UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Detecția muchiilor din imagini color

Procesarea Imaginilor

Autor: Marcu Ariana-Mălina
Grupa: 30232/2

Facultatea de Automatică
și Calculatoare

4 Iunie 2024

Cuprins

1 Introducere **2**
1.1 Context 2
1.2 Problemele ce necesită rezolvare 2
1.3 Obiectivele propuse 2

2 Fundament Teoretic **3**
2.1 Studiu bibliografic 3

3 Proiectare și Implementare **4**
3.1 PASUL 1 - Filtrarea imaginii cu un filtru Gaussian pentru eliminarea zgomotelor 4
3.2 PASUL 2 - Calculul modulului și direcției gradientului 5
3.3 PASUL 3 - Suprimarea non-maximelor modulului gradientului 6
3.4 PASUL 4 - Binarizarea adaptivă a punctelor de muchie și prelungirea muchiilor prin histereză 6

4 Rezultate **7**

5 Concluzie **9**

6 Bibliografie **10**

1 Introducere

1.1 Context

În domeniul procesării imaginilor digitale, detecția muchiilor reprezintă un pas fundamental în prelucrarea și analiza imaginilor. Muchiile sunt zone dintr-o imagine unde intensitatea pixelilor se schimbă brusc și indică limitele obiectelor din cadrul imaginii. Identificarea corectă a acestor muchii este esențială pentru diverse aplicații, cum ar fi recunoașterea obiectelor, segmentarea imaginilor, și îmbunătățirea clarității vizuale.

1.2 Problemele ce necesită rezolvare

Detecția muchiilor în imagini color prezintă provocări suplimentare față de imagini alb-negru, datorită complexității informației de culoare și a zgomotului prezent în imagini. Problemele principale ce trebuie abordate includ:

- Gestionarea eficientă a informației de culoare pentru a păstra detaliile importante.
- Reducerea zgomotului din imagine fără a afecta muchiile reale.
- Identificarea precisă a muchiilor în prezența variabilității în iluminare și textură.
- Implementarea unei metode robuste și eficiente din punct de vedere computațional pentru detecția muchiilor.

1.3 Obiectivele propuse

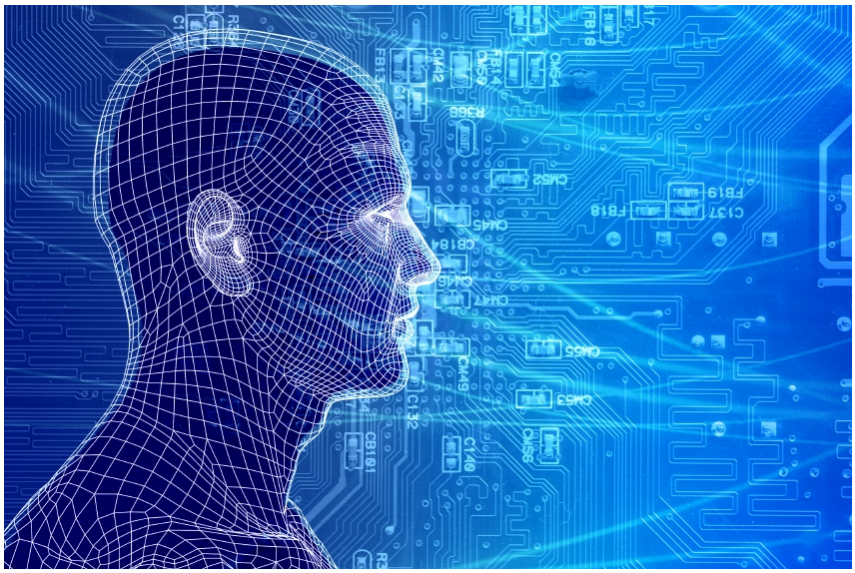
Proiectul propune studierea și implementarea unei metode eficiente pentru detecția muchiilor în imagini color, bazată pe algoritmul Canny. Obiectivele specifice ale proiectului sunt:

- implementarea unei soluții care poate procesa imagini BMP color cu 24 de biți per pixel.

- adaptarea algoritmului Canny pentru a funcționa optim cu imagini color.

- evaluarea performanței metodei implementate prin compararea rezultatelor obținute pe un set vast de imagini de test.

- asigurarea că soluția finală este eficientă din punct de vedere al timpului de execuție și al resurselor utilizate, fiind totodată ușor de utilizat și integrat în alte aplicații de procesare.



2 Fundament Teoretic

2.1 Studiu bibliografic

Detecția muchiilor este un subiect bine documentat în literatura de specialitate, iar diverse metode au fost propuse și dezvoltate de-a lungul timpului. Algoritmul Canny, propus de John F. Canny în 1986, este unul dintre cei mai renumiți algoritmi pentru detecția muchiilor datorită performanței sale și a robusteții în prezența zgomotului.

Printre sursele relevante pentru acest proiect se numără:

i. **Canny, J. (1986). A Computational Approach to Edge Detection.** Acest articol definește baza teoretică a algoritmului Canny și oferă o descriere detaliată a pașilor necesari pentru implementarea sa. [5] Principalele metode folosite și descrise în articol sunt:

- filtrarea gaussiană: Aplicarea unui filtru Gaussian pentru netezirea imaginii și reducerea zgomotului.

- calculul gradientului: identificarea punctelor de muchie prin determinarea gradientului imaginii normalizate.

- non-supresia maximelor: Subțierea muchiilor prin păstrarea numai a maximelor locale ale gradientului.

- histereza: Utilizarea a două praguri pentru a detecta muchiile reale și pentru a elimina zgomotul, asigurând continuitatea muchiilor. Aceste metode sunt combinate pentru a crea un detector de muchii robust și precis, optimizat prin criterii de detecție, localizare și răspuns unic la muchii.

ii. **Gonzalez, R. C., & Woods, R. E. (2002). Digital Image Processing.** Acest manual oferă o viziune largă asupra diverselor metode de prelucrare a imaginii, inclusiv detecția muchiilor și tehnici de reducere a zgomotului. [6] Principalele metode descrise:

* operatorii de derivare: operatorii Sobel, Prewitt și Roberts: metode de calculare a gradientului imaginii pentru identificarea muchiilor prin detectarea schimbărilor bruște de intensitate.

* transformata Hough: utilizată pentru detectarea formelor geometrice, precum linii și cercuri, în imagini.

* filtre spațiale și frecvențiale

    - Gaussiane: Pentru reducerea zgomotului și netezirea imaginii.

    - Laplaciene: Pentru evidențierea muchiilor prin calcularea celei de-a doua derivate a imaginii.

* metode de segmentare:

    - Thresholding (prag): Separarea obiectelor de fundal pe baza nivelurilor de intensitate.

    - segmentare bazată pe contururi și regiuni: Identificarea obiectelor pe baza contururilor și similarităților între regiuni de pixeli.

Cartea explică nu doar metodele de bază, ci și modul în care acestea pot fi combinate și optimizate pentru diferite aplicații în prelucrarea imaginii. Gonzalez și Woods oferă exemple practice și studii de caz pentru a ilustra aplicabilitatea acestor metode în probleme reale.

iii. **Sonka, M., Hlavac, V., & Boyle, R. (2008). Image Processing, Analysis, and Machine Vision.** este o altă lucrare care prezintă o varietate de metode de prelucrare a imaginii și discuții aprofundate despre algoritmii de detecție a muchiilor.[7] E dedicată prelucrării și analizei imaginii, precum și viziunii artificiale.

Pe scurt, principalele subiecte abordate în această carte ar fi următoarele: se prezintă conceptele fundamentale ale prelucrării digitale a imaginilor, inclusiv formatele de imagini și operațiile de bază asupra pixelilor, se discută tehnici de transformare și filtrare a imaginilor, cum ar fi transformata Fourier, filtrarea spațială și frecvențială, pentru îmbunătățirea și analiza imaginilor, metode de segmentare a imaginii, inclusiv tehnici bazate pe praguri, regiuni și contururi, pentru separarea obiectelor de fond, descrierea și reprezentarea formelor. Se discută tehnici de recunoaștere a obiectelor folosind caracteristici extrase din imagini și metode de clasificare, inclusiv rețele neuronale și alte tehnici de machine learning, etc.

3 Proiectare și Implementare

În realizarea acestui proiect, am ales să studiez și să implementez metoda detecției muchiilor ce are la bază algoritmul Canny. Principala sursă de ajutor mi-a fost chiar laboratorul 11 [4], ai cărui pași i-am urmat exact în implementarea mea.

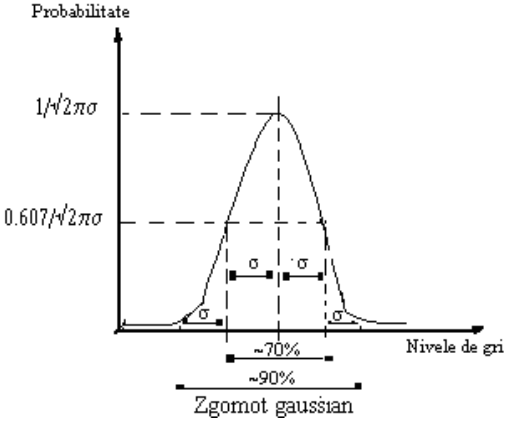
Soluția aleasă constă în aplicarea unui filtru gaussian pentru reducerea zgomotului în imaginea de intrare, urmată de detectarea muchiilor folosind operatorul Sobel și procesarea acestora pentru a fi evidențiate. Structura codului(diagrama bloc), în cazul meu, constă în două funcții principale și una ajutătoare: una în care realizez filtrarea gaussiană, a doua în care implementez restul funcționalităților necesare, iar cea ajutătoare transformă un array într-o matrice/imagine.

Pentru o prezentare mai clară, mă voi folosi de pașii din laborator, care sunt funcționali pentru imagini de tip grayscale, ceea ce face ca primul pas să fie transformarea imaginii sursă, care este una color BMP cu 24 de biţi/pixel, într-una grayscale.

3.1 PASUL 1 - Filtrarea imaginii cu un filtru Gaussian pentru eliminarea zgomotelor

Eliminarea zgomotului gaussian trebuie făcută cu un filtru având o formă și o dimensiune adecvate, în concordanță cu deviația standard sigma a zgomotului gaussian care afectează imaginea. Dimensiunea w unui astfel de filtru se alege de obicei de 6*sigma. Construcția elementelor unui astfel de nucleu/filtru gaussian G, se face cu formula de mai jos(pagina 70 din îndrumător):

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}}$$



* x0 și y0 reprezintă coordonatele coloanei și liniei centrale a nucleului

Funcția mea *filtrare_gaussiana* implementează un filtru Gaussian pe imagini grayscale. Mai întâi, deschide un fișier de imagine și încarcă imaginea color. Apoi convertește imaginea color într-o imagine grayscale prin calcularea mediei componentelor RGB pentru fiecare pixel. Creează și inițializează matrici pentru nucleul Gaussian și componentele sale 1D, calculând sigma pe baza lățimii nucleului. Populează matricea Gaussiană folosind formula specifică și extrage componentele 1D necesare pentru convoluții separabile. Aplică prima convoluție între imaginea grayscale și nucleul 1D vertical, normalizează rezultatul și aplică a doua convoluție între rezultatul anterior și nucleul 1D orizontal, obținând imaginea finală filtrată. În final, afișează imaginea grayscale și imaginea filtrată.

3.2 PASUL 2 - Calculul modulului și direcției gradientului

Calcularea modulului și direcției gradientului presupune alocarea a câte unui buffer temporar de dimensiunea imaginii și inițializarea elementelor lor în conformitate cu formulele de mai jos, unde componentele orizontale fx(x,y) și respectiv verticale fy(x,y) se pot calcula prin convoluția imaginii cu nucleul Sobel, pe care l-am declarat astfel:

Modul: $|\nabla f(x,y)| = \sqrt{(\nabla f_x(x,y))^2 + (\nabla f_y(x,y))^2}$

Direcție: $\theta(x,y) = arctg\left(\frac{\nabla f_y(x,y)}{\nabla f_x(x,y)}\right)$

$$\nabla f_x = f(x,y) * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\nabla f_y = f(x,y) * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

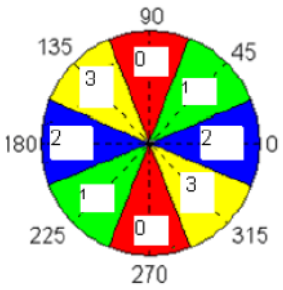
```
int sobel_x[3][3] = { -1,0,1,-2,0,2,-1,0,1 };
int sobel_y[3][3] = { 1,2,1,0,0,0,-1,-2,-1 };
```

Chiar în prima parte a funcției următoare, numită *DetectieMuchii*, se realizează acest pas, aplicând formulele prezentate mai sus și normalizând valorile în intervalul [0..255] (prin împărțire la 4 radical din 2 în cazul în care gradientul a fost calculat utilizând operatorul Sobel), iar apoi transform din float în uchar ca să pot verifica rezultatul:

```
Mat_<float> SobelX = convertArrayToMat(sobel_x);
Mat_<float> SobelY = convertArrayToMat(sobel_y);
for (int i = 0; i < img.rows; i++)
    for (int j = 0; j < img.cols; j++)
        float gradientX = 0, gradientY = 0;
        for (int m = 0; m < SobelX.rows; m++)
           for (int n = 0; n < SobelX.cols; n++)
                if (isInside(img.rows, img.cols, i + m - SobelX.rows / 2,
        j + n - SobelX.cols / 2))
                gradientX += (float)img.at<uchar>(i + m - SobelX.rows / 2,
        j + n - SobelX.cols / 2) * SobelX(m, n);
                gradientY += (float)img.at<uchar>(i + m - SobelX.rows / 2,
        j + n - SobelX.cols / 2) * SobelY(m, n);
        modul(i, j) = sqrt(pow(gradientX, 2) + pow(gradientY, 2)) / (4 * sqrt(2));
        directie(i, j) = atan2(gradientY, gradientX);
minMaxLoc(modul, &minVal, &maxVal);
modul.convertTo(modulNormalizat, CV_8UC1, 255.0 / maxVal);
```

3.3 PASUL 3 - Suprimarea non-maximelor modulului gradientului

Are drept scop subțierea muchiilor prin păstrarea doar a punctelor de muchie care au modulul maxim al gradientului de-a lungul direcției de variație a intensității (de-a lungul direcției gradientului). Primul pas constă în cuantificarea direcțiilor gradientului, pe care le-am calculat folosind figura de mai jos:



Mai întâi, codul meu convertește direcțiile gradientului fiecărui pixel din radiani în grade și ajustează valorile negative pentru a le aduce în intervalul 0-180 grade. Apoi, pentru fiecare pixel, determină direcția marginilor (orizontală, verticală sau diagonală) și compară magnitudinea gradientului pixelului curent cu magnitudinile pixelilor adiacenți din direcția respectivă. Dacă magnitudinea pixelului curent este mai mare sau egală cu cele ale pixelilor adiacenți, pixelul este păstrat; în caz contrar, este setat la zero. Aceasta ajută la eliminarea pixelilor care nu fac parte din marginea reală, rezultând margini mai clare și mai precise.

3.4 PASUL 4 - Binarizarea adaptivă a punctelor de muchie și prelungirea muchiilor prin histereză

După efectuarea cu succes a calculului gradientului și a operației de suprimare a nonmaximelor modulului gradientului, se obține o imagine în care intensitatea pixelilor este egală cu valoarea modulului gradientului în acel punct. De asemenea, grosimea punctelor de muchie din această imagine este de un pixel. În continuare se descriu pașii pentru obținerea muchiilor finale.

Binarizarea adaptivă își propune să extragă un număr constant de puncte de muchie dintr-o imagine, compensând variațiile de iluminare și contrast. Un prag fix poate fie să scoată prea multe muchii, fie să nu scoată deloc. Parametrul principal, p, variază între 0.01 și 0.1 și reprezintă raportul dintre punctele de muchie și punctele cu gradient nenul. Algoritmul:

- se calculează histograma gradientului de intensități după suprimarea non-maximelor, normalizată în intervalul [0, 255] (împărțind la 42 pentru operatorul Sobel), obținându-se Hist[i], unde Hist[i] reprezintă numărul pixelilor cu gradientul normalizat de modul i.

- se calculează numărul de pixeli nenuli care nu sunt puncte de muchie: NrNonMuchie = (1-p)*(Height*Width–Hist[0]).

- se parcurge histograma de la poziția 1 spre 255, însumând valorile. Când suma depășește NrNonMuchie, poziția curentă devine pragul adaptiv (PragAdaptiv), indicând intensitatea sub care se află NrNonMuchie pixeli.

Extinderea muchiilor prin histereză este o tehnică utilizată pentru completarea muchiilor detectate, care pot fi fragmentate sau incomplete din cauza umbrelor sau a zgomotului. Se stabilesc două praguri: un prag înalt și unul coborât, care este un procent (de exemplu, k = 0.4) din pragul înalt. Imaginea gradientului este parcursă pixel cu pixel și se marchează pixelii cu intensitate mai mare decât pragul înalt ca muchii tari, iar cei cu intensitate mai mare decât

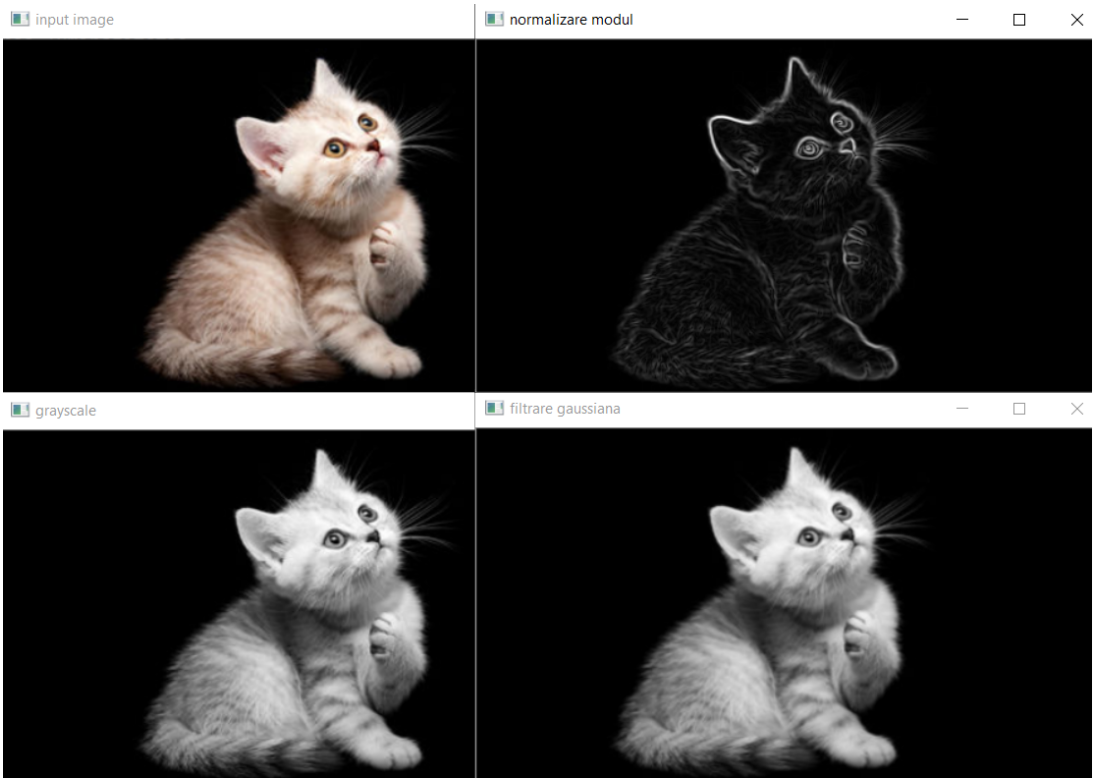
pragul coborât, dar mai mică decât pragul înalt, ca muchii slabe. Se elimină pixelii cu intensitate mai mică decât pragul coborât.

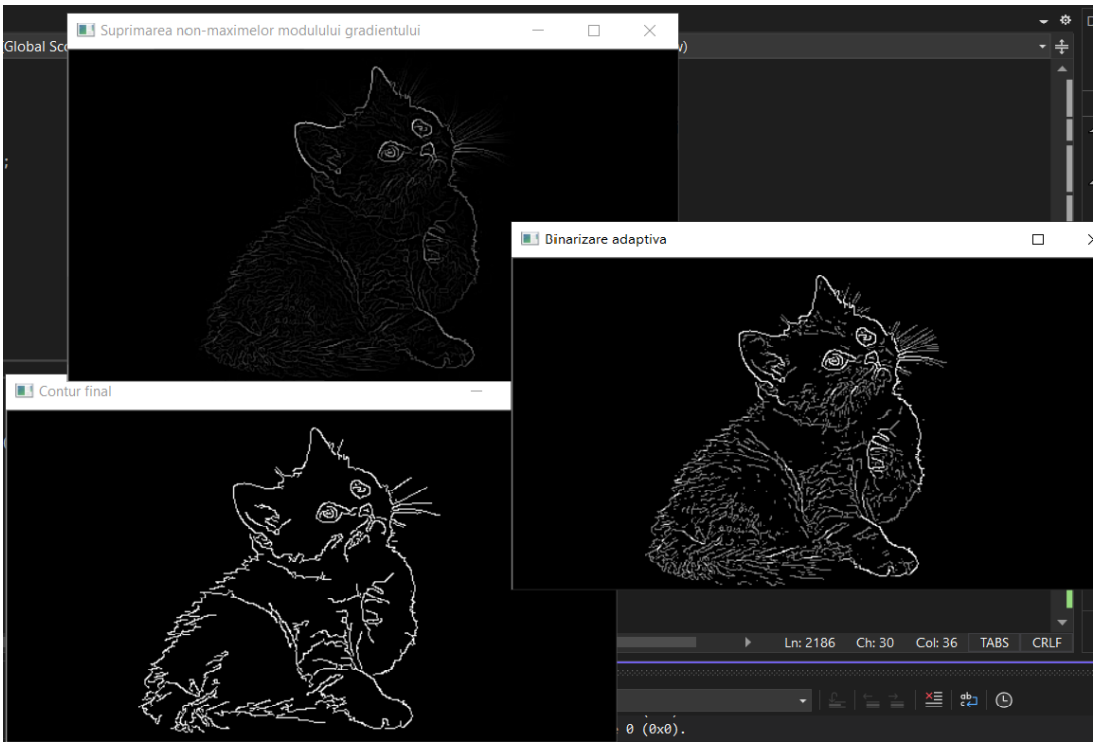
Ulterior, muchiile tari sunt extinse prin includerea pixelilor vecini slabi. Acest proces se realizează utilizând o coadă pentru a reține coordonatele muchiilor tari. Se parcurge imaginea și se identifică inițial primul punct de muchie tare, coordonatele sale fiind adăugate în coadă. Apoi, se extind muchiile tari prin includerea vecinilor slabi, iar aceștia devin și ei considerați ca muchii tari. Procesul se repetă până când nu mai există puncte de muchie tare învecinate cu puncte de muchie slabă.

În final, punctele de muchie slabă sunt eliminate din imagine. Este important să se definească conceptul de "învecinare" între un punct de muchie tare și unul slab, care poate implica pixeli adiacenți, vecinătate de 4 sau de 8, sau o distanță mai mare, pentru a ține cont de posibile întreruperi ale muchiilor din cauza zgomotului.

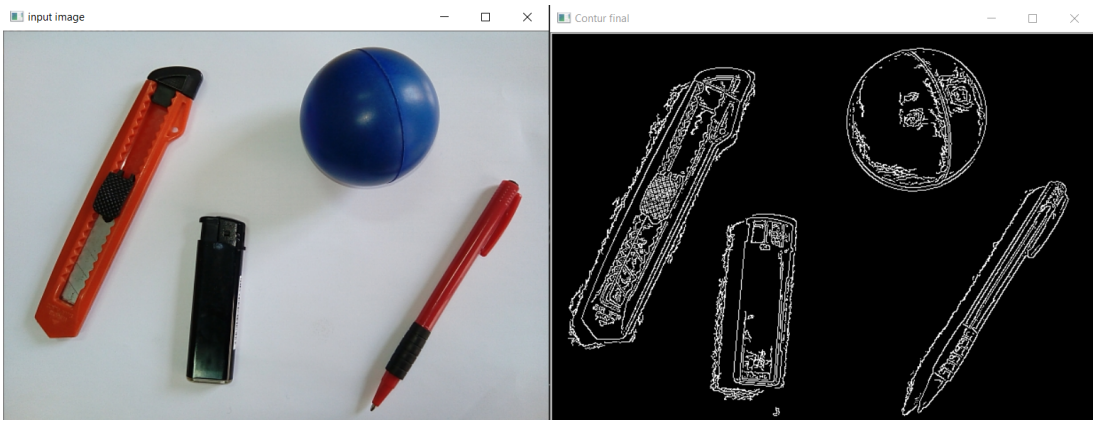
4 Rezultate

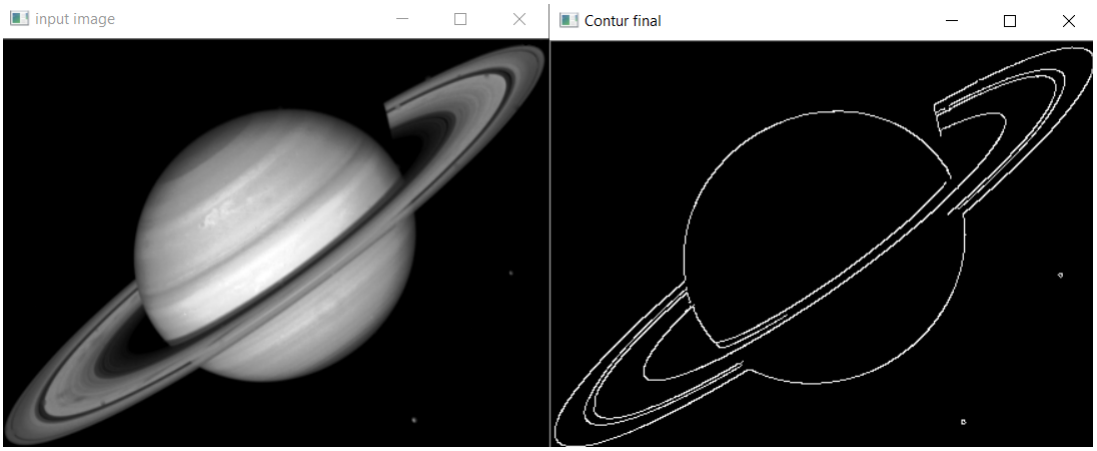
După o testare amănunțită, am reușit să obțin rezultate bune, care sa ofere un contur satisfăcător imaginilor atât color cât și grayscale. Mai jos am atașat unul din rezultatele testelor mele, în care este ilustrat fiecare pas al algoritmului, pe rând. Aici, am ales w=3, sigma=0.5, p=0.1 și k=0.4.

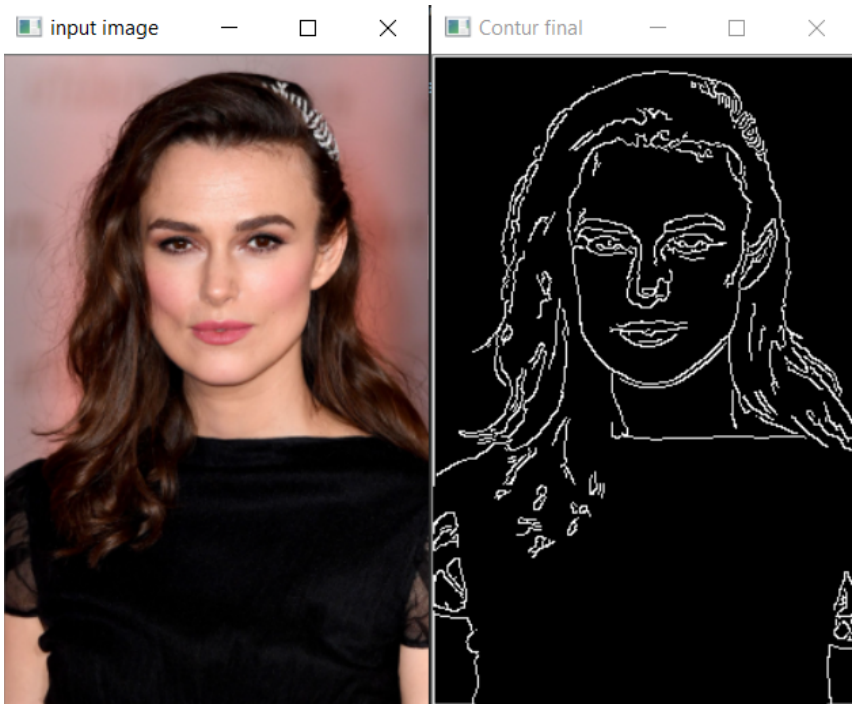




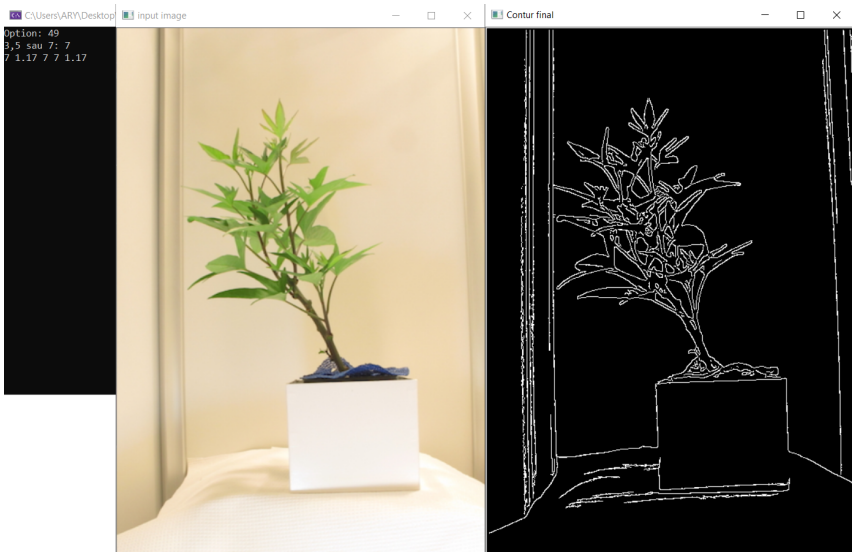
Mai jos am atașat mai multe din testele mele, pentru mai multe tipuri de imagini,chiar și cu umbre, pentru a dovedi că rezultatele pe care le-am obținut sunt unele valide. Totodată, o dovadă a corectitudinii este și testarea pe imaginea aflată în laborator la pagina 81.







În cazul în care creștem sigma, se va observa că scade si claritatea conturului:



5 Concluzie

În concluzie, implementarea funcțiilor a permis realizarea unei serii de operațiuni complexe în prelucrarea imaginilor. Prin aplicarea filtrării gaussiene, s-a obținut o imagine prelucrată cu reducerea zgomotului și a detaliilor nedorite. Apoi, prin algoritmul de detectare a marginilor bazat pe operatorul Sobel și tehnica non-maximelor, am reușit să evidențiez eficient contururile obiectelor din imagine. Aceste operațiuni au fost realizate cu succes însă, pentru viitor, se pot explora metode de optimizare a performanței algoritmilor și extinderea funcționalității pentru a aborda diverse cerințe de prelucrare a imaginilor, deoarece, comparând cu algoritmul deja existent în OpenCV, Canny, se obervă niște diferențe față de implementarea mea.

6 Bibliografie

[1] https://www.geeksforgeeks.org/real-time-edge-detection-using-opencv-python/

[2] https://learnopencv.com/edge-detection-using-opencv/

[3] https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

[4] Laboratoarele 9, 10 și 11, împreună cu imaginile de test, de pe site-ul domnului profesor: https://users.utcluj.ro/~rdanescu/teaching_pi.html

[5] Canny, J. (1986). A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6), 679-698. https://www.researchgate.net/publication/224377985_A_Computational_Approach_To_Edge_Detection

[6] Gonzalez, R. C., & Woods, R. E. (2002). Digital Image Processing. Prentice Hall. https://www.researchgate.net/publication/333856607_Digital_Image_Processing_Second_Edition

[7] Sonka, M., Hlavac, V., & Boyle, R. (2008). Image Processing, Analysis, and Machine Vision. Cengage Learning. https://link.springer.com/book/10.1007/978-1-4899-3216-7

[8] https://justin-liang.com/tutorials/canny/#:~:text=Canny%20edge%20detection%20is%20a%20image%20processing%20method,Thresholding%20Step%206%20-%20Edge%20Tracking%20by%20Hysteresis