

画像工学特論 課題: カメラ校正

本田純也 学籍番号: M233353

2025年5月21日

概要

【課題】以下を参考にして、自分のPCのカメラあるいは外付けWebカメラの校正を行い、校正の原理を簡単にまとめたもの、入力画像、校正結果、画像の歪み補正結果を添え、考察を行ってレポート(A4用紙4枚程度)を作成せよ。

- <https://github.com/PacktPublishing/OpenCV-3-Computer-Vision-with-Python-Cookbook>
- https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
- <https://learnopencv.com/camera-calibration-using-opencv/>

ただし、使用するプログラミング言語は問わない。

1 Introduction

一部のピンホールカメラは、画像に歪みをもたらすことがあるため、適切に歪みの校正を行う必要がある。[1] 本レポートでは、OpenCVライブラリを用いて実際のWebカメラの校正を実装し、その精度を評価した。チェックカーボードパターンを用いた実画像から内部パラメータと歪み係数を推定し、歪み補正の効果を定量的・視覚的に確認した。

また、原理などについては、Zhangらの論文も参考にした。[2]

2 Principle of Calibration

カメラ校正は、撮影レンズと撮影素子のシステムにおけるパラメータを決める重要なプロセスである。コンピュータビジョンにおいて、2次元画像から3次元情報を正確に抽出するためには、カメラの特性を正確に把握する必要がある。また、理論的なピンホールカメラモデルと実際のカメラには大きな差異が存在ことから、特に画像の周辺部に歪みが生じるため、測定精度に大きく影響する。

2.1 Pinhole Camera Model

理想的なピンホールカメラでは、3次元点 $\mathbf{X} = (X, Y, Z)$ の画像投影は透視投影により表される。

$$u = f \cdot \frac{X}{Z}$$

$$v = f \cdot \frac{Y}{Z}$$

ここで、 f は焦点距離、 (u, v) は画像座標である。しかし、この理想モデルでは実カメラの特性を適切に表現できない。

したがって、実際のカメラでは内部パラメータ行列 \mathbf{K} を考慮する必要がある。

$$\mathbf{K} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

ここで、

- f_x, f_y : x, y 方向の焦点距離 (ピクセル単位)
- c_x, c_y : 主点座標 (光学中心の画像上の位置)
- s : スキューパラメータ (センサーの軸の非直交性)

である。

2.2 Lens Distortion Model

実際のレンズには以下の歪みが存在し、特に画像周辺部で顕著になる。

2.2.1 放射歪み (Radial Distortion)

$$x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y' = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

2.2.2 接線歪み (Tangential Distortion)

$$x' = x + 2p_1 xy + p_2(r^2 + 2x^2)$$

$$y' = y + p_1(r^2 + 2y^2) + 2p_2 xy$$

ここで、 $r^2 = x^2 + y^2$ は画像中心からの距離である。

2.2.3 Projection Model

同次座標を用いた完全な投影式は以下のようになる。

$$\tilde{\mathbf{m}} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \tilde{\mathbf{X}}$$

ここで、

- $\mathbf{R} : 3 \times 3$ 回転行列 (カメラの向きを表す)
- $\mathbf{t} : 3 \times 1$ 並進ベクトル (カメラの位置を表す)
- $[\mathbf{R} \mid \mathbf{t}] : 3 \times 4$ の外部パラメータ行列

である。

特に、回転行列 \mathbf{R} は、世界座標系からカメラ座標系への回転変換を表し、以下の性質を持つ。

- $\mathbf{R}^T \mathbf{R} = \mathbf{I}$ (直交行列)
- $\det(\mathbf{R}) = 1$ (回転のみ、反射なし)

また、並進ベクトル \mathbf{t} は、世界座標系の原点からカメラ光学中心への並進を表す。

$$\bullet \quad \mathbf{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

これを展開すると：

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

最終的な画像座標は、

$$u = \frac{\tilde{u}}{\tilde{w}}, \quad v = \frac{\tilde{v}}{\tilde{w}}$$

となる。

この座標変換は、1. 世界座標系 → カメラ座標系、2. カメラ座標系 → 画像座標系の2段階で行われる。

2.2.4 Evaluation

校正精度は opencv で計算された、再投影誤差 (RMS: Root Mean Square) で評価する。

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^N \|m_i - \hat{m}_i\|^2}$$

ここで、 m_i は観測点、 \hat{m}_i は推定パラメータによる再投影点である。

3 Methods

実験環境の詳細を以下に示す。

- 使用ライブラリ: OpenCV 4.11.0
- プログラミング言語: Python 3.13
- カメラ: Web カメラ (1920×1080)
- チェッカーボード(図 1): 7×7 内部コーナー、20mm 正方形
- 画像解像度: 1920×1080 ピクセル

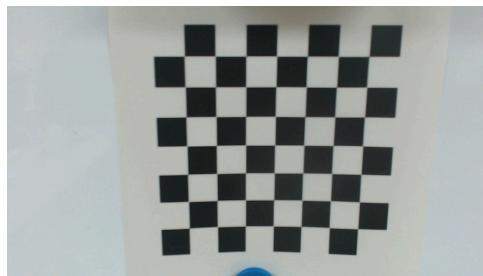


図 1: チェッカーボードの例

実験方法として、

1. 画像の撮影とコーナーの検出
2. OpenCV を用いたカメラの校正
3. 歪みの補正

のステップで行った。

4 Result

実験の結果を以下に示す。

4.1 画像の撮影とコーナーの検出

チェッカーボードを用いて、28枚の画像を撮影した。

図1に示すように、チェッカーボードを様々な角度で撮影した。画像中のチェッカーボードは明確に識別でき、 7×7 の内部コーナーが適切に配置されている。

次に、撮影した画像に対して、コード1に示すように、コーナーの検出を行った。特に、`cv2.findChessboardCorners()`は、チェッカーボードのコーナーを検出する関数である。

結果、28枚撮影した画像のうち、19枚の画像に対して、コーナーの検出を行うことができた。(検出成功率 $\frac{19}{28} = 67.9\%$)

4.2 OpenCV を用いたカメラの校正

カメラ校正是、コード2に示すようなコードを用いて行った。特に、`cv2.calibrateCamera()`の返り値の一つ目は、RMS エラーが格納される。また、`flags=cv2.CALIB_RATIONAL_MODEL`を指定している。

実際の構成結果は以下のとおりであった。

1. 内部パラメータ

- 焦点距離: $f_x = 1824.32, f_y = 1826.65$ pixels
- 中心点: cx=893.24, cy=544.59
- アスペクト比: 1.001275

したがって、カメラ行列 K は、

$$K = \begin{pmatrix} 1824.32 & 0 & 893.24 \\ 0 & 1826.65 & 544.59 \\ 0 & 0 & 1 \end{pmatrix}$$

であった。

2. 歪み係数

- k1 = -20.373670
- k2 = 166.930980
- k3 = 920.085674

4.3 歪みの補正

構成した情報を元に、コード3のようなコードを用いて行った。補正した結果を図2に示す。

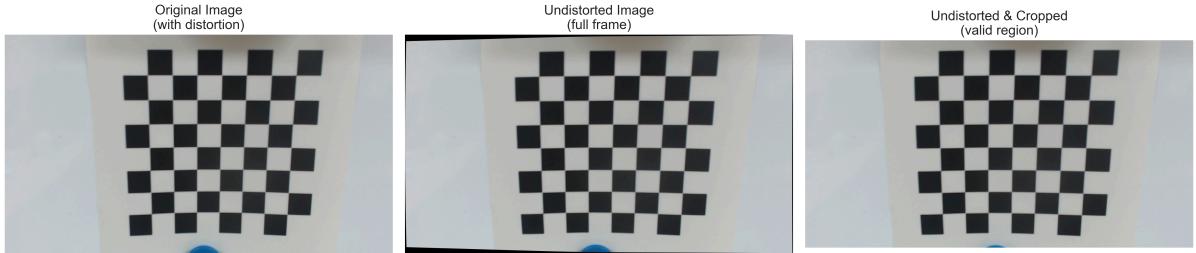


図 2: チェッカーボードの例

一番左に示している画像が、補正前の画像、真ん中が補正後の画像、右が補正後にクリップしてサイズを合わせた画像である。

補正前と後の画像を比べると、補正が行われていることがわかる。

また、カメラ校正で得られた、再投影誤差は、2.9017 であった。

5 Disucussion

今回の検出成功率 $\frac{19}{28}$ は、撮り方の問題(角度やカメラからの距離など)を考慮して撮影することで、改善が期待される。[1]

今回の再投影誤差の 2.9017 は、実用水準にある。今回は、Web カメラを用いていることから、元々の歪みが低いことが考えられる。構成に用いる画像数を上げたり、色々な角度から撮影することでより精度を高められると考える。

また、歪み係数 k_1 が負の値であったため、このカメラの歪みは、樽型であることがわかる。これは、広角レンズである今回の Web カメラの特徴が絡んでいると考えることができる。

補正後の画像を見た時にも、エッジに補正後が見られ、視覚的に歪み補正の効果を確認できたと考える。

以上より、自分が撮影したチェッカーボードパターンを用いた画像から、内部パラメータと歪み係数を推定し、歪み補正の効果を定量的・視覚的に確認することができた。

6 Appendix

注意: 実際の実験では、クラスを作成しているため、以下のコードは実験クラスに属する。ここには、レポートで参照したコードのみ載せている。全コードは Github に上げている。https://github.com/gomagoma7/advanced_image_processing_report

6.1 設定パラメータ

$$\text{設定パラメータ} = \begin{cases} \text{checkerboard_size : } (7, 7) \\ \text{square_size : } 20.0 \text{ mm} \\ \text{flags : } \text{cv2.CALIB_RATIONAL_MODEL} \\ \text{criteria : } (30, 0.001) \end{cases}$$

6.2 Code

```
def process_images(self, image_files):
    for image_file in image_files:
        img = cv2.imread(image_file)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # コーナー検出
        ret, corners = cv2.findChessboardCorners(
            gray, self.checkerboard_size,
            cv2.CALIB_CB_ADAPTIVE_THRESH +
            cv2.CALIB_CB_FAST_CHECK +
            cv2.CALIB_CB_NORMALIZE_IMAGE
        )
        if ret:
            # サブピクセル精度改良
            criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
            corners = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)

            self.object_points.append(self.objp)
            self.image_points.append(corners)
```

コード 1: process_images

```
def calibrate_camera(self):
    self.rms_error, self.camera_matrix, self.dist_coeffs, self.rvecs, self.tvecs =
    cv2.calibrateCamera(
        self.object_points, self.image_points, self.image_size, None, None,
        flags=cv2.CALIB_RATIONAL_MODEL # 高次歪みモデル使用
    )
```

コード 2: calibrate_camera

```
def demonstrate_undistortion(self, image_index=0):
    # 最適カメラ行列の計算
    new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(
        self.camera_matrix, self.dist_coeffs, (w, h), 1, (w, h)
    )
    # 歪み補正実行
    undistorted = cv2.undistort(img, self.camera_matrix,
                                self.dist_coeffs, None, new_camera_matrix)
```

コード 3: calibrate_camera

参考文献

- [1] OpenCV, 「OpenCV: Camera Calibration — docs.opencv.org」 . 2025 年.
- [2] Z. Zhang, 「A flexible new technique for camera calibration」, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000, doi: 10.1109/34.888718.