Rubik's Cube Solving Using Image Processing

ANDREI OVIDIU DANIEL, CÎRSTEA
UNIVERSITATEA TRANSILVANIA DIN BRAȘOV
Facultatea de: MATEMATICA-INFORMATICA
Specializarea: INFORMATICA
Email: andrei.cirstea@student.unitbv.ro

ALEXANDRU, EDVEȘ
UNIVERSITATEA TRANSILVANIA DIN BRAȘOV
Facultatea de: MATEMATICA-INFORMATICA
Specializarea: INFORMATICA
Email: alexandru.edves@student.unitbv.ro

Abstract

The project offers a simplified method for solving the Rubik's Cube by allowing users to upload separate images for each face of the cube. The application then guides the user through a series of moves, providing intuitive step-by-step guidance to the solution, thus facilitating the solving process and making the user experience more accessible.

The main goal of this project is to demonstrate a practical situation in which digital image processing algorithms can be applied.

The Rubik's Cube, being a globally popular subject, is the ideal platform to highlight the utility of these algorithms in an engaging context.

Keywords: *Digital Image Processing, Rubik's Cube, Algorithm, Canny Edge Detection, Gaussian Filter, Projective Transformation*

Introducere

Cubul Rubik, unul dintre cele mai cunoscute și iubite puzzle-uri din lume, a provocat mințile curioase de zeci de ani. În era tehnologică actuală, rezolvarea acestui puzzle poate beneficia de avansurile din domeniul procesării imaginilor digitale. Acest proiect propune o soluție pentru rezolvarea Cubului Rubik, utilizând tehnici avansate de procesare a imaginilor.

Scopul principal al acestui proiect este de a dezvolta o aplicație care să preia imagini ale fețelor Cubului Rubik, să le proceseze pentru a identifica culorile și configurațiile și să ofere utilizatorilor pași clari și ușor de urmat pentru rezolvarea cubului. Prin utilizarea unor algoritmi de detectare a colțurilor, filtrare gaussiană și transformare proiectivă, aplicația asigură o analiză precisă și rapidă a imaginilor încărcate de utilizatori.

Prin această abordare, proiectul demonstrează aplicabilitatea practică a teoriilor și tehnicilor de procesare a imaginilor digitale. Alegerea Cubului Rubik ca subiect central permite nu doar evidențierea utilității acestor tehnologii într-un context popular și atractiv, dar și explorarea unor metode inovative de rezolvare automată a puzzle-urilor complexe.

Tehnologii folosite

- **C#** - întreaga parte de cod a acestei aplicații a fost realizată în limbajul C#, acoperind atât implementarea algoritmilor de prelucrare a imaginilor, cât și partea de rezolvare a cubului.

- **WPF**(Windows Presentation Foundation) - pentru a crea interfața destinată încărcării imaginilor de către utilizator.
- **Unity** - a fost folosit pentru dezvoltarea interfeței grafice care are scopul de a reprezenta cubul și de a permite utilizatorului să urmarească pașii de rezolvare a acestuia.

  Interfața de rezolvare a cubului a fost modificată pentru a reflecta configurația reală a cubului pe baza imaginilor, iar funcționalitatea de prezentare a rezolvării a fost și ea modificată pentru a permite urmărirea pașilor în mod detaliat. De asemenea a fost adaugată posibilitatea de a vizualiza informații utile despre mutarea următoare(culoare, număr de rotații și direcția rotației).
- **Github** - pentru a facilita colaborarea asupra proiectului, având în vedere că acesta a fost dezvoltat de două persoane. Prin intermediul platformei, am putut lucra simultan la aplicație, gestionând eficient versiunile, urmărind modificările și asigurând o sincronizare eficientă a codului.

Modul de Funcționare

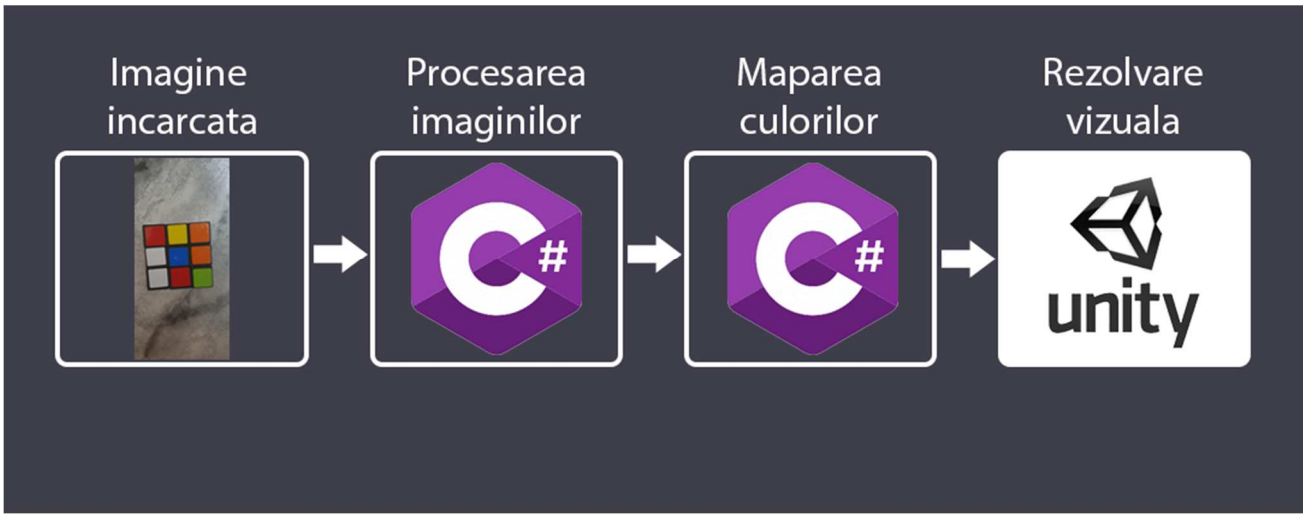

Fig. 1. Reprezentarea modului de funcționare al aplicației.
Autor: Edveș Alexandru.

Programul preia imaginile încărcate de utilizator și le supune unui proces detaliat de procesare a imaginilor pentru a analiza fațetele Cubului Rubik. Fiecare imagine a cubului trece prin următoarele faze de procesare:

- **Aplicarea Filtrului Gaussian:** Inițial, se aplică un filtru Gaussian pentru netezirea imaginii și reducerea zgomotului.
- **Detectarea Marginilor cu Canny:** Se utilizează Operatorul Canny pentru a detecta marginile cubului.
- **Detectarea Colțurilor cu Harris:** Se aplică Operatorul Harris pentru a identifica colțurile fațetelor cubului.
- **Filtrarea Colțurilor:** Se selectează colțurile care probabil formează conturul fiecărei fațe a cubului.
- **Corecția de Perspectivă și Decupare:** Se aplică corecții de perspectivă și se decupează fațetele cubului din imagine.

- **Binarizare pe Culori în Spațiul HSV:** Se aplică thresholding pe culori în spațiul HSV pentru fiecare față a cubului, pentru a separa culorile stickerelor.

La finalul acestor faze de procesare, sunt generate imagini binarizate pentru fiecare dintre cele șase culori ale cubului. Aceste imagini sunt organizate în șase foldere, câte unul pentru fiecare față a cubului, fiecare folder conținând câte șase imagini în tonuri de gri, reprezentând fiecare culoare detectată.

Algoritmi folosiți

Filtrul Gaussian

Nucleul Gaussian este generat de funcția **GenerateGaussianKernel**. Procesul include următorii pași:

1. Inițializarea unui nucleu ca o matrice bidimensională de dimensiunea **kernelSize** × **kernelSize**.
2. Calculul fiecărui element al nucleului folosind formula Gaussiană fără constanta de normalizare:

$$G(x,y) = e^{-0.5\left(\frac{(x-\text{mean})^2}{\sigma^2}+\frac{(y-\text{mean})^2}{\sigma^2}\right)}$$

   unde $\boldsymbol{x}$ și $\boldsymbol{y}$ sunt coordonatele fiecărui element în nucleu, **mean** este centrul nucleului, iar $\boldsymbol{\sigma}$ este deviația standard.
3. Normalizarea nucleului astfel încât suma tuturor valorilor sale să fie egală cu 1. Aceasta se realizează prin împărțirea fiecărui element al nucleului la suma totală a tuturor elementelor.

Nucleul Gaussian astfel generat și normalizat este utilizat pentru a aplica filtrul Gaussian asupra imaginii.

Canny pentru imagini color

A fost implementat algoritmul Canny pentru detectarea marginilor într-o imagine color. Procesul implică mai mulți pași cheie:

1. Aplicarea filtrului Gaussian pentru netezirea fiecărui canal de culoare al imaginii. Aceasta reduce zgomotul și pregătește imaginea pentru detectarea marginilor.
2. Calculul magnitudinii și direcției gradientului pentru fiecare pixel
3. Pasul de Non-maximum Suppression pentru a păstra doar contururile cele mai puternice și a elimina cele slabe sau false.
4. Aplicarea Hysteresis Thresholding pentru a stabili care margini să fie păstrate în imaginea finală.

Calculul Gradienților:

Se calculează gradientul fiecărui pixel din imagine. Gradientul este un vector care indică direcția celei mai mari variații a intensității și este calculat folosind operatorii Sobel. Acești operatori sunt matrici care detectează schimbările de intensitate în direcțiile orizontală și verticală.

Operatorii Sobel

Sunt folosite două matrici Sobel, una pentru detectarea schimbărilor pe orizontală (**sobelX**) și alta pentru verticală (**sobelY**):

$$\text{sobelX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \text{sobelY} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Calculul Magnitudinii și Direcției Gradientului:
Pentru fiecare pixel, metoda aplică aceste matrici pentru a obține valorile gradientului pe axele X și Y (gradientX, gradientY). Magnitudinea (**Magnitude**) și direcția (**Direction**) gradientului sunt calculate astfel:

$$\text{Magnitude} = \sqrt{\text{gradientX}^2 + \text{gradientY}^2}, \ \text{Direction} = \arctan 2\,(\text{gradientY}, \text{gradientX}) \times \frac{180}{\pi}$$

Harris

Funcția **DetectHarrisCorners** implementează algoritmul Harris pentru detectarea colțurilor într-o imagine.
Procesul General al Algoritmului

1. **Calculul gradientilor**: Se determină gradientii Ix și Iy ai imaginii.
2. **Produsul derivatelor**: Se calculează produsele gradientilor (Ix^2, Iy^2, IxIy).
3. **Netezirea**: Se aplică o netezire Gaussiană asupra fiecărui produs al derivatelor.
4. **Răspunsul Harris**: Se calculează un răspuns Harris pentru fiecare pixel, reprezentând potențialul acestuia de a fi un colț.
5. **Aplicarea pragului și suprimarea non-maximelor**: Se identifică punctele care depășesc un anumit prag și care sunt maxime locale, interpretate ca fiind colțuri ale imaginii.

Algoritmul Harris este eficient în detectarea colțurilor datorită sensibilității sale la variațiile de intensitate din diferite direcții.

Calculul Gradientilor
Se calculează gradientii pe axele X și Y (Ix și Iy) ai imaginii folosind operatorii Sobel. Aceasta este o etapă fundamentală pentru determinarea schimbărilor de intensitate în imagine, necesară pentru detectarea colțurilor.

Calculul Produselor Derivatelor
Sunt calculate pătratele valorilor gradientilor (Ix^2 și Iy^2) și produsul lor (IxIy). Aceste valori sunt folosite pentru a construi o matrice de autocorelație necesară în calculul răspunsului Harris.

Netezirea Gaussiană
Se aplică un filtru Gaussian asupra Ix^2, Iy^2 și IxIy pentru a netezi variațiile bruște de intensitate. Această netezire ajută la stabilizarea detectării colțurilor în prezența zgomotului.

Calculul Răspunsului Harris (CRF)
Se evaluează potențialul fiecărui pixel de a fi un colț. Se folosește formula:

$$\text{Interpolated Gradient} = \text{grad1} \times (1 - \text{weight}) + \text{grad2} \times \text{weight}$$

unde M este matricea de autocorelație a gradientilor, **det(M)** este determinantul acestei matrice, iar **trace(M)** este urma (suma elementelor de pe diagonala principală). Parametrul *kk* este un factor de sensibilitate.

Aplicarea Thresholding și Non-Maximum Suppression
După calculul răspunsului Harris, se aplică un prag pentru a identifica punctele semnificative care pot fi colțuri. Acestea sunt punctele cu un răspuns Harris mai mare decât o valoare de prag specificată. De asemenea, se aplică Non-Maximum Suppression pentru a asigura că în vecinătatea unui colț puternic nu sunt selectate alte puncte ca fiind colțuri.

Verificarea Maximului Local și a Răspunsului Cel Mai Puternic în Vecinătate
Această etapă este utilă pentru a verifica dacă un pixel este maxim local și dacă are cel mai puternic răspuns într-o anumită vecinătate. Aceasta asigură că punctele selectate ca colțuri sunt distincte și bine definite.

Filtrarea Colțurilor Feței Cubului Rubik
Se filtrează colțurile detectate de către operatorul Harris pentru a identifica cele patru care pot forma fața unui Cub Rubik. Procesul include următorii pași:

1. Verificarea dacă sunt cel puțin patru colțuri detectate de operatorul Harris.
2. Calcularea combinărilor posibile de patru colțuri și evaluarea fiecăreia pentru a determina cea mai probabilă față a cubului.
3. Pentru fiecare grup de patru puncte, se calculează aria, aspect ratio și consistența unghiurilor.
4. Se alege combinația cu cea mai mare arie și scorul cel mai mic bazat pe aspect ratio și consistența unghiurilor.

Calculul Aspect Ratio
Această etapă determină raportul de aspect al unui cvadrilater format din patru colțuri. Aceasta măsoară lungimea medie a laturilor opuse și calculează raportul dintre lățime și înălțime. Un raport de aspect apropiat de 1 indică o formă pătratică, așteptată în cazul unei fețe de cub.

Calculul Consecvenței Unghiurilor
Se evaluează cât de apropiate sunt unghiurile cvadrilaterului de 90 de grade. Acest lucru ajută la identificarea cvadrilaterelor cu unghiuri drepte, care sunt mai susceptibile de a forma o față a cubului Rubik.

Calculul Ariei Cvadrilaterului
Este calculată aria unui cvadrilater folosind coordonatele celor patru colțuri. O arie mai mare sugerează un cvadrilater mai probabil să reprezinte o față a cubului.

Ordonarea Colțurilor
Se aranjează colțurile unui cvadrilater într-o ordine specifică (sus-stânga, sus-dreapta, jos-dreapta, jos-stânga). Aceasta facilitează calculele ulterioare, cum ar fi corecția de perspectivă și decuparea imaginii.

Transformarea proiectivă

Prezentare
Acest algoritm este dedicat corectării perspectivei unei fețe a cubului. Beneficiind de punctele determinate în algoritmul anterior, acesta efectuează o rotație și extrage exclusiv fața cubului, eliminând detaliile din fundal. Astfel, se obține o imagine centrată și corectată a feței respective, contribuind la precizia și coerența procesului de identificare și interpretare a fețelor cubului.

Implementarea algoritmului

Se transformă fiecare listă de puncte într-o matrice. Matricea este calculată după formula:

$$\mathrm{P} = \begin{bmatrix} lista[0].X & lista[1].X & lista[2].X \\ lista[0].Y & lista[1].Y & lista[2].Y \\ 1 & 1 & 1 \end{bmatrix}$$

Calcularea vectorilor de ponderi:
Datorită faptului că cele două pătrate, delimitate de cele două liste de puncte, sunt nedegenerate, primele trei puncte din fiecare listă (**lista[0]**, **lista[1]**, **lista[2]**) sunt liniar independente. Din această afirmație rezultă că matricile P și P' sunt inversabile. Astfel, vectorii de ponderi b și b' pot fi calculați după următoarele formule:

$$b = P{-}1 * P4$$
$$b' = (P'){-}1 * P4'$$

Unde $P4$ reprezintă matricea:

$$P_4 = \begin{bmatrix} lista[3].X \\ lista[3].Y \\ 1 \end{bmatrix}$$

Compunerea matricei de transformare:
Prin această matrice de transformare A, dorim maparea patrulaterului determinat de punctele din imaginea inițială pe patrulaterul determinat de punctele din a doua listă în imaginea rezultat. Această matrice se poate calcula după formula:

$$A = h_4 * \left(\frac{b'_1}{b_1} P'_1 \; \frac{b'_2}{b_2} P'_2 \; \frac{b'_3}{b_3} P'_3 \right) P^{-1}$$

Unde $h4 \neq 0$ este arbitrar. De obicei se consideră $h4=1$.

Maparea punctelor:
Fiecare pixel din imaginea sursă (x, y), considerat în coordonate omogene $(x,y,1)T(x,y,1)T$ se mapează pe un pixel (x', y') din imaginea rezultat astfel:

$$\begin{bmatrix} \hat{x'} \\ \hat{y'} \\ h \end{bmatrix} = A * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

De unde, prin împărțirea la h, se obțin coordonatele din imaginea rezultat $x'=x'/h$, $y'=y'/h$ pe care se va plasa pixelul din imaginea sursă aflat la coordonatele (x, y). Prin transformarea directă de la imaginea sursă către imaginea rezultat, o parte din pixeli nu vor avea corespondent. O soluție la această problemă este să pornim de la imaginea rezultat către imaginea sursă folosind transformarea inversă, dată de $A-1A-1$. Însă, în loc să calculăm matricea A și după să o inversăm, putem utiliza toate calculele de mai sus, dar schimbăm semnificația punctelor care determină vârfurile dreptunghiului în modul următor:

- În loc să considerăm $P1$, $P2$, $P3$,$P4P1$,$P2$,$P3$,$P4$ din imaginea sursă și $P1'$,$P2'$,$P3'$,$P4'P1'$,$P2'$,$P3'$,$P4'$ din imaginea rezultat, vom considera invers ($P1$,$P2$,$P3$,$P4P1$,$P2$,$P3$,$P4$ din imaginea rezultat și $P1'$,$P2'$,$P3'$,$P4'P1'$,$P2'$,$P3'$,$P4'$ din imaginea sursă).
- Prin această metodă matricea A va reprezenta exact transformarea inversă.

Procesul de Thresholding pe Culori și Obținerea Șirului de Culori

Acest algoritm aplică binarizarea pe culori pentru toate fețele cubului Rubik, separând fiecare culoare a stickerelor. În primul rând, imaginile fețelor cubului sunt convertite în spațiul de culori HSV. Apoi, pentru fiecare interval de culoare HSV definit, se aplică thresholding pentru a crea măști binare care separă culorile.

Un caz special este tratat pentru culoarea roșie, care necesită două imagini datorită poziției sale în spațiul HSV. Aceste imagini sunt combinate pentru a crea o singură mască binară reprezentând culoarea roșie.

După ce toate măștile de culoare sunt generate, se parcurg și se împart în grile de 3x3. Fiecare grilă este analizată pentru a determina dacă are mai mulți pixeli albi decât negri, corespunzând prezenței unei anumite culori. Această informație este transformată într-un șir de litere, fiecare literă reprezentând inițiala unei culori. Șirurile pentru fiecare față a cubului sunt apoi combinate pentru a crea un șir complet care reflectă aranjamentul culorilor pe cub.

Rezultatele Procesului de Prelucrare a Imaginilor

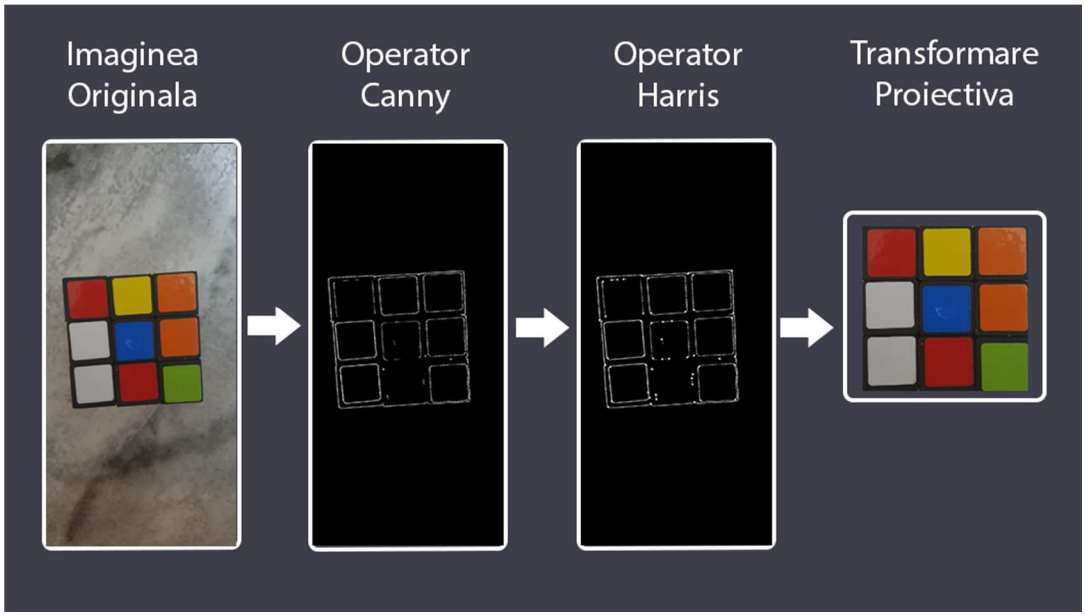

Fig. 2. Rezultatul procesării imaginilor.
Autor: Cîrstea Andrei.

Concluzii

În cadrul acestui proiect, am abordat dezvoltarea unei aplicații complexe de procesare a imaginilor pentru rezolvarea unui Cub Rubik. Prin intermediul acestui proiect, am consolidat și aplicat cunoștințe fundamentale din domeniul procesării imaginilor digitale.

Am implementat cu succes un sistem care detectează și extrage fețele cubului din imaginile furnizate de utilizator, urmat de aplicarea unor praguri pentru identificarea culorilor. De asemenea, am adaptat o interfață grafică pentru afișarea soluțiilor, care a contribuit la o mai bună interactivitate și ușurință în utilizarea aplicației.

Prin acest proiect, am demonstrat cum teoria procesării imaginilor poate fi aplicată într-un context practic și util. Rezultatele obținute validează abordările teoretice studiate și deschid calea pentru îmbunătățiri ulterioare. Acest proiect a fost, de asemenea, un mod interactiv de a învăța algoritmi de procesare a imaginilor.

Bibliografie

I. C. Plajer, Procesarea digitală a imaginilor, 2023.

1]

2020. [Interactiv]. Available: https://www.youtube.com/watch?v=JN9vx0veZ-c. [Accesat 2024]. -

2] Adaptat nevoilor proiectului