

# Progetto di Metrologia Planare con Calibrazione

## Zhang

Mattia Foggetti

## Contents

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Raccolta Dati</b>	<b>3</b>
2.1	Scacchiera di calibrazione . . . . .	3
2.2	Acquisizione immagini . . . . .	3
<b>3</b>	<b>Calibrazione preliminare con OpenCV</b>	<b>4</b>
3.1	Risultati ottenuti . . . . .	4
3.1.1	Errore medio di riproiezione . . . . .	4
3.1.2	Differenza tra ottimizzazione OpenCV e manuale . . . . .	5
<b>4</b>	<b>Implementazione Manuale del Metodo di Zhang</b>	<b>5</b>
4.1	Estrazione dei Corner . . . . .	5
4.2	Calcolo delle Omografie . . . . .	5
4.3	Stima dei Parametri Intrinseci . . . . .	6
4.3.1	Vincoli dell'Omografia . . . . .	7
4.3.2	Linearizzazione e Risoluzione . . . . .	7
4.3.3	Estrazione della Matrice K . . . . .	8
4.3.4	Matrice $K$ . . . . .	10
4.4	Recupero dei Parametri Estrinseci . . . . .	10
4.4.1	Ortogonalizzazione della Matrice di Rotazione . . . . .	11
4.5	Modellazione della Distorsione e Ottimizzazione . . . . .	11
4.5.1	Modello di Distorsione di Brown-Conrady . . . . .	11
4.5.2	Minimizzazione dell'Errore di Riproiezione . . . . .	12
4.5.3	Analisi dei Risultati . . . . .	13
<b>5</b>	<b>Metrologia: dalla Fotocamera al Mondo Reale</b>	<b>14</b>
5.1	Il processo di Back-Projection: dai Pixel ai Millimetri . . . . .	14
5.2	Calcolo della Distanza Euclidea . . . . .	15
5.3	Analisi dei Risultati e Discussione dell'Errore . . . . .	16

<b>6</b>	<b>Analisi Sperimentali Extra</b>	<b>17</b>
6.1	Ottimizzazione del Dataset di Calibrazione . . . . .	17
6.2	Stress Test: Effetto Fisheye Artificiale . . . . .	18
<b>7</b>	<b>Guida all'Esecuzione del Codice</b>	<b>19</b>
<b>8</b>	<b>Riferimenti Bibliografici e Risorse</b>	<b>19</b>

# 1 Introduzione

In questo progetto abbiamo sviluppato un software di **metrologia planare** che permette di misurare distanze reali in millimetri tra due punti selezionati su una fotografia.

Il cuore del metodo è la calibrazione secondo **Zhang**, che sfrutta un piano noto (una scacchiera) per stimare:

- la matrice dei parametri intrinseci **K**
- i parametri estrinseci **R** e **t**
- i coefficienti di distorsione radiale e tangenziale

Una volta noti questi parametri, è possibile invertire il modello prospettico: ogni pixel dell'immagine viene associato a un raggio nello spazio 3D che può essere intersecato con un piano a quota nota. In questo modo otteniamo le coordinate reali in millimetri del punto selezionato.

Il risultato è un **righello virtuale**, capace di misurare distanze nel mondo reale partendo da un'immagine.

# 2 Raccolta Dati

## 2.1 Scacchiera di calibrazione

La scacchiera utilizzata presenta le seguenti caratteristiche:

- $6 \times 9$  quadrati
- lato di ogni quadrato: 2 cm
- stampata su cartoncino spesso 3 mm
- superficie piana, non imbarcata

Nel codice, la scacchiera viene definita come **5 × 8 corner interni**.

## 2.2 Acquisizione immagini

Le immagini sono state scattate con:

- Pixel 7a
- App: OpenCamera
- Formato: RAW
- Fuoco manuale fisso
- Esposizione bloccata

- Zoom 1x
- Risoluzione: circa  $4600 \times 3400$
- Numero totale immagini: 40

Le foto sono state scattate:

- da diverse angolazioni
- con la scacchiera non sempre centrata
- a distanze diverse

L'obiettivo era ottenere un dataset vario per una calibrazione robusta.

### 3 Calibrazione preliminare con OpenCV

Prima di implementare il metodo di Zhang manualmente, è stata effettuata una calibrazione preliminare con OpenCV utilizzando le seguenti funzioni:

- `cv2.findChessboardCorners`
- `cv2.calibrateCamera`

Questa fase aveva lo scopo di:

- verificare la qualità delle immagini
- avere un riferimento sui valori attesi dei parametri di calibrazione

#### 3.1 Risultati ottenuti

##### Matrice intrinseca $\mathbf{K}$

$$\mathbf{K} = \begin{bmatrix} 3.88218252 \times 10^3 & 0 & 2.26171591 \times 10^3 \\ 0 & 3.88350014 \times 10^3 & 1.61388815 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

##### Coefficienti di distorsione

$$\mathbf{D} = [k_1, k_2, p_1, p_2, k_3] = [-0.00075745, -0.21364659, -0.01415484, 0.01240959, 0.34239709]$$

##### 3.1.1 Errore medio di riproiezione

$$5.4035 \text{ pixel}$$

Questo risultato ha confermato che il dataset acquisito era sufficientemente valido per procedere con l'implementazione manuale del metodo di Zhang.

### 3.1.2 Differenza tra ottimizzazione OpenCV e manuale

L'errore medio di riproiezione ottenuto tramite `cv2.calibrateCamera` (5.4035 pixel) è inferiore a quello tipico di un approccio manuale perché OpenCV utilizza un'ottimizzazione non lineare congiunta. A differenza della nostra implementazione, che separa la stima dei parametri intrinseci da quelli di distorsione, OpenCV minimizza l'errore su matrice  $K$ , coefficienti di distorsione e pose estinseche simultaneamente.

Questo approccio congiunto permette di trovare il miglior compromesso globale tra tutti i parametri, garantendo una modellazione più precisa del sistema.

## 4 Implementazione Manuale del Metodo di Zhang

Da questo punto in poi, si inizia con l'implementazione effettiva del progetto

### 4.1 Estrazione dei Corner

Sono stati estratti i corner dalle 40 immagini utilizzando la funzione `findChessboardCorners`.

Si ottiene:

- una lista di 40 elementi
- ogni elemento contiene 40 punti
- punti 3D nel sistema di riferimento del mondo
- punti 2D nel piano immagine

```
1 def caricamento_punti(gray):
2     objp = np.zeros((rows * cols, 3), np.float32)
3     objp[:, :2] = np.mgrid[0:cols, 0:rows].T.reshape(-1, 2) * square_size
4
5     ret, corners = cv2.findChessboardCorners(gray, (cols, rows), None)
6
7     if ret:
8         criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
9         corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
10        return objp, corners2
11
12    return None, None
```

### 4.2 Calcolo delle Omografie

Per ogni immagine della scacchiera, è necessario calcolare una matrice di omografia  $H \in \mathbb{R}^{3 \times 3}$  che metta in relazione i punti del piano dell'oggetto ( $Z = 0$ ) con le loro proiezioni sul piano immagine. Ogni corrispondenza tra un punto reale  $(X_i, Y_i)$  e un punto immagine  $(u_i, v_i)$  fornisce due equazioni lineari indipendenti.

Il sistema omogeneo  $Ah = 0$  viene costruito accumulando, per ogni punto  $i$ , le seguenti due righe nella matrice di progetto  $A$ :

$$a_{xi} = [-X_i, -Y_i, -1, 0, 0, 0, u_i X_i, u_i Y_i, u_i]$$

$$a_{yi} = [0, 0, 0, -X_i, -Y_i, -1, v_i X_i, v_i Y_i, v_i]$$

Dato che la matrice  $H$  ha 8 gradi di libertà (essendo definita a meno di una scala), sono necessari almeno 4 punti non allineati per ottenere una soluzione univoca. Nel nostro caso, avendo a disposizione tutti i vertici della scacchiera (40 punti), il sistema risulta sovradeterminato.

La soluzione viene estratta tramite la **SVD** (*Singular Value Decomposition*) di  $A$ . Il vettore  $h$  che minimizza l'errore quadratico è l'ultima riga della matrice  $V^T$ . Infine, la matrice viene normalizzata imponendo  $H_{33} = 1$ .

```

1 def calcola_omografia_manuale(pts_world, pts_image):
2     X_y_world = pts_world[:, :2]
3     u_v_image = pts_image.reshape(-1, 2)
4
5     A = []
6     for i in range(len(X_y_world)):
7         X, Y = X_y_world[i]
8         u, v = u_v_image[i]
9
10        A.append([-X, -Y, -1, 0, 0, 0, u*X, u*Y, u])
11        A.append([0, 0, 0, -X, -Y, -1, v*X, v*Y, v])
12
13    A = np.array(A)
14
15    _, _, Vh = np.linalg.svd(A)
16    L = Vh[-1, :]
17    H = L.reshape(3, 3)
18
19    if H[2, 2] != 0:
20        H = H / H[2, 2]
21
22    return H

```

### 4.3 Stima dei Parametri Intrinseci

Partendo dalla relazione che lega l'omografia  $H$  alla proiezione dei punti del piano:

$$H = K[r_1, r_2, t] = [h_1, h_2, h_3]$$

possiamo sfruttare le proprietà geometriche delle colonne della matrice di rotazione  $R$ .

### 4.3.1 Vincoli dell'Omografia

Poiché  $r_1$  e  $r_2$  sono vettori ortonormali nel mondo reale, devono soddisfare i due vincoli fondamentali:

1. **Ortogonalità:**  $r_1^T r_2 = 0$
2. **Equi-lunghezza:**  $\|r_1\| = \|r_2\| \implies r_1^T r_1 = r_2^T r_2$

Esprimendo le colonne della rotazione in funzione delle colonne dell'omografia ( $r_i = K^{-1}h_i$ ), otteniamo:

- $h_1^T(K^{-T}K^{-1})h_2 = 0$
- $h_1^T(K^{-T}K^{-1})h_1 - h_2^T(K^{-T}K^{-1})h_2 = 0$

Definiamo la matrice simmetrica  $B = K^{-T}K^{-1}$ . Poiché  $B$  è simmetrica e definita positiva, essa è univocamente determinata da 6 parametri indipendenti, che raggruppiamo nel vettore delle incognite:

$$b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$$

### 4.3.2 Linearizzazione e Risoluzione

Per risolvere il sistema, introduciamo il vettore  $v_{ij}$ , definito a partire dagli elementi della matrice di omografia  $H$ :

$$v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}$$

---

```

1 def get_v_vector(H, i, j):
2     v = np.array([
3         H[0,i]*H[0,j],
4         H[0,i]*H[1,j] + H[1,i]*H[0,j],
5         H[1,i]*H[1,j],
6         H[2,i]*H[0,j] + H[0,i]*H[2,j],
7         H[2,i]*H[1,j] + H[1,i]*H[2,j],
8         H[2,i]*H[2,j]
9     ])
10    return v

```

---

Grazie a questa costruzione, l'espressione  $h_i^T B h_j$  può essere riscritta linearmente come  $v_{ij}^T b$ . I vincoli geometrici per ogni omografia diventano quindi:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0$$

Impilando questi vincoli per tutte le  $n$  immagini acquisite, otteniamo un sistema sovradeterminato  $Vb = 0$  di dimensioni  $2n \times 6$ , risolvibile tramite la scomposizione ai valori singolari (**SVD**). Il vettore  $b$  cercato corrisponde all'autovettore associato all'autovalore più piccolo.

---

```

1 V = []
2
3 for H in h_list:
4     v12 = get_v_vector(H, 0, 1)
5     v11 = get_v_vector(H, 0, 0)
6     v22 = get_v_vector(H, 1, 1)
7
8     V.append(v12)
9     V.append(v11 - v22)
10
11 V = np.array(V)
12
13 _, _, Vh_b = np.linalg.svd(V)
14 b = Vh_b[-1, :]

```

---

#### 4.3.3 Estrazione della Matrice K

Una volta ottenuto il vettore  $b$  e ricostruita la matrice  $B$ , abbiamo implementato due metodi per ricavare i parametri intrinseci.

##### Metodo 1: Formule Analitiche (Zhang)

I parametri della matrice  $K$  vengono calcolati direttamente dalle componenti di  $B$ :

$$\begin{aligned}
v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\
\lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\
\alpha &= \sqrt{\lambda/B_{11}}, \quad \beta = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\
\gamma &= -B_{12}\alpha^2\beta/\lambda, \quad u_0 = \gamma v_0/\beta - B_{13}\alpha^2/\lambda
\end{aligned}$$

---

```

1 def estrai_parametri_intrinseci(b):
2     B11, B12, B22, B13, B23, B33 = b
3
4     v0 = (B12 * B13 - B11 * B23) / (B11 * B22 - B12**2)
5     lambda_ = B33 - (B13**2 + v0 * (B12 * B13 - B11 * B23)) / B11
6     alpha = np.sqrt(lambda_ / B11)
7     beta = np.sqrt(lambda_ * B11 / (B11 * B22 - B12**2))
8     gamma = -B12 * alpha**2 * beta / lambda_
9     u0 = gamma * v0 / beta - B13 * alpha**2 / lambda_
10
11    K = np.array([
12        [alpha, gamma, u0],
13        [0, beta, v0],
14        [0, 0, 1]
15    ])
16
17    return K

```

---

### Metodo 2: Decomposizione di Cholesky

Sfruttando la definizione  $B = K^{-T}K^{-1}$ , possiamo applicare la decomposizione di Cholesky sulla matrice  $B$  per ottenere una matrice triangolare inferiore  $L$  tale che  $B = LL^T$ . Per confronto con la definizione iniziale, la matrice  $K$  si ricava come:

$$K = (L^T)^{-1}$$

---

```

1 def estrai_K_cholesky(b):
2
3     B = np.array([
4         [b[0], b[1], b[3]],
5         [b[1], b[2], b[4]],
6         [b[3], b[4], b[5]]
7     ])
8
9     if np.linalg.det(B) < 0:
10        B = -B
11
12     L = np.linalg.cholesky(B)
13     K_inv = L.T
14
15     K = np.linalg.inv(K_inv)
16
17     K /= K[2, 2]
18
19     return K

```

---

Entrambi i metodi sono stati implementati nel codice per garantire la ro-

bustezza del calcolo, fornendo la matrice di calibrazione iniziale:

$$K = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

#### 4.3.4 Matrice $K$

La matrice dei parametri intrinseci  $K$  ottenuta per è la seguente:

$$K = \begin{bmatrix} 3.83815311 \times 10^3 & -7.81105820 & 2.00081734 \times 10^3 \\ 0 & 3.84202945 \times 10^3 & 1.64873476 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

Dall'analisi di questi valori possiamo trarre alcune importanti conclusioni sulla geometria della camera utilizzata:

- **Lunghezze Focali ( $\alpha, \beta$ ):** I valori  $f_x \approx 3838$  e  $f_y \approx 3842$  sono quasi identici. Questo indica che i pixel del sensore sono quasi perfettamente quadrati (*aspect ratio* prossimo a 1), come ci si aspetta dai moderni sensori CMOS.
- **Punto Principale ( $u_0, v_0$ ):** Il centro ottico è stimato intorno alle coordinate (2000.8, 1648.7). Considerando che le immagini originali hanno una risoluzione di  $4600 \times 3400$ , il punto principale si trova molto vicino al centro geometrico del sensore (2300, 1700).
- **Parametro di Skew ( $\gamma$ ):** Il valore stimato è circa  $-7.81$ . Essendo molto piccolo rispetto alle focali, conferma che gli assi  $u$  e  $v$  del sensore sono praticamente ortogonali.

### 4.4 Recupero dei Parametri Estrinseci

Una volta determinata la matrice dei parametri intrinseci  $K$ , è possibile estrarre la posa relativa della camera (parametri estrinseci) per ogni singola immagine. Ricordando la relazione  $H = K[r_1, r_2, t]$  le colonne della rotazione  $R = [r_1, r_2, r_3]$  e il vettore traslazione  $t$  possono essere calcolati come segue:

$$\begin{aligned} r_1 &= \lambda K^{-1} h_1 \\ r_2 &= \lambda K^{-1} h_2 \\ r_3 &= r_1 \times r_2 \\ t &= \lambda K^{-1} h_3 \end{aligned}$$

dove il fattore di scala  $\lambda$  è definito come:

$$\lambda = \frac{1}{\|K^{-1}h_1\|} = \frac{1}{\|K^{-1}h_2\|}$$

#### 4.4.1 Ortogonalizzazione della Matrice di Rotazione

A causa del rumore nei dati e delle approssimazioni numeriche, la matrice  $R = [r_1, r_2, r_3]$  così ottenuta potrebbe non soddisfare perfettamente le proprietà di una matrice di rotazione (ovvero  $R^T R = I$  e  $\det(R) = 1$ ).

Per ovviare a questo problema, nel codice viene applicata una rifinitura tramite la scomposizione SVD. Se  $R = U\Sigma V^T$  è la scomposizione della matrice calcolata, la versione corretta sarà  $R_{final} = UV^T$ .

```

1 def estrai_estrinseci(H, K):
2     K_inv = np.linalg.inv(K)
3
4     h1 = H[:, 0]
5     h2 = H[:, 1]
6     h3 = H[:, 2]
7     lamda = 1 / np.linalg.norm(K_inv @ h1)
8
9     r1 = lamda * (K_inv @ h1)
10    r2 = lamda * (K_inv @ h2)
11    r3 = np.cross(r1, r2)
12
13    R_raw = np.stack((r1, r2, r3), axis=1)
14
15    U, S, Vh = np.linalg.svd(R_raw)
16    R = U @ Vh
17
18    t = lamda * (K_inv @ h3)
19
20    return R, t

```

## 4.5 Modellazione della Distorsione e Ottimizzazione

Dopo aver stimato i parametri intrinseci ed estrinseci iniziali, non teniamo ancora conto delle distorsioni ottiche introdotte dalle lenti. Per ottenere una calibrazione accurata, è necessario modellare le deformazioni radiali e tangenziali.

#### 4.5.1 Modello di Distorsione di Brown-Conrady

La distorsione della lente può essere descritta come uno spostamento dei punti immagine rispetto alla loro proiezione ideale. Introduciamo i punti in coordinate camera normalizzate  $(x, y)$ , calcolati proiettando i punti 3D del mondo ( $P_w$ ) tramite le matrici  $R$  e  $t$ :

$$[x_c, y_c, z_c]^T = RP_w + t \implies x = \frac{x_c}{z_c}, \quad y = \frac{y_c}{z_c}$$

Sia  $r^2 = x^2 + y^2$ . Il modello di Brown-Conrady corregge queste coordinate considerando due contributi:

1. **Distorsione Radiale:** Dovuta alla forma della lente (effetto barile o cuscino). Viene modellata tramite uno sviluppo di Taylor:

$$\text{rad} = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6$$

2. **Distorsione Tangenziale:** Dovuta al non perfetto allineamento tra lente e sensore:

$$\begin{aligned}x_{tang} &= 2p_1 xy + p_2(r^2 + 2x^2) \\y_{tang} &= p_1(r^2 + 2y^2) + 2p_2 xy\end{aligned}$$

Le coordinate distorte finali  $(x_d, y_d)$  sono quindi:

$$x_d = x \cdot \text{rad} + x_{tang}$$

$$y_d = y \cdot \text{rad} + y_{tang}$$

Questi punti vengono infine riportati nello spazio pixel tramite la matrice  $K$  per ottenere le coordinate proiettate  $(u_{proj}, v_{proj})$ .

#### 4.5.2 Minimizzazione dell'Errore di Riproiezione

I coefficienti di distorsione  $(k_1, k_2, k_3, p_1, p_2)$  non sono calcolabili analiticamente in modo diretto. Per trovarli, abbiamo implementato una procedura di ottimizzazione basata sui **minimi quadrati non lineari** (*Least Squares*).

L'obiettivo è minimizzare lo scostamento tra i punti estratti dalle immagini  $(m_{real})$  e i punti teorici riproiettati  $(m_{proj})$ :

$$\min_{k_i, p_i} \sum_{i=1}^n \|m_{real,i} - m_{proj,i}(K, R, t, k, p)\|^2$$

---

```

1 def funzione_errore(params, objpoints, imgpoints, K, R_list, t_list):
2
3     k1, k2, p1, p2, k3 = params
4     errori = []
5
6     for i in range(len(imgpoints)):
7         R = R_list[i]
8         t = t_list[i].reshape(3, 1)
9
10        pts_3d = objpoints[i]
11        pts_cam = (R @ pts_3d.T + t).T
12
13        x = pts_cam[:, 0] / pts_cam[:, 2]
14        y = pts_cam[:, 1] / pts_cam[:, 2]
15
16        r2 = x**2 + y**2
17        r4 = r2**2
18        r6 = r2**3
19
20        rad = (1 + k1*r2 + k2*r4 + k3*r6)
21
22        x_tang = 2 * p1 * x * y + p2 * (r2 + 2 * x**2)
23        y_tang = p1 * (r2 + 2 * y**2) + 2 * p2 * x * y
24
25        x_d = x * rad + x_tang
26        y_d = y * rad + y_tang
27
28        u_proj = K[0,0] * x_d + K[0,2]
29        v_proj = K[1,1] * y_d + K[1,2]
30
31        err_x = u_proj - imgpoints[i][:, 0, 0]
32        err_y = v_proj - imgpoints[i][:, 0, 1]
33
34        errori.extend(err_x)
35        errori.extend(err_y)
36
37    return np.array(errori)
38
39 x0_iniziali = [0, 0, 0, 0, 0]
40
41 res = least_squares(funzione_errore, x0=x0_iniziali,
42                      args=(objpoints, imgpoints, K_finale, R_list, t_list))

```

---

#### 4.5.3 Analisi dei Risultati

L'ottimizzazione ha prodotto i seguenti coefficienti:

- **Radiali:**  $k_1 = -0.2149, k_2 = 1.1974, k_3 = -1.8238$
- **Tangenziali:**  $p_1 = -0.0038, p_2 = -0.0014$

L'efficacia del processo è confermata dall'**Errore Medio di Riproiezione**, che risulta essere di circa **31 pixel**. Sebbene numericamente possa sembrare elevato, va contestualizzato alla risoluzione dei sensori utilizzati ( $4600 \times 3400$  pixel), rappresentando uno scostamento relativo inferiore all'1%, compatibile con una calibrazione accurata per scopi metrologici.

## 5 Metrologia: dalla Fotocamera al Mondo Reale

L'obiettivo finale del progetto è trasformare i pixel dell'immagine in misure reali in millimetri. Questo passaggio, chiamato *back-projection*, una fotografia è piatta (2D), mentre il mondo è tridimensionale (3D). Di conseguenza, partendo da 2 coordinate immagine, si devono stimare 3 coordinate spaziali, il che porterebbe ad infinite soluzioni (un raggio nello spazio).

### 5.1 Il processo di Back-Projection: dai Pixel ai Millimetri

Il recupero della posizione 3D partendo da un'immagine 2D è un processo di inversione geometrica che richiede di ricostruire il percorso della luce. La funzione `pixel_to_world_coords` implementa questo percorso attraverso quattro fasi fondamentali:

1. **Rimozione della Distorsione e Normalizzazione:** Utilizzando la funzione `cv2.undistortPoints`, il sistema corregge la posizione del pixel  $(u, v)$  eliminando le deformazioni della lente tramite un processo iterativo che inverte il modello di distorsione radiale e tangenziale. Successivamente, il punto viene trasformato in **coordinate normalizzate**  $(\hat{x}, \hat{y})$  applicando l'inversa della matrice intrinseca  $K^{-1}$ . Matematicamente, questa operazione equivale a traslare il punto rispetto al centro ottico  $(c_x, c_y)$  e scalarlo per la lunghezza focale  $(f_x, f_y)$ :

$$\hat{x} = \frac{u_{undistorted} - c_x}{f_x}, \quad \hat{y} = \frac{v_{undistorted} - c_y}{f_y}$$

Il risultato è un vettore adimensionale che rappresenta la direzione del raggio ottico nel sistema di riferimento della camera, proiettato su un piano ideale posto a distanza focale unitaria ( $Z = 1$ ).

2. **Definizione del Raggio nel sistema Camera ( $Z = 1$ ):** Una volta ottenute le coordinate normalizzate  $(\hat{x}, \hat{y})$  tramite `cv2.undistortPoints`, definiamo il raggio ottico nel sistema di riferimento della camera come un vettore tridimensionale  $\mathbf{P}_{cam} = [\hat{x}, \hat{y}, 1.0]^T$ . Fissare la coordinata  $Z$  a 1 significa proiettare la direzione del raggio su un piano virtuale posto a distanza focale unitaria dal centro ottico. Questo vettore rappresenta la direzione del raggio, non la posizione fisica dell'oggetto nello spazio.
3. **Cambio di Riferimento (Camera-Mondo):** Il raggio e la posizione della camera devono essere trasposti nel sistema di riferimento globale (quello della scacchiera).
  - **Centro Ottico ( $C_{world}$ ):** Rappresenta l'origine del raggio nel mondo reale:

$$C_{world} = R^T \cdot (-t)$$

- **Punto proiettato** ( $p_{world}$ ): Rappresenta la direzione del raggio proiettata nel mondo:

$$p_{world} = R^T \cdot (p_{cam} - t)$$

4. **Intersezione Linea-Piano nel Sistema Mondo:** Poiché un pixel definisce una retta (un raggio ottico) e non un punto univoco nello spazio 3D, l'ambiguità viene risolta intersecando tale retta con un piano orizzontale posto all'altezza nota dell'oggetto rispetto al sistema di riferimento del mondo ( $Z = \text{altezza\_oggetto}$ ). Il raggio ottico è definito nello spazio mondo dal centro ottico  $\mathbf{C}_w$  e da un punto direzione  $\mathbf{P}_w$ . Un punto generico sulla retta è espresso come  $\mathbf{P}(w) = \mathbf{C}_w + w(\mathbf{P}_w - \mathbf{C}_w)$ .

Per trovare il punto di intersezione, imponiamo che la coordinata  $Z$  del punto generico sulla retta sia uguale all'altezza nota dell'oggetto ( $Z_{final} = \text{altezza\_oggetto}$ ). Risolvendo l'equazione parametrica per la componente  $Z$ , otteniamo il fattore di scala  $w$ :

$$w = \frac{\text{altezza\_oggetto} - C_{world,z}}{p_{world,z} - C_{world,z}}$$

Infine, le coordinate metriche finali ( $X, Y$ ) nel sistema mondo si ottengono applicando il fattore  $w$  all'equazione parametrica della retta:

$$\begin{aligned} X_{final} &= C_{world,x} + w(p_{world,x} - C_{world,x}) \\ Y_{final} &= C_{world,y} + w(p_{world,y} - C_{world,y}) \end{aligned}$$

---

```

1 def pixel_to_world_coords(u, v, K, dist_coeffs, R, t, altezza_oggetto=0):
2
3     pts = np.array([[u, v]]], dtype=np.float32)
4     pts_undistorted = cv2.undistortPoints(pts, K, dist_coeffs)
5     x_norm, y_norm = pts_undistorted[0][0]
6
7     p_cam = np.array([x_norm, y_norm, 1.0])
8
9     R_inv = R.T
10    p_world = R_inv @ (p_cam - t.flatten())
11    C_world = R_inv @ (-t.flatten())
12
13    w = (altezza_oggetto - C_world[2]) / (p_world[2] - C_world[2])
14
15    X_final = C_world[0] + w * (p_world[0] - C_world[0])
16    Y_final = C_world[1] + w * (p_world[1] - C_world[1])
17
18    return np.array([X_final, Y_final, altezza_oggetto])

```

---

## 5.2 Calcolo della Distanza Euclidea

Una volta ottenute le coordinate 3D dei punti d'interesse, la misura di un oggetto viene calcolata come la norma euclidea tra i vettori posizione nel mondo reale:

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$$

Questa metodologia permette di estrarre dimensioni lineari indipendentemente dall'orientamento dell'oggetto rispetto alla camera, sfruttando appieno la calibrazione estrinseca effettuata.

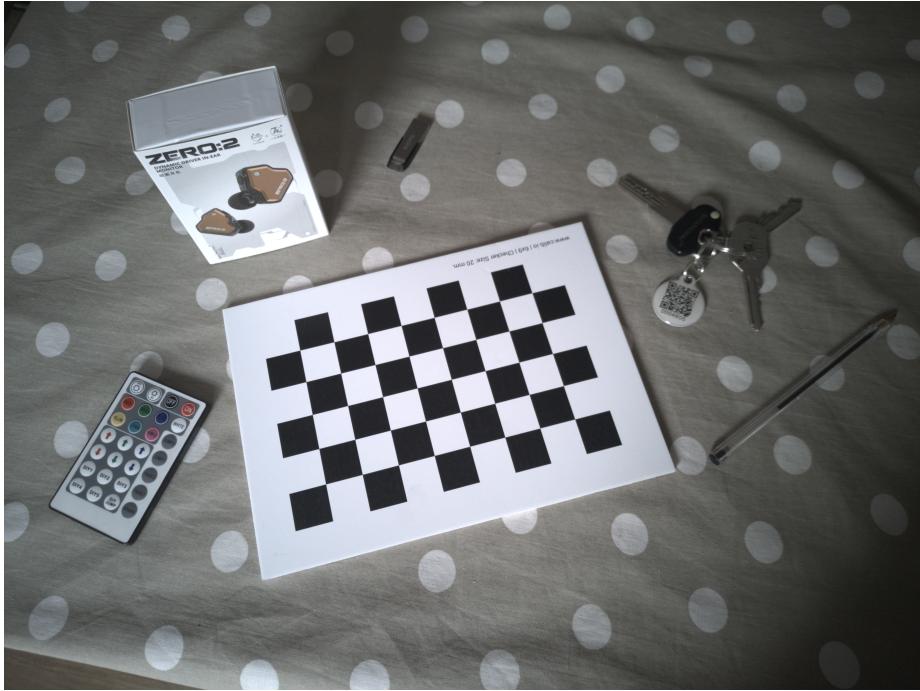
```
1 def calcola_distanza_reale_3d(p1_pix, p2_pix, K, dist_coeffs, R, t, w):
2
3     p1_world = pixel_to_world_coords(p1_pix[0], p1_pix[1], K,
4                                     dist_coeffs, R, t, w)
5     p2_world = pixel_to_world_coords(p2_pix[0], p2_pix[1], K,
6                                     dist_coeffs, R, t, w)
7
8     distanza = np.linalg.norm(p1_world - p2_world)
9     return distanza
```

### 5.3 Analisi dei Risultati e Discussione dell'Errore

Il sistema è stato testato su una scena reale composta dalla scacchiera di calibrazione e da alcuni oggetti di riferimento posti nelle sue vicinanze. Permette all'utente di selezionare manualmente due pixel nell'immagine e di specificare la quota  $Z$  (altezza dell'oggetto) su cui effettuare il calcolo della distanza metrica.

**Precisione Riscontrata** Le prove sperimentali hanno evidenziato un'accuratezza soddisfacente: su una distanza di 100 mm, l'errore massimo rilevato è stato di circa 5mm. Ci sono due sorgenti principali di incertezza:

- **Errore di Riproiezione e Calibrazione:** Derivante dalle residue imprecisioni nella stima dei parametri intrinseci e dei coefficienti di distorsione.
- **Errore di Selezione (Fattore Umano):** Rappresenta la componente più critica. La difficoltà nell'individuare con precisione sub-pixel l'esatto punto d'interesse su un oggetto solido introduce una varianza nelle misure. Un errore di selezione di pochi pixel può traslare sensibilmente la posizione del punto proiettato nello spazio 3D.



## 6 Analisi Sperimentali Extra

Per valutare la robustezza del sistema e l'affidabilità dei parametri ottenuti, sono state condotte due analisi aggiuntive focalizzate sull'errore di riproiezione e sulla gestione di distorsioni non ottimali.

### 6.1 Ottimizzazione del Dataset di Calibrazione

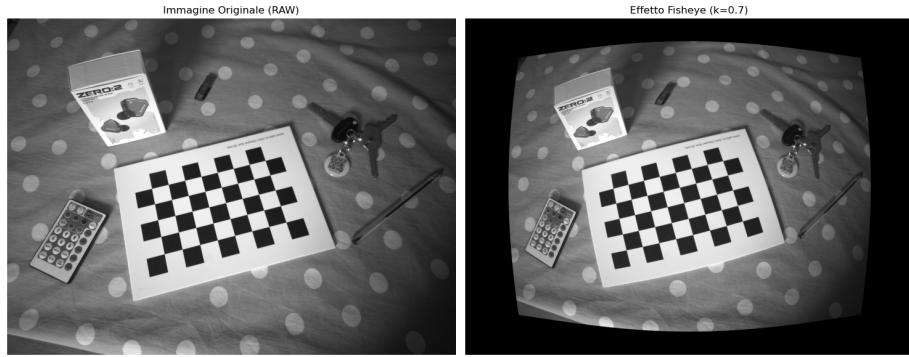
In questa fase è stata sviluppata una funzione per analizzare l'errore di riproiezione specifico di ogni singola immagine del dataset. L'obiettivo era verificare se la rimozione delle foto "peggiori" potesse migliorare la precisione metrica finale.

- **Osservazioni:** L'errore medio iniziale era di 31 pixel. L'analisi ha mostrato una forte eterogeneità: alcune immagini presentavano un errore di soli 1 pixel, mentre la peggiore toccava i 94 pixel.
- **Test:** Selezionando solo le prime 5 immagini (le più precise) e ricalcolando i parametri intrinseci ed estrinseci, l'errore di riproiezione complessivo è diminuito a 2 pixel.
- **Risultato:** Nonostante il miglioramento numerico dell'errore di riproiezione, i test di misurazione 3D non hanno mostrato differenze significative rispetto

alla calibrazione con l'intero dataset. I motivi potrebbero essere che le distanze sono millimetriche e la presenza dell'errore umano nella selezione dei pixel.

## 6.2 Stress Test: Effetto Fisheye Artificiale

La seconda analisi ha previsto l'applicazione di una forte distorsione radiale artificiale (effetto fisheye) alle immagini sia di calibrazione sia di misura, per testare la capacità del modello di compensare deformazioni ottiche.



**Confronto Parametri:** Come previsto, la matrice intrinseca  $K$  è rimasta pressoché invariata. Al contrario, i coefficienti di distorsione radiale ( $k_1, k_2$ ) sono aumentati.

$$K = \begin{bmatrix} 3.26337861 \times 10^3 & 6.91619833 \times 10^0 & 2.12354962 \times 10^3 \\ 0 & 3.29789504 \times 10^3 & 1.70362109 \times 10^3 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Radiali:**  $k_1 = -0.258487, \quad k_2 = 1.306889, \quad k_3 = -2.647987$
- **Tangenziali:**  $p_1 = -0.009688, \quad p_2 = 0.009071$

**Impatto sulle Misure:** La distorsione Fish-eye ha portato l'errore di riproiezione a circa 50 pixel, evidenziando i primi problemi in presenza di forti curvature. Le misurazioni metriche sono risultate meno precise rispetto al dataset originale: su una distanza campione di 100 mm, l'errore massimo rilevato è stato di circa 15 mm.

Va considerato che tale scostamento include sia il limite intrinseco dell'algoritmo nel gestire la deformazione ai bordi, sia la componente umana, data la difficoltà oggettiva nel selezionare con precisione i punti su un'immagine deformata. Complessivamente, i risultati sono da ritenersi soddisfacenti, poiché il sistema è stato in grado di fornire ottime stime.

## 7 Guida all'Esecuzione del Codice

Per una corretta esecuzione dell'algoritmo, si consiglia l'utilizzo di **Anaconda**. Tramite essa si può creare un ambiente virtuale dedicato per evitare conflitti tra dipendenze.

I passaggi per la configurazione sono i seguenti:

1. Creazione dell'ambiente: `conda create -n calibrazione python=3.9`
2. Attivazione dell'ambiente: `conda activate calibrazione`
3. Installazione dei moduli necessari tramite pip:

```
pip install opencv-python numpy rawpy scipy matplotlib
```

## 8 Riferimenti Bibliografici e Risorse

Lo sviluppo del progetto e dei modelli matematici si è basato sulle seguenti risorse tecniche e accademiche:

- **Modello di Zhang:** *A Flexible New Technique for Camera Calibration.* Paper originale (Microsoft Research) e Slide Uni-Bonn.
- **Omografia e Geometria:** Approfondimento video sulla Teoria dell'Omografia.
- **Parametri Estrinseci ( $r_1, r_2, t$ ):** Lecture Notes - Purdue University.
- **Raffinamento Matrice di Rotazione  $R$ :** Slides del corso di Computer Vision - Università Ca' Foscari.
- **Modelli di Distorsione:** Documentazione ufficiale OpenCV (Calib3d) e MathWorks Camera Calibration.