FMI FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

PRELUCRAREA IMAGINILOR

OPRIȘAN ELENA-GRAȚIELA

Interfața

Proiectul implementează, în Python, toți algoritmii studiați în cadrul laboratorului, precum și temele din acest semestru.

Biblioteci Utilizate.

- **Tkinter**: crearea interfeței grafice.
- **OpenCV**: operații de procesare a imaginilor.
- **NumPy**: operații numerice eficiente.
- **Matplotlib**: generarea vizualizărilor și graficelor.

Interfața grafică a fost realizată folosind biblioteca **Tkinter**, oferind utilizatorului o experiență vizuală plăcută și interactivă. Elementele cheie ale interfeței sunt:

- **Canvas cu gradient de fundal**: Un efect vizual plăcut este obținut prin desenarea unui gradient din albastru deschis spre albastru închis.
- **Frame-uri pentru organizare**: Funcționalitățile sunt împărțite în două secțiuni principale:
  - **Laboratoare**: implementările algoritmilor studiați.
  - **Teme**: aplicațiile practice realizate în cadrul semestrului.
- **Butoane interactive**: Fiecare laborator sau temă are asociat un buton, cu o denumire clară și design uniform. Butoanele sunt colorate și stilizate pentru a fi ușor de identificat.
- **Dialoguri grafice de selectare a imaginilor**: Utilizatorul poate selecta cu ușurință imaginile din sistemul său de fișiere.

Structura și Organizarea Codului. Codul este structurat modular pentru a asigura o utilizare eficientă și o întreținere ușoară:

(1) **Funcții de laborator**: Fiecare laborator este implementat ca o funcție independentă, care realizează o anumită prelucrare a imaginilor. Exemple includ:
   - conversia în nuanțe de gri și binarizarea,
   - histogramă și egalizare,
   - aplicarea filtrelor spațiale și frecvențiale,
   - analiza componentelor conectate.

(2) **Funcții pentru teme**: Aceste funcții sunt mai complexe, implementând algoritmi specifici cerințelor tematice, cum ar fi:
   - detectarea pragurilor ,
   - corecția luminozității și contrastului,
   - aplicarea filtrelor Gaussiene și bidimensionale,

Facultatea de Matematică şi Informatică, Anul 3, Secţia Informatică
Adresa e-mail: elena.oprisan03@e-uvt.ro.

(3) **Funcții auxiliare**: Pentru reutilizare și modularitate, sunt implementate funcții generale, precum:
   - selectarea imaginilor,
   - afișarea rezultatelor folosind OpenCV,
   - salvarea imaginilor procesate.

(4) **Legarea interfeței cu funcțiile de procesare**: Fiecare buton din interfață este asociat unei funcții corespunzătoare prin metoda `command`.

Fluxul de Lucru al Aplicației. Fluxul principal al aplicației este următorul:

(1) La deschiderea aplicației, utilizatorul este întâmpinat de o fereastră cu două secțiuni: **Laboratoare** și **Teme**.
(2) Fiecare laborator sau temă poate fi accesat printr-un buton specific.
(3) Odată apăsat un buton, utilizatorului i se cere să selecteze o imagine folosind un dialog grafic.
(4) Procesul selectat este aplicat imaginii, iar rezultatele sunt afișate pe ecran.

Concluzii și Caracteristici Distincte. Această aplicație oferă o platformă interactivă pentru explorarea și aplicarea algoritmilor de prelucrare a imaginilor, fiind un instrument util atât pentru învățare, cât și pentru demonstrații practice. Caracteristicile sale cheie includ:

- **Interfață intuitivă**: ușor de utilizat de către studenți și utilizatori cu cunoștințe limitate în domeniul procesării imaginilor.
- **Flexibilitate și extensibilitate**: permite adăugarea de noi funcții sau algoritmi fără a afecta structura generală.
- **Vizualizare detaliată**: imaginile procesate și graficele sunt afișate clar și pot fi explorate simultan.
- **Automatizare parțială**: rezultatele sunt salvate automat, evitând pierderile accidentale de date.

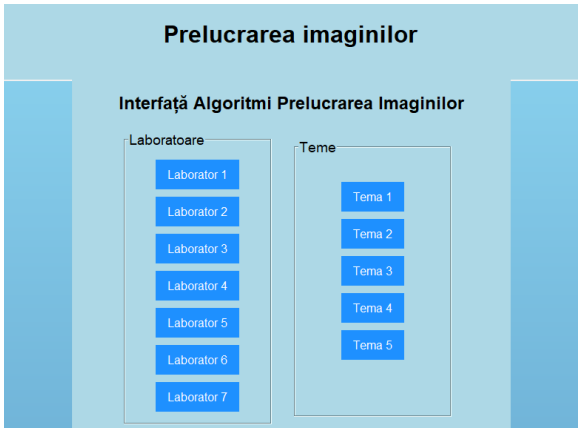


FIGURA 1. Interfata utilizator

1. LABORATOR 1

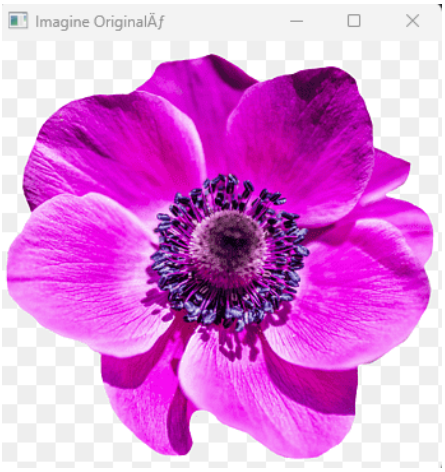
Funcția `laborator_1` este concepută pentru a afișa o imagine originală selectată de utilizator utilizând biblioteca `OpenCV`. Procesul începe cu selecția imaginii prin

funcția `select_image()`. Dacă utilizatorul nu selectează nicio imagine, funcția notifică prin mesajul „Nicio imagine NU e selectată.” și oprește execuția.

Imaginea selectată este încărcată folosind `cv2.imread()`. Dacă imaginea nu este validă sau nu poate fi găsită, se afișează mesajul „Imaginea nu există sau formatul este invalid.” și execuția este oprită. Dacă imaginea este validă, aceasta este afișată într-o fereastră utilizând `cv2.imshow()` sub titlul „Imagine Originală”. Programul așteaptă o intrare de la tastatură pentru a continua. La apăsarea tastei `ESC`, toate ferestrele sunt închise cu `cv2.destroyAllWindows()`.

```
def laborator_1():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine NU e selectata.")
        return

    img = cv2.imread(image_path)
    if img is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return
    cv2.imshow('Imagine Originala', img)
    key = cv2.waitKey(0)
    if key == 27:
        cv2.destroyAllWindows()
```

Listing 1. Funcția laborator_1



(a) Imaginea originală

Figura 2. Output funcție laborator_1

2. Laborator 2

Funcția `laborator_2` permite procesarea unei imagini selectate de utilizator, aplicând diferite transformări și afișând rezultatele în ferestre separate. De asemenea, generează histograme care descriu distribuția pixelilor imaginii. Imaginile procesate sunt salvate într-un director specificat.

Funcția începe prin selectarea imaginii de către utilizator. Dacă nu se selectează o imagine, un mesaj notifică acest lucru, iar execuția se oprește. Imaginea selectată este validată, iar dacă formatul sau fișierul nu este valid, se oprește execuția și se afișează un mesaj de eroare.

Imaginea este procesată astfel:

- Este convertită la o reprezentare în niveluri de gri (*grayscale*).
- Se aplică un prag pentru a obține o imagine binară (*alb-negru*).
- Este transformată în spațiul de culoare HSV (*nuanță, saturație, valoare*).

Toate versiunile imaginii sunt afișate în ferestre separate: imaginea originală, *grayscale*, *alb-negru* și HSV.

Funcția creează două histograme pentru a vizualiza distribuția pixelilor:

- O histogramă *grayscale*, care prezintă distribuția intensităților pixelilor în imaginea *grayscale*.
- O histogramă color, care prezintă distribuțiile pentru canalele de culoare *roșu*, *verde* și *albastru*.

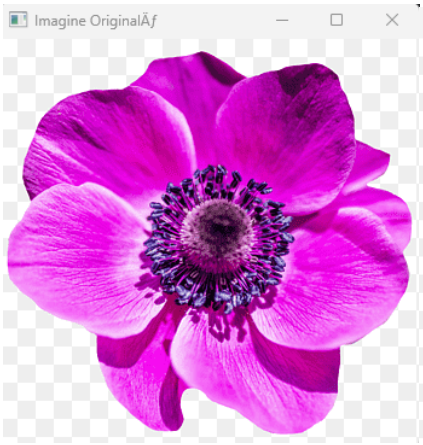
Imaginile rezultate sunt salvate automat într-un director predefinit de pe sistemul utilizatorului. Dacă directorul nu există, funcția îl creează. Numele fișierelor salvate reflectă tipul de procesare aplicată imaginii.

Funcția finalizează execuția așteptând o acțiune din partea utilizatorului. Toate ferestrele sunt închise atunci când utilizatorul apasă tasta `ESC`.

```
def laborator_2():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    directory = r'C:/Users/alex_/Desktop/pi'
    img = cv2.imread(image_path)
    if img is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    (thresh, BlackAndWhiteImage) = cv2.threshold(gray, 127, 255, cv2.
        THRESH_BINARY)
    hsvImage = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    cv2.imshow('Original Image', img)
    cv2.imshow('Gray Image', gray)
    cv2.imshow('Black & White Image', BlackAndWhiteImage)
    cv2.imshow('HSV Image', hsvImage)

    def show_gray_histogram():
        histogram = cv2.calcHist([gray], [0], None, [256], [0, 256])
        hist_img = np.zeros((300, 512, 3), dtype=np.uint8)
        cv2.normalize(histogram, histogram, 0, 300, cv2.NORM_MINMAX)
        for x in range(1, 256):
            cv2.line(hist_img,
                     (int((x-1)*2), 300 - int(histogram[x-1])),
                     (int(x*2), 300 - int(histogram[x])),
                     (255, 255, 255), thickness=1)
```

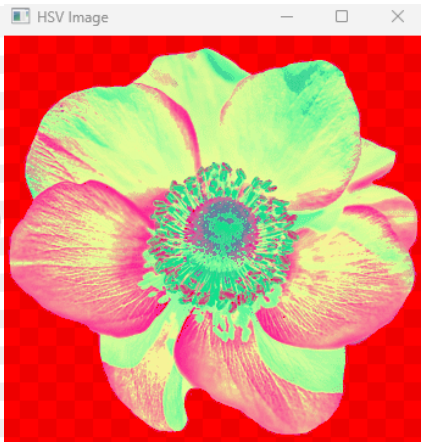
```
    cv2.imshow("Histograma Grayscale", hist_img)

def show_color_histogram():
    hist_img = np.zeros((300, 512, 3), dtype=np.uint8)
    colors = ('b', 'g', 'r')
    for i, col in enumerate(colors):
        histogram = cv2.calcHist([img], [i], None, [256], [0,
            256])
        cv2.normalize(histogram, histogram, 0, 300, cv2.
            NORM_MINMAX)
        for x in range(1, 256):
            cv2.line(hist_img,
                     (int((x-1)*2), 300 - int(histogram[x-1])),
                     (int(x*2), 300 - int(histogram[x])),
                     (255 if col == 'b' else 0, 255 if col == 'g'
                         else 0, 255 if col == 'r' else 0),
                         thickness=1)
    cv2.imshow("Histograma Color", hist_img)

show_gray_histogram()
show_color_histogram()
os.makedirs(directory, exist_ok=True)
os.chdir(directory)
cv2.imwrite('SavedFlower_Gray.jpg', gray)
cv2.imwrite('SavedFlower_BW.jpg', BlackAndWhiteImage)
cv2.imwrite('SavedFlower_HSV.jpg', hsvImage)
print('Imagini salvate cu succes.')
key = cv2.waitKey(0)
if key == 27:
    cv2.destroyAllWindows()
```

LISTING 2. Funcția laborator_2



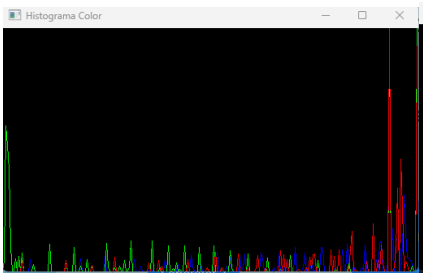
(A) Imaginea originală

(B) Imaginea grayscale

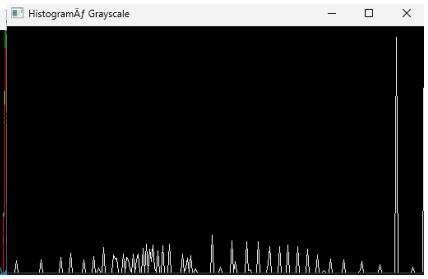


(C) Imaginea HSV

(D) Imaginea alb-negru



(E) Histograma color



(F) Histograma grayscale

FIGURA 3. Imagini generate de funcția `laborator_2`

3. LABORATOR 3

Funcția `laborator_3` aplică mai multe transformări asupra unei imagini selectate de utilizator, afișând rezultatele într-un mod intuitiv și interactiv. Imaginile rezultate evidențiază diferite tehnici de procesare.

La început, utilizatorul selectează imaginea folosind funcția `select_image()`. Dacă nu este selectată o imagine sau dacă aceasta nu este validă, funcția afișează un mesaj de eroare și oprește execuția.

Funcția efectuează următoarele procesări:

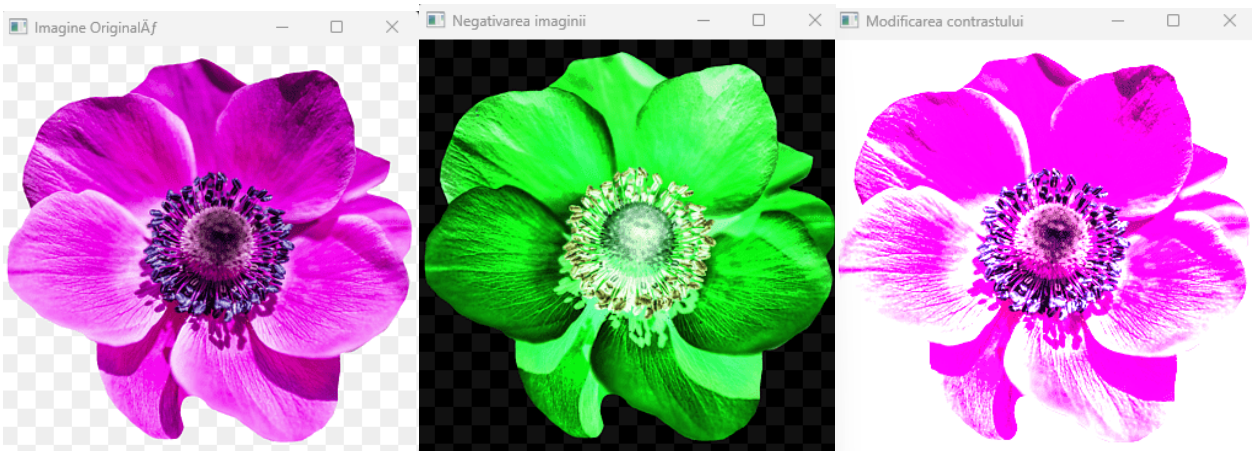
- *Negativarea imaginii*: transformă fiecare pixel astfel încât valorile de intensitate să fie inversate (255 - valoarea pixelului).
- *Modificarea contrastului*: crește contrastul imaginii folosind un factor de scalare (`alpha=2.0`) și un offset nul (`beta=0`).

- *Corecția gamma*: ajustează luminozitatea imaginii prin aplicarea unei transformări neliniare bazate pe o valoare de gamma.
- *Modificarea luminozității*: crește luminozitatea imaginii prin adăugarea unei valori constante (`beta=60`).

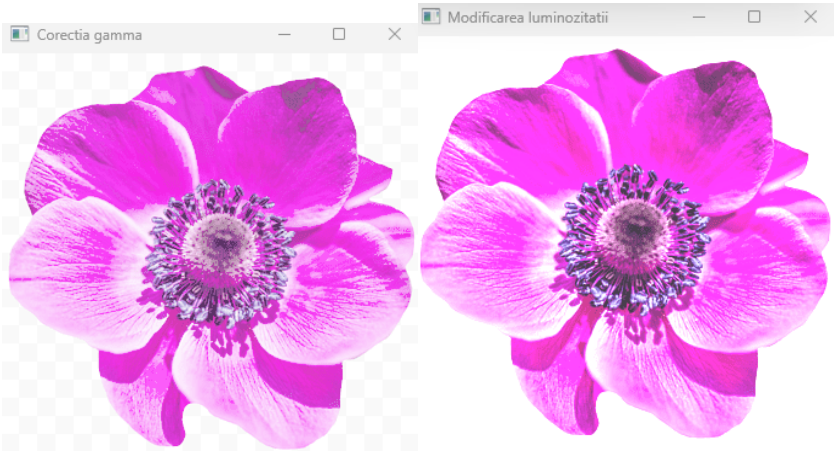
Toate imaginile procesate, inclusiv imaginea originală, sunt afișate în ferestre separate utilizând `cv2.imshow()`. Utilizatorul poate închide toate ferestrele apăsând tasta `ESC`.

```
def laborator_3():
        image_path = select_image()
        if not image_path:
            print("Nicio imagine selectata.")
            return

        img = cv2.imread(image_path)
        if img is None:
            print("Imaginea nu exist  sau formatul este invalid.")
            return

        negative_img = 255 - img
        contrast_img = cv2.convertScaleAbs(img, alpha=2.0, beta=0)
        gamma = 3
        invGamma = 1.0 / gamma
        table = np.array([((i / 255.0) ** invGamma) * 255 for i in np.
            arange(0, 256)]).astype("uint8")
        gamma_img = cv2.LUT(img, table)
        brightness_img = cv2.convertScaleAbs(img, alpha=1, beta=60)

        cv2.imshow("Imaginea originala", img)
        cv2.imshow('Negativarea imaginii', negative_img)
        cv2.imshow('Modificarea contrastului', contrast_img)
        cv2.imshow('Corectia gamma', gamma_img)
        cv2.imshow('Modificarea luminozitatii', brightness_img)

        key = cv2.waitKey(0)
        if key == 27:
            cv2.destroyAllWindows()
```

Listing 3. Funcția laborator_3



(A) Imaginea originală (B) Negativarea imaginii (C) Modificarea contrastului



(D) Corecția gamma (E) Modificarea luminozității

FIGURA 4. Imagini generate de funcția `laborator_3`

4. LABORATOR 4

Descriere pentru funcția `laborator_4`. Funcția `laborator_4` implementează și compară diverse metode de procesare a unei imagini, utilizând filtre bine cunoscute în prelucrarea imaginilor digitale. Scopul principal este de a demonstra cum influențează aceste filtre structura și detaliile imaginii.

La început, utilizatorul este invitat să selecteze o imagine prin funcția `select_image()`. Imaginea este convertită automat în tonuri de gri utilizând funcția `cv2.imread()` cu parametrul `cv2.IMREAD_GRAYSCALE`, astfel încât procesările să fie aplicate pe o singură dimensiune de intensitate.

Funcția aplică următoarele operații:

- *Filtrul mediei aritmetice*:

  - Se utilizează un kernel de dimensiune 3×3, cu toate valorile egale cu $\frac{1}{9}$.
  - Kernelul este aplicat pe întreaga imagine folosind funcția `cv2.filter2D()`.
  - Rezultatul este o imagine netezită, în care detaliile fine sunt atenuate, iar zgomotul este redus.
- *Gaussian Blur*:
  - Se utilizează funcția `cv2.GaussianBlur()` pentru a aplica un filtru gaussian.
  - Kernelul gaussian este generat automat de funcție pe baza dimensiunii date (3×3) și a parametrului `sigmaX`, care controlează cât de mult este estompată imaginea.
  - Gaussian Blur păstrează tranzițiile subtile între intensități și este util pentru reducerea zgomotului fără a sacrifica prea multe detalii.
- *Filtrul Laplace*:
  - Se calculează operatorul Laplace al imaginii utilizând funcția `cv2.Laplacian()`.
  - Operatorul evidențiază marginile prin detectarea gradientelor mari ale intensităților pixelilor.
  - Rezultatul este o imagine în care marginile sunt bine definite, iar detaliile fine devin mai evidente.
- *Kernel personalizat (filtru de trecere sus)*:
  - Se definește un kernel personalizat 3×3 pentru a detecta detaliile fine din imagine.
  - Kernelul este aplicat folosind `cv2.filter2D()`, similar cu filtrul mediei aritmetice.
  - Rezultatul este o imagine în care componentele de frecvență ridicată sunt accentuate, iar fundalul este estompat.

Rezultatele sunt afișate utilizând două metode:

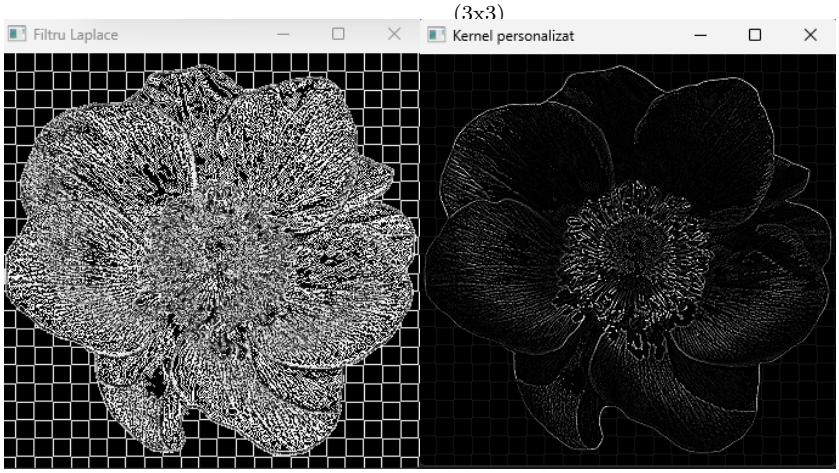
- *Ferestre interactive*: Imaginile procesate sunt afișate cu `cv2.imshow()` în ferestre separate.
- *Diagramă cu mai multe subgraficuri*: Rezultatele sunt plasate într-o diagramă generată cu `matplotlib`, având titluri și dimensiuni uniforme pentru o comparație clară.

```
def contur_extragere(imagine):
    gri = cv2.cvtColor(imagine, cv2.COLOR_BGR2GRAY)
    blur = cv2.blur(gri, (3, 3))
    kernel = np.ones((5, 5), np.uint8)
    gradient = cv2.morphologyEx(blur, cv2.MORPH_GRADIENT, kernel)
    return gradient


def gaussian_blur(image, kernel_size, sigma):
    kernel = cv2.getGaussianKernel(kernel_size, sigma)
    gaussian_kernel = np.outer(kernel, kernel)
    return cv2.filter2D(image, -1, gaussian_kernel)


def umplere_regiuni(imagine):
    gri = cv2.cvtColor(imagine, cv2.COLOR_BGR2GRAY)
    _, binar = cv2.threshold(gri, 127, 255, cv2.THRESH_BINARY_INV)
    kernel = np.ones((9, 9), np.uint8)
    closing = cv2.morphologyEx(binar, cv2.MORPH_CLOSE, kernel)
    return closing


def bidimensional_filter(image, kernel):
    return cv2.filter2D(image, -1, kernel)


def show_images(original, filtered1, filtered2):
    cv2.imshow("Original", original)
    cv2.imshow("Gaussian Blur", filtered1)
    cv2.imshow("Bidimensional Filter", filtered2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


def laborator_4():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return

    mean_kernel = np.ones((3, 3), np.float32) / 9
    img_mean_blur = cv2.filter2D(img, -1, mean_kernel)
    img_gaussian_blur = cv2.GaussianBlur(img, (3, 3), sigmaX=1)
    img_laplacian = cv2.Laplacian(img, cv2.CV_64F)
    custom_kernel = np.array([[0, -1, 0],
                              [-1, 4, -1],
                              [0, -1, 0]])
    img_high_pass = cv2.filter2D(img, -1, custom_kernel)
```

```
fig, axs = plt.subplots(2, 3, figsize=(12, 8))
fig.suptitle("Compararea filtrelor aplicate")

axs[0, 0].imshow(img, cmap='gray')
axs[0, 0].set_title("Originala")
axs[0, 0].axis('off')

axs[0, 1].imshow(img_mean_blur, cmap='gray')
axs[0, 1].set_title("Medie aritmetica (3x3)")
axs[0, 1].axis('off')

axs[0, 2].imshow(img_gaussian_blur, cmap='gray')
axs[0, 2].set_title("Gaussian Blur")
axs[0, 2].axis('off')

axs[1, 0].imshow(img_laplacian, cmap='gray')
axs[1, 0].set_title("Filtru Laplace")
axs[1, 0].axis('off')

axs[1, 1].imshow(img_high_pass, cmap='gray')
axs[1, 1].set_title("Kernel personalizat")
axs[1, 1].axis('off')

axs[1, 2].axis('off')

cv2.imshow("Originala", img)
cv2.imshow("Medie aritmetica (3x3)", img_mean_blur)
cv2.imshow("Gaussian Blur", img_gaussian_blur)
cv2.imshow("Filtru Laplace", np.uint8(img_laplacian))
cv2.imshow("Kernel personalizat", img_high_pass)

key = cv2.waitKey(0)
if key == 27:
    cv2.destroyAllWindows()
```

LISTING 4. Funcția laborator_4

(A) Imaginea originală (B) Medie aritmetică (3x3) (C) Gaussian Blur



(D) Filtru Laplace (E) Kernel personalizat

FIGURA 5. Imagini generate de funcția `laborator_4`, inclusiv analiza suplimentară.

5. LABORATOR 5

Funcția `laborator_5` implementează și compară două metode de filtrare utilizate frecvent în prelucrarea imaginilor digitale: filtrarea Gaussiană și filtrarea bidimensională. Scopul este de a analiza performanța și efectele fiecărei metode asupra unei imagini.

La început, utilizatorul este invitat să selecteze o imagine. Imaginea este validată, iar dacă aceasta nu este validă, funcția afișează un mesaj de eroare și oprește execuția.

Funcția aplică următoarele operații:

- **Gaussian Blur**:

  - Aplică un filtru gaussian pe imagine, utilizând un kernel de dimensiune 5×5 și un factor de estompare `sigma=1.5`.
  - Estomparea gaussiană este calculată utilizând funcția `gaussian_blur()`, iar timpul de execuție este măsurat pentru a analiza eficiența metodei.
- **Filtru bidimensional**:
  - Aplică un kernel bidimensional definit manual, care scoate în evidență marginile și detaliile din imagine.
  - Kernelul utilizat este:

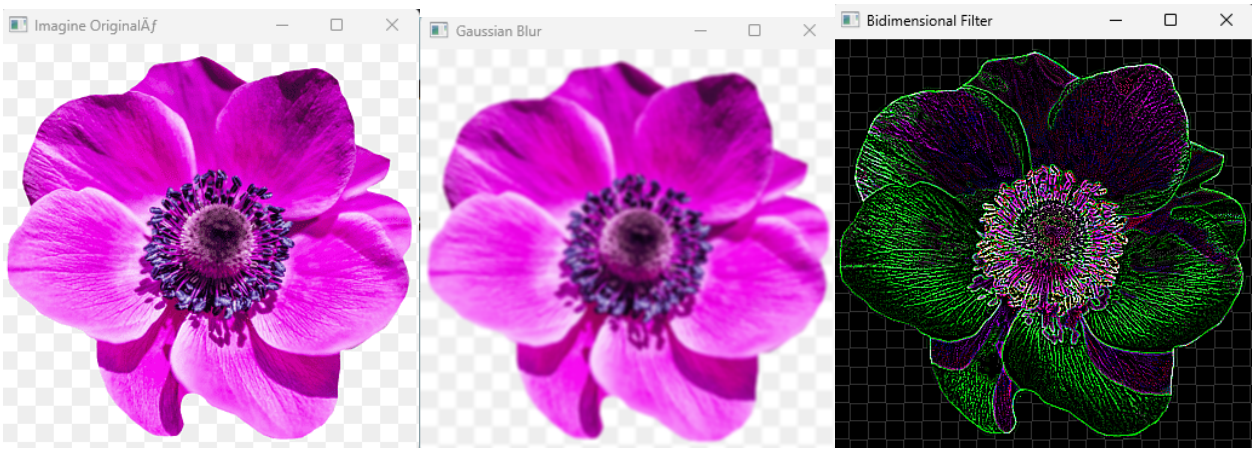
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

  - Timpul de execuție pentru această operație este măsurat folosind funcția `bidimensional_filter()`.

Rezultatele includ imaginea originală, imaginea estompată cu Gaussian Blur și imaginea procesată cu filtrul bidimensional. Acestea sunt afișate utilizând funcția `show_images()`.

```
def laborator_5():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    image = cv2.imread(image_path)
    if image is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return

    kernel_size = 5
    sigma = 1.5
    start_time = time.time()
    gaussian_filtered = gaussian_blur(image, kernel_size, sigma)
    gaussian_time = time.time() - start_time
    print(f"Timp procesare Gaussian Blur: {gaussian_time:.5f} secunde"
        )

    bidimensional_kernel = np.array([[1, 1, 1],
                                     [1, -8, 1],
                                     [1, 1, 1]])

    start_time = time.time()
    bidimensional_filtered = bidimensional_filter(image,
        bidimensional_kernel)
    bidimensional_time = time.time() - start_time
    print(f"Timp procesare Bidimensional Filter: {bidimensional_time
        :.5f} secunde")

    show_images(image, gaussian_filtered, bidimensional_filtered)
```

Listing 5. Funcția laborator_5



(A) Imaginea originală (B) Gaussian Blur (C) Filtru bidimensional

FIGURA 6. Imagini generate de funcția `laborator_5`.

6. LABORATOR 6

Funcția `laborator_6` implementează algoritmi pentru identificarea componentelor conexe într-o imagine binară, utilizând două metode: *BFS* (parcurgere în lățime) și *Two-Pass* (în două treceri). Scopul este de a analiza modul în care aceste metode segmentează regiuni distincte din imagine.

Algoritmi utilizați:

- **BFS (Breadth-First Search)**:
  - Parcurge imaginea folosind o coadă (`deque`), marcând fiecare pixel conectat în cadrul aceleiași componente conexe.
  - Fiecare componentă conexă primește o etichetă unică.
- **Two-Pass**:
  - În prima trecere, fiecare pixel este etichetat pe baza vecinilor săi.
  - Se construiește un grafic al relațiilor dintre etichete.
  - În a doua trecere, etichetele sunt reduse la reprezentantul lor unic utilizând parcurgeri suplimentare.

Funcția începe prin convertirea imaginii selectate într-o imagine binară utilizând o funcție de prag (`threshold`). Aceasta transformă pixelii cu intensități mari ($\geq$ 127) în alb (255) și restul în negru (0).

Rezultatele includ:

- Imaginea originală binară.
- Imaginea segmentată utilizând algoritmul BFS.
- Imaginea segmentată utilizând algoritmul Two-Pass.

Fiecare componentă conexă este vizualizată folosind culori diferite pentru o interpretare ușoară.

```
from collections import deque

def connected_components_bfs(img):
```

```
    height, width = img.shape
    labels = np.zeros((height, width), dtype=np.int32)
    label = 0

    for i in range(height):
        for j in range(width):
            if img[i, j] == 0 and labels[i, j] == 0:
                label += 1
                q = deque()
                q.append((i, j))
                labels[i, j] = label

                while q:
                    x, y = q.popleft()
                    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
                        nx, ny = x + dx, y + dy
                        if 0 <= nx < height and 0 <= ny < width and
                            img[nx, ny] == 0 and labels[nx, ny] == 0:
                            labels[nx, ny] = label
                            q.append((nx, ny))
    return labels

def connected_components_two_pass(img):
    height, width = img.shape
    labels = np.zeros((height, width), dtype=np.int32)
    label = 0
    edges = {}

    for i in range(height):
        for j in range(width):
            if img[i, j] == 0:
                neighbors = []
                if i > 0 and labels[i - 1, j] > 0:
                    neighbors.append(labels[i - 1, j])
                if j > 0 and labels[i, j - 1] > 0:
                    neighbors.append(labels[i, j - 1])

                if not neighbors:
                    label += 1
                    labels[i, j] = label
                    edges[label] = []
                else:
                    min_label = min(neighbors)
                    labels[i, j] = min_label
                    for neighbor in neighbors:
                        if neighbor != min_label:
                            edges[min_label].append(neighbor)
                            edges[neighbor].append(min_label)

    new_labels = np.zeros(label + 1, dtype=np.int32)
    new_label = 0

    for i in range(1, label + 1):
```

```
            if new_labels[i] == 0:
                new_label += 1
                q = deque([i])
                new_labels[i] = new_label

                while q:
                    x = q.popleft()
                    for y in edges.get(x, []):
                        if new_labels[y] == 0:
                            new_labels[y] = new_label
                            q.append(y)

        for i in range(height):
            for j in range(width):
                if labels[i, j] > 0:
                    labels[i, j] = new_labels[labels[i, j]]

        return labels

    def laborator_6():
        image_path = select_image()
        if not image_path:
            print("Nicio imagine selectata.")
            return

        binary_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if binary_image is None:
            print("Imaginea nu exista sau formatul este invalid.")
            return

        _, binary_image = cv2.threshold(binary_image, 127, 255, cv2.
            THRESH_BINARY)

        labels_bfs = connected_components_bfs(binary_image)

        labels_two_pass = connected_components_two_pass(binary_image)

        def visualize_labels(labels):
            unique_labels = np.unique(labels)
            colored_labels = np.zeros((*labels.shape, 3), dtype=np.uint8)
            for label in unique_labels:
                if label == 0:
                    continue
                mask = (labels == label)
                color = np.random.randint(0, 255, size=3)
                colored_labels[mask] = color
            return colored_labels

        colored_bfs = visualize_labels(labels_bfs)
        colored_two_pass = visualize_labels(labels_two_pass)

        cv2.imshow("Imagine Originala", binary_image)
        cv2.imshow("BFS Algorithm", colored_bfs)
```

```
    cv2.imshow("Two-Pass Algorithm", colored_two_pass)

    key = cv2.waitKey(0)
    if key == 27:
        cv2.destroyAllWindows()
```

LISTING 6. Funcția laborator_6

(A) Imaginea Originală (B) BFS Algorithm (C) Two-Pass Algorithm

FIGURA 7. Imagini generate de funcția `laborator_6`.

7. LABORATOR 7

Acest laborator explorează procesul de detectare a marginilor din imagini, utilizând o combinație de metode avansate de procesare a imaginilor. Obiectivul principal este să obținem o reprezentare binară clară a marginilor unei imagini, aplicând două tehnici esențiale: thresholding adaptiv, care segmentează imaginea pe baza unor praguri locale, și hysteresis edge tracking, care consolidează marginile slabe prin analiza conectivității lor cu marginile puternice. La final, algoritmul produce două imagini distincte:

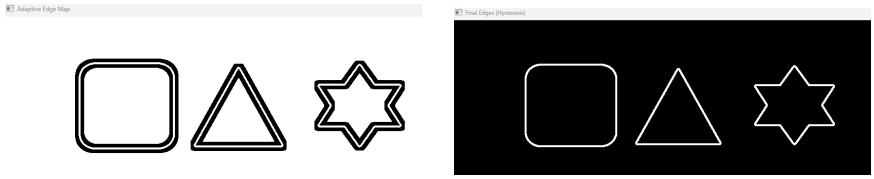
Harta marginilor adaptive, generată prin thresholding adaptiv, care oferă o perspectivă inițială asupra contururilor. Harta marginilor consolidate, obținută prin procesul de histerezis, care reprezintă marginile finale, mai robuste și mai bine definite.

```
def laborator_7(image_path, window_size=15, C=2, low_ratio=0.5,
    high_ratio=0.2, blur_kernel=(3, 3)):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise FileNotFoundError(f"Image at path '{image_path}' could
            not be loaded.")

    img_blur = cv2.GaussianBlur(img, blur_kernel, 0)

    grad_x = cv2.Sobel(img_blur, cv2.CV_32F, 1, 0, ksize=3)
    grad_y = cv2.Sobel(img_blur, cv2.CV_32F, 0, 1, ksize=3)
    magnitude = cv2.magnitude(grad_x, grad_y)

    mag_8u = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX).
        astype(np.uint8)
    edge_map_adaptive = cv2.adaptiveThreshold(
        mag_8u, 255,
```

```
            cv2.ADAPTIVE_THRESH_MEAN_C,
            cv2.THRESH_BINARY,
            window_size,
            C
        )

        mag_8u_float = mag_8u.astype(np.float32)
        max_val = mag_8u_float.max()
        high_thresh = max_val * high_ratio
        low_thresh = high_thresh * low_ratio

        result = np.zeros_like(mag_8u, dtype=np.uint8)
        strong_i, strong_j = np.where(mag_8u_float > high_thresh)
        result[strong_i, strong_j] = 255

        weak_i, weak_j = np.where((mag_8u_float <= high_thresh) & (
            mag_8u_float >= low_thresh))
        weak_set = set(zip(weak_i, weak_j))

        neighbors = [(-1, -1), (-1, 0), (-1, 1),
                     (0, -1),           (0, 1),
                     (1, -1), (1, 0), (1, 1)]

        to_visit = list(zip(strong_i, strong_j))
        while to_visit:
            x, y = to_visit.pop()
            for dx, dy in neighbors:
                nx, ny = x + dx, y + dy
                if 0 <= nx < mag_8u.shape[0] and 0 <= ny < mag_8u.shape
                    [1]:
                    if (nx, ny) in weak_set:
                        result[nx, ny] = 255
                        weak_set.remove((nx, ny))
                        to_visit.append((nx, ny))

        return edge_map_adaptive, result

    def laborator_7_button():
        image_path = select_image()
        if not image_path:
            print("Nicio imagine selectata.")
            return

        try:
            adaptive_edges, final_edges = laborator_7(
                image_path, window_size=15, C=2, low_ratio=0.5, high_ratio
                    =0.2
            )

            cv2.imshow("Adaptive Edge Map", adaptive_edges)
            cv2.imshow("Final Edges (Hysteresis)", final_edges)
            cv2.waitKey(0)
```

```
        cv2.imwrite("edges_adaptive.jpg", adaptive_edges)
        cv2.imwrite("edges_final_hysteresis.jpg", final_edges)
    except Exception as e:
        print(f"Error: {e}")
```

Listing 7. Funcția laborator_7



TEME

8. Tema 1

Funcția `tema_1` implementează un algoritm de cuantizare și dithering aplicat asupra unei imagini în tonuri de gri. Scopul este de a reduce numărul de niveluri de gri utilizând praguri detectate automat și de a îmbunătăți vizualizarea prin metoda Floyd-Steinberg.

- **Detectarea vârfurilor din histogramă**:
  - Histogramă normalizată este calculată din imagine.
  - Vârfurile histogramelor (praguri de cuantizare) sunt identificate utilizând o metodă bazată pe ferestre locale.
  - Aceste praguri servesc drept referințe pentru cuantizarea imaginii.
- **Cuantizarea imaginii**:
  - Fiecare pixel este asociat celui mai apropiat prag (vârf din histogramă).
  - Rezultatul este o imagine cu niveluri de gri limitate la valorile pragurilor.
- **Dithering Floyd-Steinberg**:
  - Se aplică metoda Floyd-Steinberg pentru distribuirea erorii de cuantizare către pixelii vecini.
  - Această metodă îmbunătățește aspectul vizual al imaginii prin reducerea efectelor de banding.
- **Generarea histogramei normalizate**:
  - Este afișată histograma normalizată pentru a vizualiza distribuția nivelurilor de gri.

Rezultatele includ:

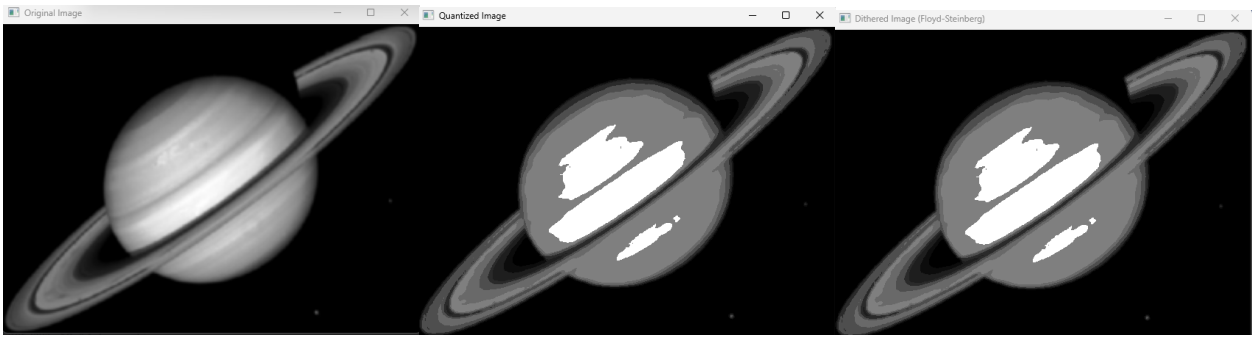
- Imaginea originală.
- Imaginea cuantizată.
- Imaginea ditherată utilizând metoda Floyd-Steinberg.
- Histogramă normalizată a imaginii originale.

```
def tema_1():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return
```

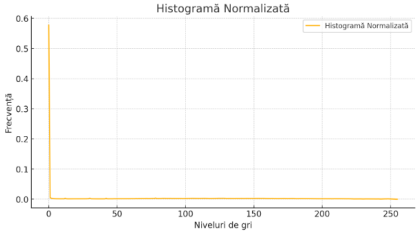
```
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
if img is None:
    print("Imaginea nu exista sau formatul este invalid.")
    return

def detect_histogram_peaks(img, window_size=5, threshold=0.0003):
    histogram, bin_edges = np.histogram(img.flatten(), bins=256,
        range=[0, 256], density=True)
    normalized_hist = histogram / histogram.sum()
    peaks = []
    for idx in range(window_size, 256 - window_size):
        local_mean = np.mean(normalized_hist[idx - window_size:
            idx + window_size + 1])
        if (normalized_hist[idx] > local_mean + threshold and
            normalized_hist[idx] >= np.max(normalized_hist[idx -
                window_size: idx + window_size + 1])):
            peaks.append(idx)
    peaks = [0] + peaks + [255]
    return peaks, normalized_hist

def apply_quantization(img, peaks):
    quantized_img = img.copy()
    rows, cols = img.shape
    for row in range(rows):
        for col in range(cols):
            pixel_val = img[row, col]
            nearest_peak = min(peaks, key=lambda p: abs(int(p) -
                int(pixel_val)))
            quantized_img[row, col] = nearest_peak
    return quantized_img

def apply_floyd_steinberg_dithering(img, peaks):
    img = img.astype(float)
    rows, cols = img.shape
    for row in range(rows):
        for col in range(cols):
            current_pixel = img[row, col]
            quantized_pixel = min(peaks, key=lambda p: abs(p -
                current_pixel))
            img[row, col] = quantized_pixel
            error = current_pixel - quantized_pixel
            if col + 1 < cols:
                img[row, col + 1] += error * 7 / 16
            if row + 1 < rows:
                if col > 0:
                    img[row + 1, col - 1] += error * 3 / 16
                img[row + 1, col] += error * 5 / 16
                if col + 1 < cols:
                    img[row + 1, col + 1] += error * 1 / 16
    return np.clip(img, 0, 255).astype(np.uint8)

peaks, normalized_hist = detect_histogram_peaks(img)
```

```
print("Histogram peaks (threshold values):", peaks)

quantized_img = apply_quantization(img, peaks)
dithered_img = apply_floyd_steinberg_dithering(quantized_img,
    peaks)
cv2.imshow("Original Image", img)
cv2.imshow("Quantized Image", quantized_img)
cv2.imshow("Dithered Image (Floyd-Steinberg)", dithered_img)

def plot_histogram():
    plt.plot(normalized_hist, label="Histograma Normalizata")
    plt.title("Histograma Normalizata")
    plt.xlabel("Niveluri de gri")
    plt.ylabel("Frecventa")
    plt.legend()

show_plot_in_cv2(plot_histogram, title="Histograma")

key = cv2.waitKey(0)
if key == 27:
    cv2.destroyAllWindows()
```

LISTING 8. Funcția tema_1



(A) Imaginea Originală (B) Imagine Cuantizată (C) Imagine(Floyd-Steinberg)



(D) Histograma

FIGURA 8. Imagini generate de funcția `tema_1`

9. Tema 2

Funcția `tema_2` implementează două operații esențiale pentru prelucrarea imaginilor în tonuri de gri: binarizarea globală și egalizarea histogramei. Scopul este de a transforma și îmbunătăți vizibilitatea detaliilor în imagine.

- **Binarizarea globală**:
  - Este determinată o valoare prag calculată ca media nivelurilor de gri din imagine.
  - Fiecare pixel este transformat în alb sau negru în funcție de această valoare prag.
- **Egalizarea histogramei**:
  - Se aplică un algoritm care redistribuie valorile pixelilor astfel încât să crească contrastul imaginii.
  - Această metodă este utilă pentru a evidenția detaliile în imagini întunecate sau slab iluminate.
- **Generarea histogramelor**:
  - Se afișează histograma imaginii originale pentru a vizualiza distribuția inițială a nivelurilor de gri.
  - Se generează histograma imaginii egalizate pentru a observa impactul procesului de egalizare.

Rezultatele includ:

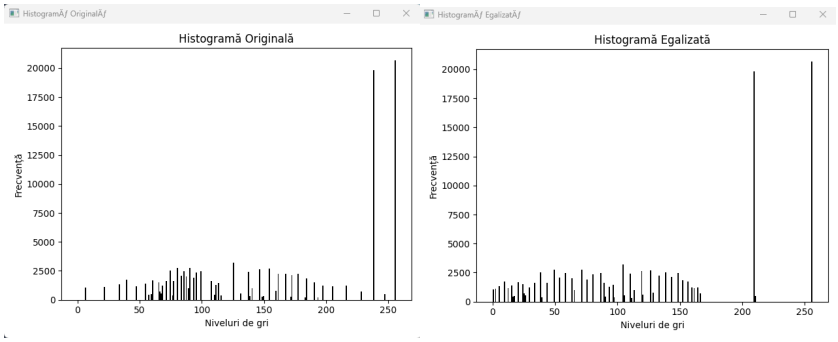
- Imaginea originală.
- Imaginea binarizată utilizând pragul calculat automat.
- Imaginea cu histograma egalizată.
- Histogramă a imaginii originale și egalizate.

```
def tema_2():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return

    def prag_binarizare_globala(image):
        valoare_prag = np.mean(image)
        _, imagine_binara = cv2.threshold(image, valoare_prag, 255,
            cv2.THRESH_BINARY)
        return imagine_binara, valoare_prag

    def egalizare_histograma(image):
        imagine_egalizata = cv2.equalizeHist(image)
        return imagine_egalizata

    imagine_binara, valoare_prag = prag_binarizare_globala(image)
    imagine_egalizata = egalizare_histograma(image)
```

```
cv2.imshow("Imagine␣originala", image)
cv2.imshow(f"Imagine␣binarizata␣(Prag:␣{valoare_prag:.2f})",
    imagine_binara)
cv2.imshow("Imagine␣egalizata", imagine_egalizata)

def plot_original_histogram():
    plt.title("Histograma␣Originala")
    plt.hist(image.ravel(), bins=256, range=[0, 256], color='black
        ')
    plt.xlabel("Niveluri␣de␣gri")
    plt.ylabel("Frecventa")

def plot_equalized_histogram():
    plt.title("Histograma␣Egalizata")
    plt.hist(imagine_egalizata.ravel(), bins=256, range=[0, 256],
        color='black')
    plt.xlabel("Niveluri␣de␣gri")
    plt.ylabel("Frecventa")

show_plot_in_cv2(plot_original_histogram, title="Histograma␣
    Originala")
show_plot_in_cv2(plot_equalized_histogram, title="Histograma␣
    Egalizata")

key = cv2.waitKey(0)
if key == 27:
    cv2.destroyAllWindows()
```

LISTING 9. Funcția tema_2

(A) Imaginea Originală (B) Imagine Binarizată (C) Imagine Egalizată



(D) Histograma Originală (E) Histograma Egalizată

FIGURA 9. Imagini generate de funcția `tema_2`

10. TEMA 3

Funcția `tema_3` implementează analiza și filtrarea unei imagini în domeniul frecvenței utilizând transformata Fourier. Se aplică diferite tipuri de filtre (trece-jos și trece-sus, atât circulare cât și gaussiene) pentru a evidenția sau elimina anumite frecvențe din imagine.

Operații realizate:

- **Calculul spectrului Fourier al imaginii**:
  - Imaginea este transformată din domeniul spațial în domeniul frecvenței utilizând transformata Fourier 2D.
  - Spectrul de frecvență este centrat pentru o mai bună vizualizare.
- **Crearea și aplicarea de filtre**:
  - **Filtre circulare**:
    - Filtru trece-jos: permite doar frecvențele joase și elimină frecvențele înalte.

    - Filtru trece-sus: permite doar frecvențele înalte și elimină frecvențele joase.
  - **Filtre gaussiene**:
    - Filtru trece-jos gaussian: aplică o funcție gaussiană pentru a reține frecvențele joase.
    - Filtru trece-sus gaussian: inversează filtrul gaussian pentru a păstra frecvențele înalte.
- **Reconstrucția imaginii filtrate**:
  - Frecvențele filtrate sunt transformate înapoi în domeniul spațial utilizând transformata Fourier inversă.

Rezultate generate:

- Imaginea originală.
- Spectrul Fourier al imaginii originale.
- Imaginile și spectrele rezultate după aplicarea fiecărui tip de filtru.

```
def tema_3():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return

    def compute_fourier_spectrum(img):
        dft = np.fft.fft2(img)
        dft_shift = np.fft.fftshift(dft)
        spectrum = 20 * np.log(np.abs(dft_shift) + 1)
        return dft_shift, spectrum

    def apply_frequency_filter(img, filter_mask):
        dft_shift, _ = compute_fourier_spectrum(img)
        filtered_dft = dft_shift * filter_mask
        spectrum = 20 * np.log(np.abs(filtered_dft) + 1)
        inverse_dft = np.fft.ifft2(np.fft.ifftshift(filtered_dft))
        return np.abs(inverse_dft), spectrum

    def create_filter_mask(shape, filter_type, radius):
        rows, cols = shape
        crow, ccol = rows // 2, cols // 2
        mask = np.zeros((rows, cols), np.float32)

        if filter_type == "low_pass":
            cv2.circle(mask, (ccol, crow), radius, 1, thickness=-1)
        elif filter_type == "high_pass":
            mask[:] = 1
            cv2.circle(mask, (ccol, crow), radius, 0, thickness=-1)

        return mask
```

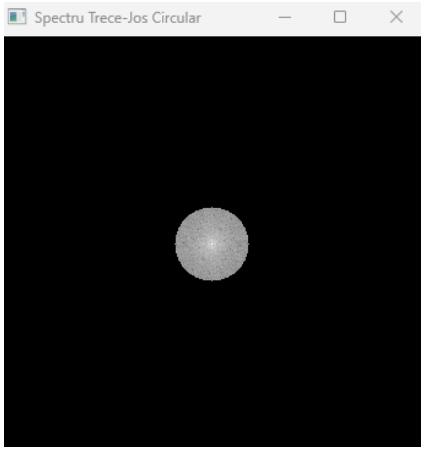
```
def create_gaussian_filter(shape, filter_type, sigma):
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    x = np.arange(cols) - ccol
    y = np.arange(rows) - crow
    x, y = np.meshgrid(x, y)

    if filter_type == "low_pass":
        mask = np.exp(-(x**2 + y**2) / (2 * sigma**2))
    elif filter_type == "high_pass":
        mask = 1 - np.exp(-(x**2 + y**2) / (2 * sigma**2))
    return mask

dft_shift, original_spectrum = compute_fourier_spectrum(img)

radius = 30
low_pass_mask = create_filter_mask(img.shape, "low_pass", radius)
high_pass_mask = create_filter_mask(img.shape, "high_pass", radius
    )

img_low_pass, spectrum_low_pass = apply_frequency_filter(img,
    low_pass_mask)
img_high_pass, spectrum_high_pass = apply_frequency_filter(img,
    high_pass_mask)

sigma = 30
gaussian_low_pass = create_gaussian_filter(img.shape, "low_pass",
    sigma)
gaussian_high_pass = create_gaussian_filter(img.shape, "high_pass"
    , sigma)

img_gaussian_low_pass, spectrum_gaussian_low_pass =
    apply_frequency_filter(img, gaussian_low_pass)
img_gaussian_high_pass, spectrum_gaussian_high_pass =
    apply_frequency_filter(img, gaussian_high_pass)

original_spectrum = cv2.normalize(original_spectrum, None, 0, 255,
     cv2.NORM_MINMAX).astype(np.uint8)
spectrum_low_pass = cv2.normalize(spectrum_low_pass, None, 0, 255,
     cv2.NORM_MINMAX).astype(np.uint8)
spectrum_high_pass = cv2.normalize(spectrum_high_pass, None, 0,
    255, cv2.NORM_MINMAX).astype(np.uint8)
spectrum_gaussian_low_pass = cv2.normalize(
    spectrum_gaussian_low_pass, None, 0, 255, cv2.NORM_MINMAX).
    astype(np.uint8)
spectrum_gaussian_high_pass = cv2.normalize(
    spectrum_gaussian_high_pass, None, 0, 255, cv2.NORM_MINMAX).
    astype(np.uint8)

cv2.imshow("Imagine Originala", img)
cv2.imshow("Spectru Fourier Original", original_spectrum)
cv2.imshow("Imagine Trece-Jos Circular", img_low_pass.astype(np.
    uint8))
```

```
cv2.imshow("Spectru Trece-Jos Circular", spectrum_low_pass)
cv2.imshow("Imagine Trece-Sus Circular", img_high_pass.astype(np.
    uint8))
cv2.imshow("Spectru Trece-Sus Circular", spectrum_high_pass)
cv2.imshow("Imagine Trece-Jos Gaussian", img_gaussian_low_pass.
    astype(np.uint8))
cv2.imshow("Spectru Trece-Jos Gaussian",
    spectrum_gaussian_low_pass)
cv2.imshow("Imagine Trece-Sus Gaussian", img_gaussian_high_pass.
    astype(np.uint8))
cv2.imshow("Spectru Trece-Sus Gaussian",
    spectrum_gaussian_high_pass)

key = cv2.waitKey(0)
if key == 27:
    cv2.destroyAllWindows()
```

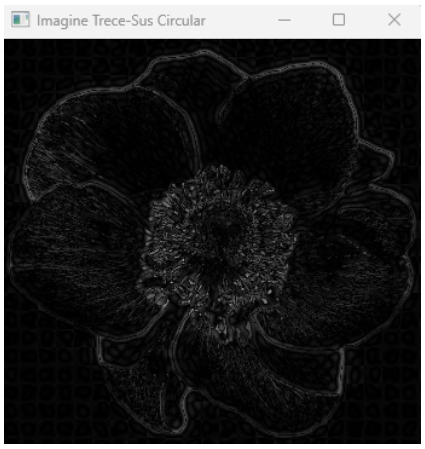
LISTING 10. Funcția tema_3


Imagine OriginalÃƒ

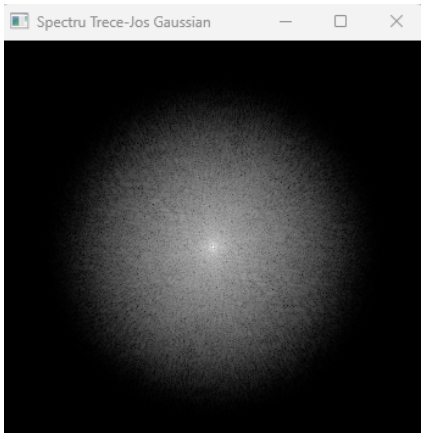

Spectru Fourier Original

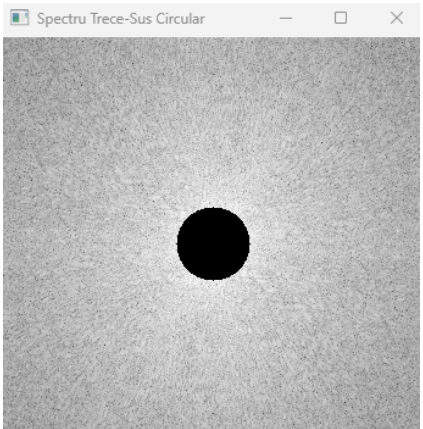

Spectru Trece-Jos Circular


Imagine Trece-Jos Circular


Imagine Trece-Sus Circular


Imagine Trece-Jos Gaussian


Spectru Trece-Jos Gaussian


Spectru Trece-Sus Circular

11. Tema 4

Funcția `tema_4` implementează procesarea imaginii prin două tehnici de morfologie: extragerea contururilor și umplerea regiunilor. Aceasta este însoțită de o interfață grafică simplă creată cu biblioteca `tkinter`.

- **Extragerea contururilor utilizând morfologie**:
  - Se identifică contururile obiectelor din imagine folosind operațiuni morfologice.
- **Umplerea regiunilor utilizând morfologie**:
  - Se umplu regiunile obiectelor utilizând operații morfologice.
- **Interfața grafică cu Tkinter**:
  - Aplicația afișează titlul "Prelucrarea imaginilor".
  - Interfața permite utilizatorului să selecteze și să proceseze imaginea.
- Imaginea originală.
- Imaginea procesată pentru extragerea contururilor.
- Imaginea procesată pentru umplerea regiunilor.

```
def tema_4():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    imagine = cv2.imread(image_path)
    if imagine is None:
        print("Imaginea nu exista sau formatul este invalid.")
        return

    contur = contur_extragere(imagine)
    umplere = umplere_regiuni(imagine)

    cv2.imshow("Imaginea Originala", imagine)
    cv2.imshow("Extragere Contur cu Morfologie", contur)
    cv2.imshow("Umplere Regiuni cu Morfologie", umplere)

    key = cv2.waitKey(0)
    if key == 27:
        cv2.destroyAllWindows()

root = tk.Tk()
root.title("PI")
root.geometry("800x600")
root.minsize(600, 400)

frame_content = tk.Frame(root, bg="lightblue", padx=20, pady=20)
frame_content.pack(fill="both", expand=True)

label_title = tk.Label(
    frame_content,
    text="Prelucrarea imaginilor",
    font=("Arial", 24, "bold"),
    bg="lightblue",
```

```
    fg="black"
)
label_title.pack(pady=20)

def contur_extragere(imagine):
    imagine_gray = cv2.cvtColor(imagine, cv2.COLOR_BGR2GRAY)
    kernel = np.ones((3, 3), np.uint8)
    eroziune = cv2.erode(imagine_gray, kernel, iterations=1)
    dilatare = cv2.dilate(imagine_gray, kernel, iterations=1)
    contur = cv2.absdiff(dilatare, eroziune)
    return contur

def umplere_regiuni(imagine):
    imagine_gray = cv2.cvtColor(imagine, cv2.COLOR_BGR2GRAY)
    _, imagine_binara = cv2.threshold(imagine_gray, 127, 255, cv2.
        THRESH_BINARY)
    kernel = np.ones((3, 3), np.uint8)
    umplere = cv2.morphologyEx(imagine_binara, cv2.MORPH_CLOSE, kernel
        )
    return umplere
```

Listing 11. Funcția tema_4

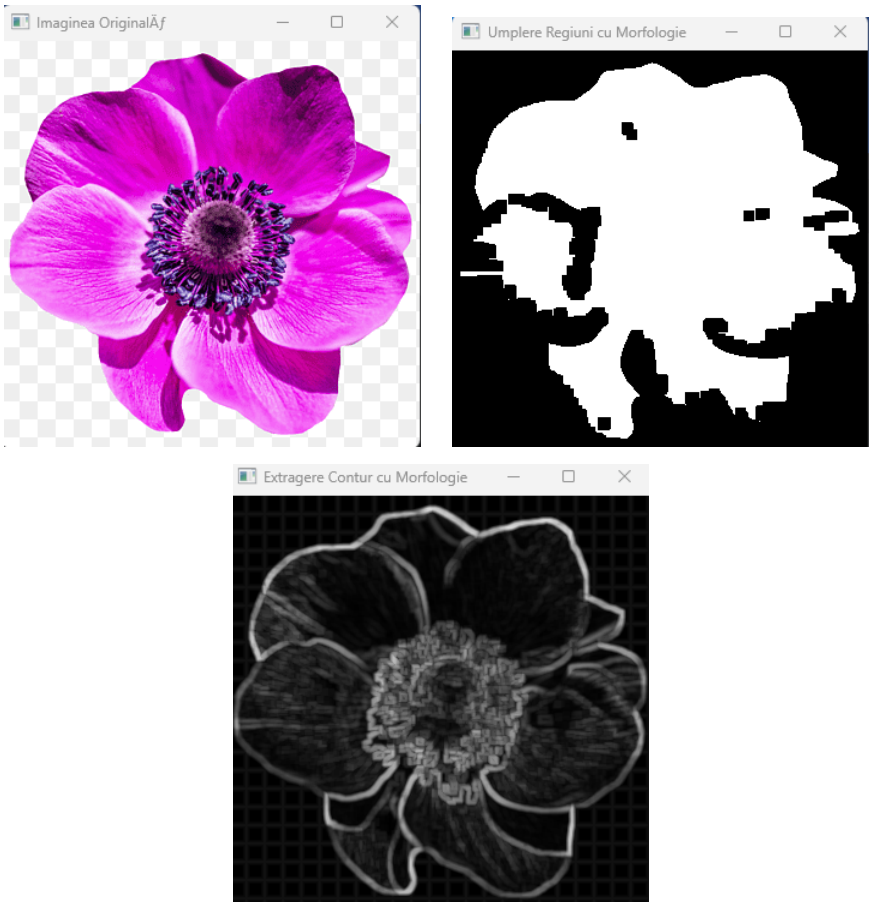


FIGURA 11. Imaginea generate de tema 4

12. TEMA 5

Funcția `tema_5` implementează algoritmi pentru binarizarea unei imagini, detectarea contururilor folosind metoda urmărirea contururilor și codificarea acestora prin codul în lanț (chain code). Rezultatele includ afișarea contururilor pe imagine și codurile în lanț pentru fiecare contur.

- **Crearea imaginii binare**:
  - Imaginea originală este convertită într-o imagine binară, utilizând un prag fix (127).
- **Extragerea contururilor prin urmărire**:
  - Se detectează contururile în imaginea binară folosind un algoritm de urmărire, ce marchează contururile pixel cu pixel.
  - Contururile detectate sunt salvate într-o listă.
- **Codificarea contururilor cu cod în lanț**:
  - Pentru fiecare contur, se generează un cod în lanț care reprezintă direcțiile relative între punctele succesive ale conturului.
- **Suprapunerea contururilor pe imagine**:

  - Contururile detectate sunt desenate pe imaginea binară, utilizând o culoare distinctă (verde).
- Imaginea binară.
- Imaginea cu contururile suprapuse.
- Codurile în lanț afișate în consolă pentru fiecare contur detectat.

```
def tema_5():
    image_path = select_image()
    if not image_path:
        print("Nicio imagine selectata.")
        return

    binary_image = create_binary_image(image_path)
    contours = contour_following(binary_image)
    chain_codes = [chain_code(binary_image, contour) for contour in
        contours]

    overlay_image = overlay_contours(binary_image, contours)
    cv2.imshow("Contours Overlay", overlay_image)


    for i, chain in enumerate(chain_codes):
        print(f"Chain Code for Contour {i + 1}: {chain}")

    key = cv2.waitKey(0)
    if key == 27:
        cv2.destroyAllWindows()
```

LISTING 12. Funcția tema_5

```
def create_binary_image(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    _, binary = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
    return binary

def contour_following(binary_image):
    contours = []
    visited = np.zeros_like(binary_image, dtype=bool)
    rows, cols = binary_image.shape

    for r in range(rows):
        for c in range(cols):
            if binary_image[r, c] == 0 and not visited[r, c]:
                contour = []
                start = (r, c)
                current = start
                prev_dir = 0

                while True:
                    contour.append(current)
                    visited[current] = True

                    found_next = False
```

```
                    for i in range(8):
                        next_dir = (prev_dir + i) % 8
                        dr, dc = direction_offset(next_dir)
                        nr, nc = current[0] + dr, current[1] + dc

                        if 0 <= nr < rows and 0 <= nc < cols and
                            binary_image[nr, nc] == 0 and not visited[
                            nr, nc]:
                            current = (nr, nc)
                            prev_dir = (next_dir + 4) % 8
                            found_next = True
                            break

                    if not found_next or current == start:
                        break

                contours.append(contour)

    return contours

def direction_offset(direction):
    offsets = [
        (-1, 0), (-1, 1), (0, 1), (1, 1),
        (1, 0), (1, -1), (0, -1), (-1, -1)
    ]
    return offsets[direction]

def chain_code(binary_image, contour):
    chain = []
    start = contour[0]
    current = start
    prev_dir = 0

    for _ in range(len(contour) - 1):
        for i in range(8):
            next_dir = (prev_dir + i) % 8
            dr, dc = direction_offset(next_dir)
            next_pixel = (current[0] + dr, current[1] + dc)

            if next_pixel in contour:
                chain.append(next_dir)
                current = next_pixel
                prev_dir = (next_dir + 4) % 8
                break

    return chain

def overlay_contours(binary_image, contours):
    overlay = cv2.cvtColor(binary_image, cv2.COLOR_GRAY2BGR)
    for contour in contours:
        for point in contour:
            overlay[point[0], point[1]] = (0, 255, 0)
```

```
    return overlay
```

LISTING 13. Funcții auxiliare pentru procesarea imaginilor

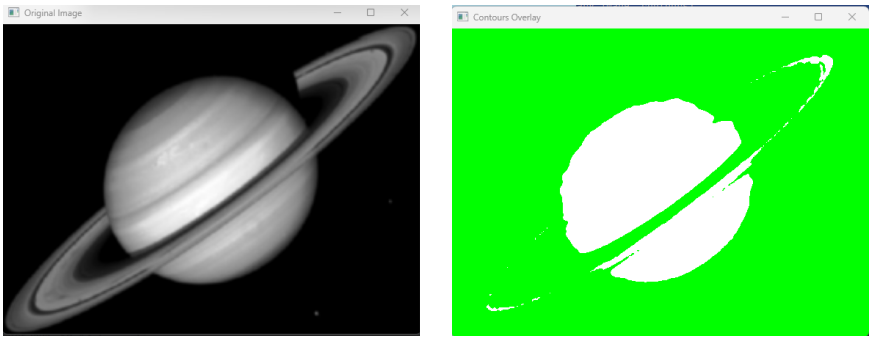


FIGURA 12. Imagini generate de funcția `tema_5`

BIBLIOGRAFIE

[1] G. BRADSKI şi A. KAEHLER, *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Sebastopol, 2008.
[2] https://github.com/Gratiela2963/PI_PROIECT.git
[3] https://classroom.google.com/c/NzE5MzExMTI0MjI1