. In this project, α is crucial for:

1. Converting homogeneous coordinates into Cartesian coordinates.
2. Adjusting for scale differences between the 3D model points and the 2D image plane.
3. Compensating for scale variations introduced by the intrinsic and extrinsic camera parameters.

The relationship is given by:

$$H = K(R|T)$$

where K is the intrinsic camera matrix. First, we multiply the matrix H by the inverse of K to obtain the first two columns R_1, R_2 (rotation) and the third column T (translation):

$$(R_1, R_2, T) = K^{-1}H$$

The third row R_3 is calculated using the cross product:

$$R_3 = R_1 \times R_2$$

The rotation matrix must be orthogonal, so we have:

$$\det(R_1, R_2, R_3) = 1$$

From this, α is determined as:

$$\det(R_1, R_2, R_3) = \alpha^4$$

$$\alpha = \sqrt[4]{\det(R_1, R_2, R_3)}$$

Hence, the scaling factor has two solutions. In general, we choose the positive value as the scaling coefficient

$$P = \begin{pmatrix} \frac{R_1}{\alpha} & \frac{R_2}{\alpha} & \frac{R_3}{\alpha^2} & \frac{T}{\alpha} \end{pmatrix}.$$

5.4. Validation of the Projection Matrix with 9 Colors

After calculating the projection matrix **P**, it is essential to validate its effectiveness. In this project, a grid of 9 specific colors is used to represent the points of interest. Each color corresponds to a real-world coordinate, as shown in Figure 8.

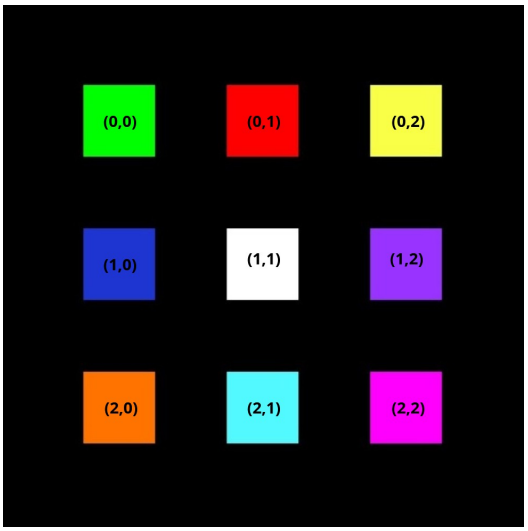

Figure 8: Coordinates of color blocks in the image and their real-world correspondences.

The correspondences between the colors and their real-world coordinates (x, y) are defined as follows:

- **Green**: $(0, 0)$ (top-left corner)

- **Red**: $(1, 0)$ (top-center)
- **Yellow**: $(2, 0)$ (top-right corner)
- **Blue**: $(0, 1)$ (middle-left)
- **White**: $(1, 1)$ (center)
- **Violet**: $(2, 1)$ (middle-right)
- **Orange**: $(0, 2)$ (bottom-left corner)
- **Cyan**: $(1, 2)$ (bottom-center)
- **Pink**: $(2, 2)$ (bottom-right corner)

These correspondences are used to compute the matrix $\mathbf{P}$ and project the real-world coordinates onto the image. The matrix is validated by testing if the projections match the detected color blocks in the image.

Figure 9: Validation example with projected color block centroids.

As shown in Figure 9, each projected block centroid is marked with a circle. These tests validate the accuracy of the matrix $\mathbf{P}$, ensuring that the correspondences between real-world and image coordinates are precise. Correct projection is essential for applications such as overlaying 3D objects onto 2D images.

5.5. Projection and Display of a 3D Cube

Using the obtained camera parameters, a 3D cube is projected onto the 2D image by utilizing the coordinates of its vertices. These vertices are then connected to form a complete cube.

6. Projection and Display of a 3D Cube

After calculating the projection matrix $\mathbf{P}$, the final step is to use this information to project a 3D cube onto a 2D image. This step involves using the coordinates of the cube's 3D vertices, which are then projected onto the 2D plane based on the camera's intrinsic and extrinsic parameters.

Methodology

1. **Defining the Cube Vertices**: The cube is defined in 3D space by its 8 vertices, centered at a given point. For example, if the center is $(1, 1, 0)$ and the cube size is 1, the vertices are calculated as:

$$\text{Vertices} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

2. **Projecting the Vertices**: The vertices defined in the 3D reference frame are projected onto the 2D plane using the projection matrix $\mathbf{P}$. The projection relation is given by:

$$\text{Projected Coordinates} = \mathbf{K} \cdot \mathbf{P} \cdot \text{Vertices}$$

   The projected coordinates are then normalized to obtain the positions (x, y) on the image.

3. **Drawing the Edges**: The vertices are connected by lines to form the cube. The 12 edges of the cube are defined by pairs of indices corresponding to the connected vertices.

4. **Rendering**: The edges are drawn on the image using functions such as 'cv2.line' in Python.

Results and Visualization

Once the vertices are projected and the edges are connected, a virtual cube is correctly overlaid onto the image. like Figure 10 shows .

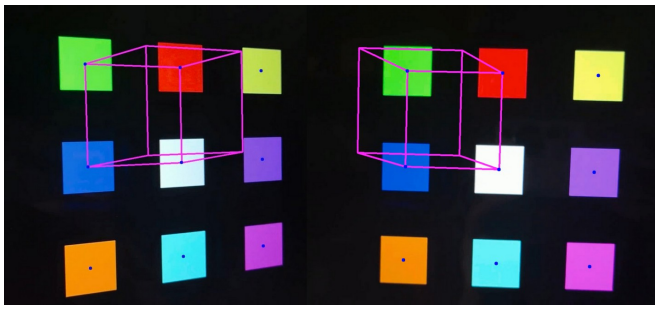

Figure 10: Example of a 3D cube projected onto a 2D image.

7. Method Tierces: Automatic K vs. Manual Calibration

We used both **automatic calibration** (Method Tierces) and **manual calibration** (e.g., MATLAB) to obtain the camera matrix K. Their main differences are summarized below.

7.1 Workflow

- **Automatic:** Runs `calibrateCamera` on video frames (detecting checkerboard corners) and directly returns K.
- **Manual:** Uses MATLAB's *camera calibrator* offline; the resulting K is hard-coded or loaded into the Python pipeline.

7.2 Code Structure

- **Automatic:** A single function reads frames, detects corners, and computes K.
- **Manual:** Skips automated detection; simply imports a precomputed K for 3D overlays.

7.3 Results

Both methods yield nearly identical 3D overlays in practice. Small differences in focal length or principal point location generally do not affect final projection quality. **Automatic calibration** is ideal for quick setups, while **manual calibration** provides more rigorous control and error analysis.

8. Further Improvements: Addressing Color Shift

While our pipeline successfully projects 3D objects onto real-world scenes, the final output video sometimes shows color mismatches. These shifts reduce the realism of augmented content. Potential improvements include:

8.1. White Balance

- **Adaptive White Balance**: Use methods like `cv2.xphoto.whiteBalance` to dynamically correct each frame.
- **Gray World Assumption**: Calibrate color temperature by referencing neutral or white patches.

8.2. Calibration-Based Correction

- **Color Chart**: Capture a standard chart and map its known sRGB values for accurate transformations.
- **Per-Channel Adjustment**: Compare each channel to a reference target to reduce hue and intensity errors.

8.3. Photometric Consistency

- **Histogram Matching**: Align rendered object histograms with the surrounding scene.
- **Illumination Estimation**: Estimate lighting properties (e.g., direction, intensity) and adapt rendered colors.

Implementing one or more of these techniques helps correct color shifts and improve overall visual coherence in augmented videos.

Summary

This step combines the information calculated in the previous sections to overlay a 3D object onto a 2D image. The projection process relies on the matrix **P**, the intrinsic parameters **K**, and the real-world coordinates. The success of this projection confirms the accuracy of the calculations for homography and camera calibration. This result opens avenues for applications such as augmented reality, where virtual objects are seamlessly integrated into real environments.

References

[1] https://zhuanlan.zhihu.com/p/583883569.

[2] https://blog.csdn.net/oSunBo/article/details/53698481.

[3] https://blog.csdn.net/pumpkinn233/article/details/143644092.

[4] https://blog.csdn.net/m0_37623374/article/details/125124836.

[5] https://cloud.tencent.com/developer/article/1513195.