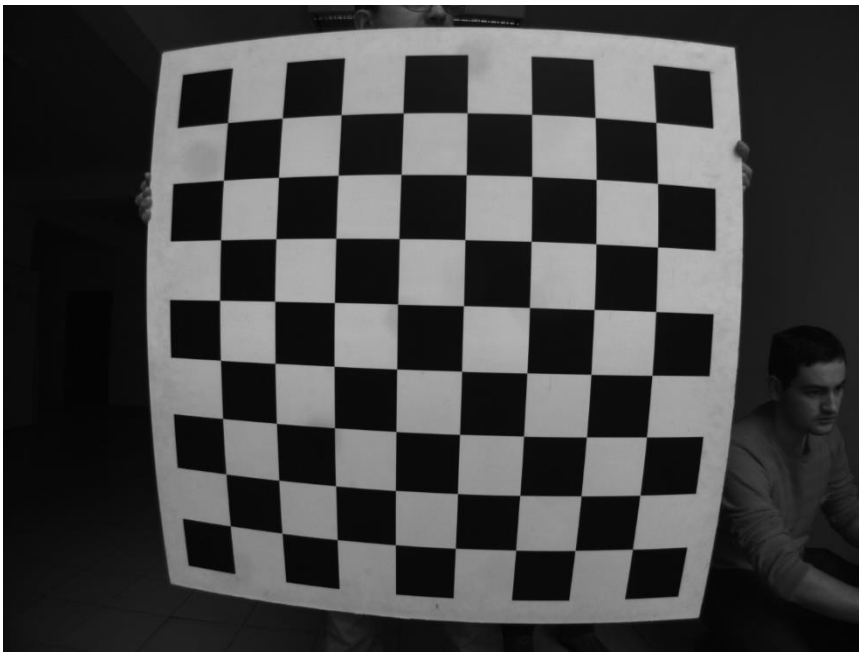
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare

Departamentul de Calculatoare

Documentație proiect

Eliminarea distorsiunilor lentilelor



Profesor îndrumător: Vancea Cristian

Student: Astalîș Lorena-Maria

Grupa: 30234

Anul 2022

Cuprins

1. Introducere

Obiectivul acestei lucrări este de a elimina distorsiunile din imagini generate de lentilele unei camere (se cunosc parametrii intrinseci ai acesteia). Ideal, o fotografie are o perspectivă perfect mapată pe scena reală care este surprinsă de aceasta, în realitate, mai ales când dorim să folosim imagini pentru a reconstrui un accident sau alte situații în care avem nevoie de o reprezentare cât mai realistă pe imagini pentru a deduce diverse date, acestea nu oferă informații relevante în forma brută care ajung, din cauza lentilelor folosite de către cameră. Pentru a ține distorsiunile lentilelor la un nivel cât mai scăzut se recomandă un obiectiv cu unghi mic. În practică acest lucru este greu de obținut din cauza distanței față de care trebuie fotografiat obiectul, pentru cazul cu reconstrucția unui accident de obicei avem nevoie de imagini cu un câmp vizual larg. Din fericire prin tehnici software putem elimina aceste distorsiuni ale imaginilor pentru a elimina cât mai mult aceste erori. [1]

2. Considerații teoretice

Ideea generală de la care se pornește în implemnetarea unei soluții de eliminare a distorsiunilor unei imagini constă în maparea pixelilor din imaginea cu distorsiune în imaginea rezultat care elimină curburile generate de obietive.

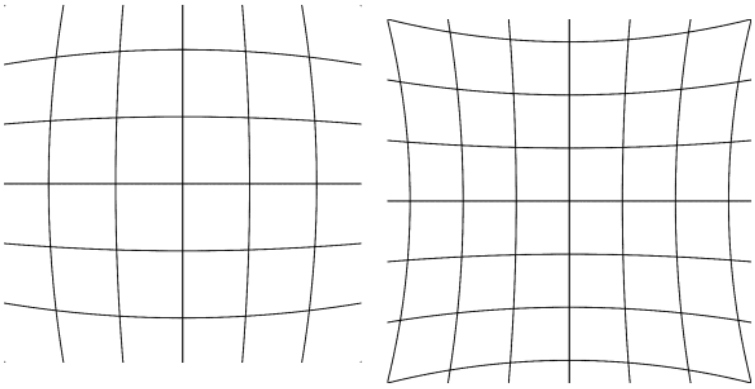


Figura 2.1: Cele mai întâlnite tipuri de distorsiuni (barrel – stânga și pincushion – dreapta) [1]

Date pe care le vom cunoaște legate de imagine:

- Distanța focală – distanța dintre centrul optic și planul imagine `(fx, fy)`
- Punctul principal – coordonatele centrului real al imaginii, intersecția dintre axa optică și planul imagine (coordonate date ăn pixeli) `(u0, v0)`
- Coeficienți de distorsiune
    - Radiali `k1, k2`

- Tangențiali `p1, p2`

Principiul care stă la bază este corespondența unui pixel din imagine sursă la o altă locație în imaginea destinație

```
(x', y') = (x+δx, y+δy)
```

Algoritmul de corecție

```
Pentru fiecare pixel (u, v) din imaginea destinație D
- Se calculează (x, y) în planul imagine
    ○ x = (u-u0)/fx
    ○ y = (v-v0)/fy
- Se calculează coordonatele în imaginea distorsionată
  S: (x', y') = (x+δx, y+δy)
- Se calculează coordonatele în pixeli în imaginea
  distorsionată S:
    ○ u' = u0 + x' fx
    ○ v' = v0 + y' fy
- Se atribuie pixelului destinație valoarea pixelului
  sursă în poziția găsită
      D(u, v) = S(u', v')
```

Soluția naivă, cea mai simplificată at fi maparea forward, presupunem că matricea A este sursa și B destinația [2]

```
B(u(x, y), v(x, y)) = A(x, y)
```

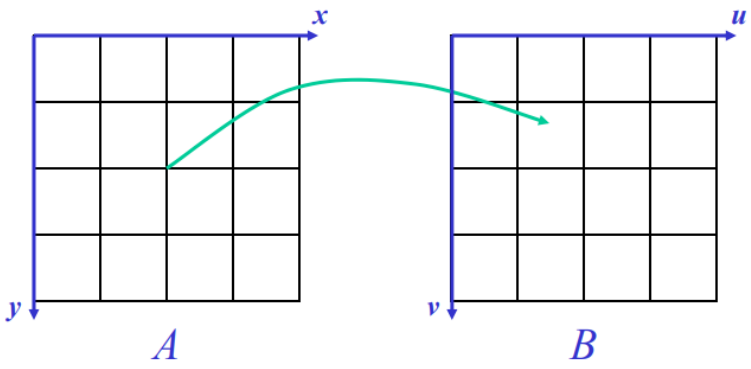


Figura 2.1: Maparea unui pixel din matricea A în matricea B [2]

Problema principală a acestei abordări este faptul că nu întotdeauna rezultatul va fi exact locația unui pixel deoarece rezultatul va fi un număr real, iar pixelii au coordonate numere întregi, așadar este nevoie de o aproximare. Această aproximare va cauza o pierdere a calității și va rezulta o imagine destinație pixelată. Soluția propusă pentru aceasta este să ”împrăștiem” efectul aplicat pe un anumit pixel și pe vecinătatea acestuia prin interpolare bilineară, după cum se poate observa în figura 2.2.

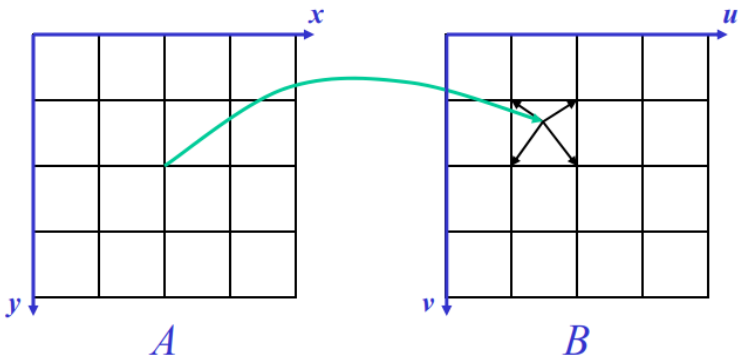


Figura 2.2: Soluția pentru problema interpolării forward cu interpolare bilineară

Interpolarea bilineară – perspectiva matematică [3]
Sau interpolarea 2-D este definită ca o interpolare liniară pe 2 axe (x și y).
Să presupunem că avem punctele definite prin coordonatele (xk, yk) unde k = 1, 2. Aceste puncte sunt locațiile lui Q11, Q12, Q21, Q22. Pentru orice x și y dat care sunt între xk și yk, prin aplicarea interpolării bilineare putem găsi punctul P (definit de x și y).

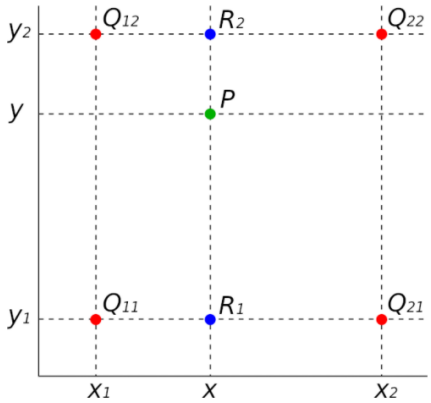


Figura 2.3: Pricipiul de funcționare a interpolării bilienare

Pentru a găsi punctul P(x, y) este nevoie de 2 etape

- 2 interpolări liniare pe axa x, pentru a găsi punctele R1 și R2 imediate
- 1 interpolare liniară pe axa y pentru a găsi punctul P

Așadar, interpolarea bilineară constă în 2 interpolări, una pe axa x și alta pe axa y.

Punctul R1(x, y) va fi definit ca

```
R1(X, Y) = Q11(X2-X)/(X2-X1) + Q21(X-X1)(X2-X1)
```

R2(x, y) va fi definit ca:

```
R2(X, Y) = Q12(X2-X)/(X2-X1) + Q22(X-X1)(X2-X1)
```

Punctul interpolat P(x, y) este definit ca:

```
P(X,Y) = R1(Y2-Y)/(Y2-Y1)+R2(Y-Y1)/(Y2-Y1)
```

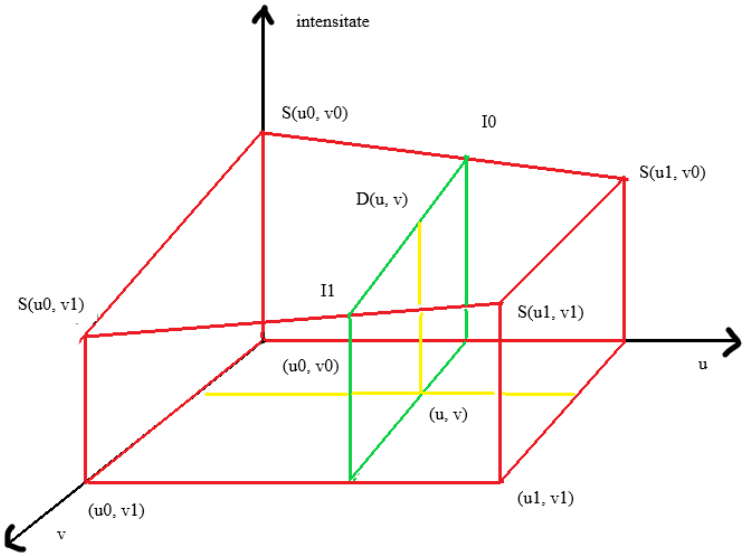


Figura 2.4: Exemplu vizual cu interpolarea biliniară a unui pixel

Imaginea de mai sus transpusă în pseudocod:

```
u0 = integer(u')
v0 = integer(v')
u1 = u0 + 1
v1 = v0 + 1
I0 = S(u0, v0) (u1 – u') + S(u0, v1) (u' – u0)
I1 = S(u0, v1) (u1 – u') + S(u1, v1) (u' – u0)
D(u, v) = I0 (v1 – v') + I1 (v' – v0)
```

3. Specificații de implementare

Pentru implementare a fost folosită librăria OpenCV pentru citirea și afișarea imaginii sub formă de matrice și pentru a elimina distorsiunile și cu funcția deja implementată în OpenCV pentru eliminarea distorsiunilor. Apoi se dau paramaetrii de intrare menționați la punctul 2:

- Distanța focală – distanța dintre centrul optic și planul imagine `(fx, fy)`
- Punctul principal – coordonatele centrului real al imaginii, intersecția dintre axa optică și planul imagine (coordonate date ăn pixeli) `(u0, v0)`
- Coeficienți de distorsiune
  - Radiali `k1, k2`
  - Tangențiali `p1, p2`

Cu ajutorul acestora am implementat algoritmul descris în cod C++.

```
for (int u = 0; u < dst.rows; u++) {
      for (int v = 0; v < dst.cols; v++) {
            // Coords in image plan
            float x = (u - v0) / fx;
            float y = (v - u0) / fy;

            // Coords in distors image
            float r2 = pow(x, 2) + pow(y, 2);
            float dx = x * (k1 * r2 + k2 * pow(r2, 2)) + 2 * p1 * x * y + p2 *
      (r2 + 2 * pow(x, 2));
            float dy = y * (k1 * r2 + k2 * pow(r2, 2)) + p1 * (r2 + 2 * pow(y,
2)) + 2 * p2 * x * y;

            // Coords in pixels in distors image
            float x_ = x + dx;
            float y_ = y + dy;
            float u_ = v0 + x_ * fx;
            float v_ = u0 + y_ * fy;

            // Interpolation
            int u00 = (int)u_;
            int v00 = (int)v_;
            int u01 = u00 + 1;
            int v01 = v00 + 1;
            float i00 = src.at<uchar>(u00, v00) * (float) (u01 - u_) +
src.at<uchar>(u00, v01) * (float) (u_ - u00);
            float i01 = src.at<uchar>(u00, v01) * (float) (u01 - u_) +
src.at<uchar>(u01, v01) * (float) (u_ - u00);
            dst.at<uchar>(u, v) = i00 * (v01 - v_) + i01 * (v_ - v00);
            }
   }
```

Prima fază a implementării, după pseudocodul dat mai sus, am parcurs imaginea destinație pixel cu pixel (notație (u, v)), pentru fiecare fiind calculat care pixel din imaginea sursă i se va atribui, în conformitate cu formulele oferite, cu mențiunea că δx, δy sunt obținuți din formula:

$$\begin{bmatrix}\partial x\\ \partial y\end{bmatrix}=\begin{bmatrix}\partial x^r+\partial x^t\\ \partial y^r+\partial y^t\end{bmatrix}=\begin{bmatrix}x\cdot\left(k_1\cdot r^2+k_2\cdot r^4\right)+2p_1\cdot xy+p_2(r^2+2x^2)\\ y\cdot\left(k_1\cdot r^2+k_2\cdot r^4\right)+p_1(r^2+2y^2)+2p_2\cdot xy\end{bmatrix}$$

Figura 3.1: δx, δy [4]

Atribuirea pixelilor destinație la cei din imaginea sursă se realizează cu interpolare biliniară, după specificațiile teoretice.

4. Rezultate experimentale

Pentru partea de testare a implementării, ca rezultatul să poată fi verificat am ales să afișez imaginea inițială prelucrată și cu ajutorul funcțiti din OpenCv

```
void cv::undistort ( InputArray    src,
                     OutputArray   dst,
                     InputArray    cameraMatrix,
                     InputArray    distCoeffs,
                     InputArray    newCameraMatrix = noArray()
                   )
```

- src – imaginea sursă (distorsionată)
- dst – imaginea destinație
- cameraMatrix

$$A=\begin{bmatrix}f_x & 0 & c_x\\ 0 & f_y & c_y\\ 0 & 0 & 1\end{bmatrix}$$

Figura 3.2: Matricea cameră [5]

- coeficienții de distorsiune: în această ordine – k1, k2, p1, p2

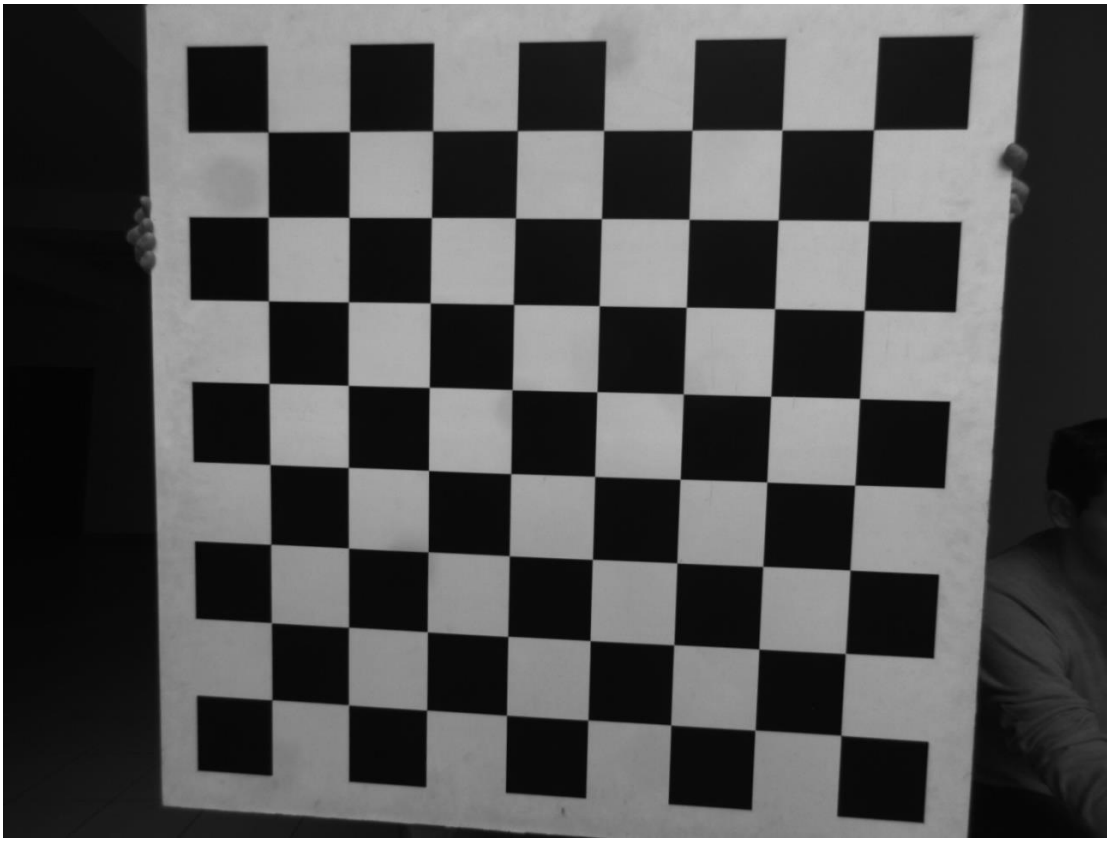


Figura 4.1: Rezultat așteptat, obținut cu OpenCv::undistort()

Pentru algoritmul implementat în primă fază [6] (vezi commit 9 aprilie) s-au obținut următoarele rezultate:

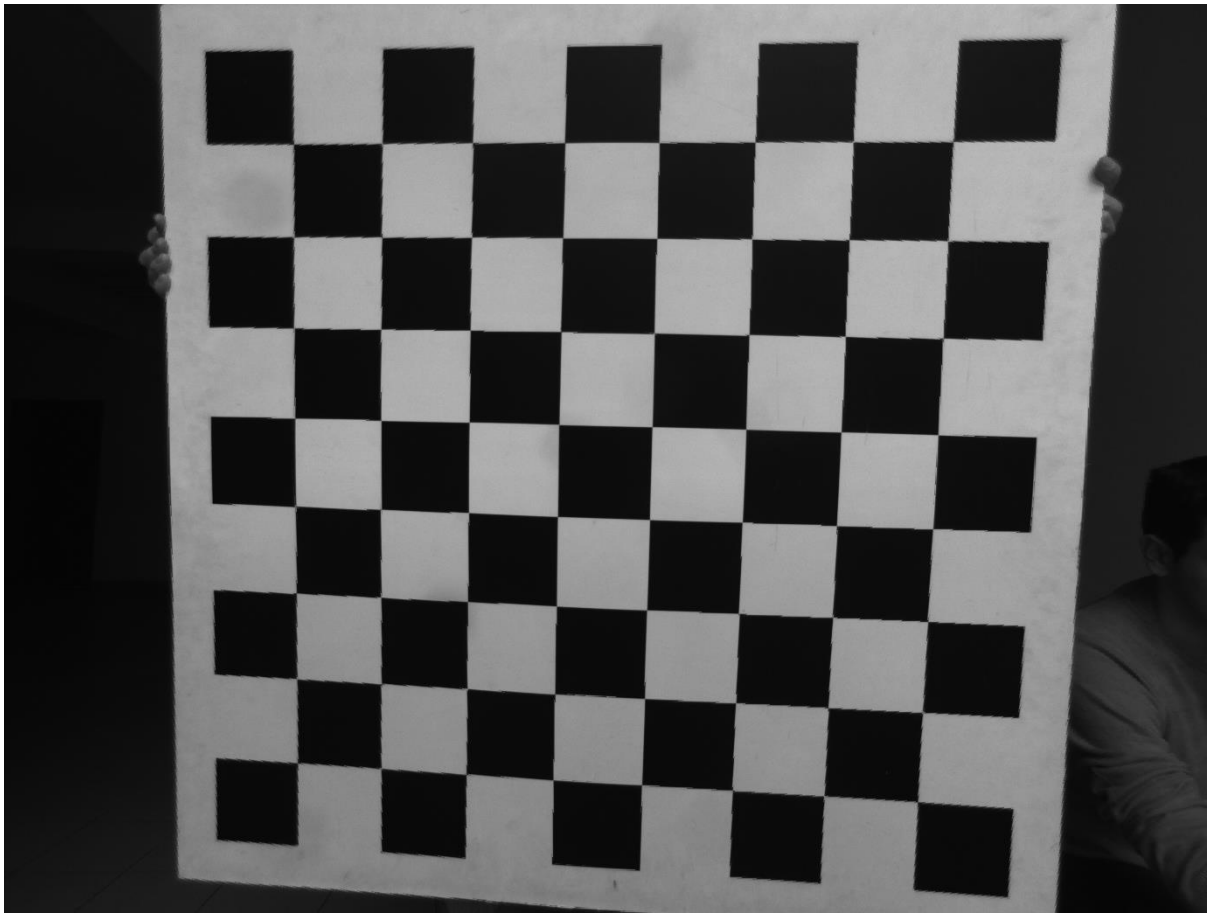


Figura 4.2: Rezultat fără interpolare (se poate observa pe pătratele negre, acestea sunt pixelate)

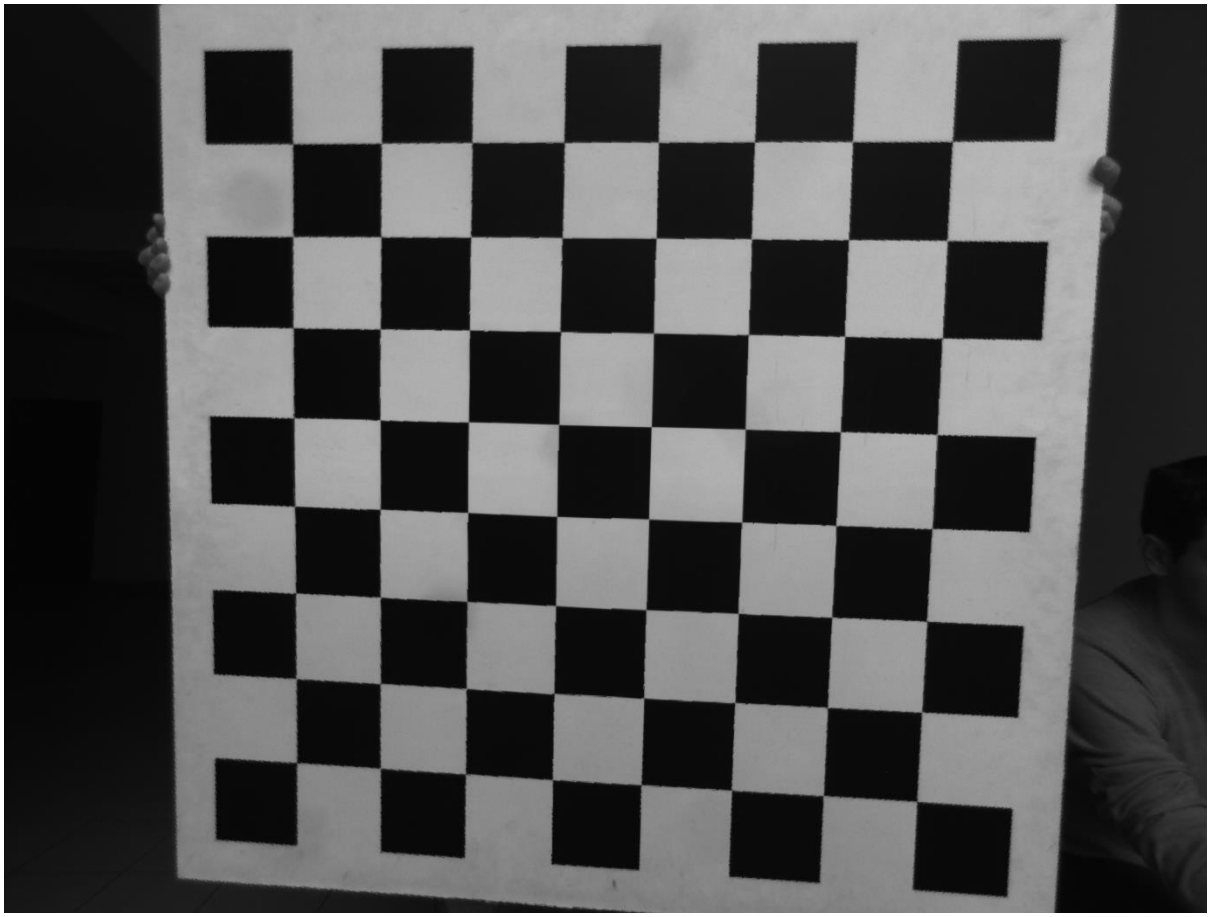


Figura 4.3: Rezultat cu interpolare (efectul de pixelare este mai puțin evident în acest caz)

5. Concluzii

În concuzie, chiar dacă momentan lentilele nu sunt perfecte și nu oferă o imagine cu acuratețe mare, pentru a avea o imagine cât mai bună a realității, acest lucru este regalbil prin niște calcule pentru a elimina erorile. Tehnica de eliminarea distorsiunilor este foarte folosită mai ales la filmări, unde în spate se regăsește un ecran verde, pentru a putea pune alte imagini sau animații 3d pe acestea, este nevoie de o imagine fără distorsiuni pentru ca să pară cât mai reală, pentru aceasta se folosește o tablă de șah, similară cu imaginea de test din proiect pentru a se identifica exact parametrii și pentru a prelucra imaginile.

6. Bibliografie

[1] H. Wolfgang, Correcting lens distortions in digital photographs, 2010.
http://www.imagemagick.org/Usage/lens/correcting_lens_distortions.pdf
[2] W. Wriggers, Interpolation and Morphing, The University of Texas.
http://www.biomachina.org/courses/imageproc/051.pdf
[3] Bilinear Interpolation
https://x-engineer.org/bilinear-interpolation/
[4] R. Dănescu, Modelul camerei. Procesul de formare a imaginilor
https://users.utcluj.ro/~rdanescu/pi_c02.pdf

[5] OpenCV Documenation
https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga69f2545a8b62a6b0fc2ee060dc30559d
[6] GitHub https://github.com/astalislorena/lenses-distortion-correction-in-image