Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

Ghid de utilizare

Image Processing Platform

Autor: **Cosmin Polifronie**
<cosmin.polifronie@gmail.com>

Brașov, 2020

Cuprins

1 Descrierea aplicației

Aplicația este distribuită prin intermediul platformei Github: https://github.com/unitbvcv/image_processing_platform

Pentru a clona aplicația este necesar utilitarul Git. Clonarea se face folosind comanda de mai jos:

```
git clone https://github.com/unitbvcv/image_processing_platform.git
```

Pentru descărcarea unor modificări efectuate în repository-ul Github se va folosi următoarea comandă in directorul creat automat mai sus:

```
git pull
```

1.1 Ghid de instalare

1.1.1 Platforme suportate

Aplicația are suport multiplatformă, acesta fiind oferit de limbajul Python și framework-ul Qt. Platformele suportate sunt: [7]

- Microsoft Windows 7 sau mai nou
- Apple Mac OS X 10.13 High Sierra sau mai nou
- Linux: diverse distribuții recente care folosesc protocolul de afișare X11 (testat pe Ubuntu 18.04 LTS, Arch Linux 2019.03)

1.1.2 Cerințe minime de sistem

Aplicația nu are cerințe suplimentare celor cerute de platformele suportate enumerate mai sus.

Singura excepție este cantitatea de memorie prezentă: aplicația va reuși să pornească în cazul în care sunt disponibili cel putin ~100 MB RAM. Cantitatea de RAM necesară pentru rularea algoritmilor este dependentă de algoritmul implementat și nu poate fi controlată.

În funcție de cantitatea de SWAP disponibilă pe sistemul dumneavoastră, șansele ca aplicația să se închidă din cauza lipsei memoriei sunt foarte mici. În cazul în care vă aflați în această situație, veți fi anuntat printr-un mesaj pe ecran.

1.1.3 Dependințe

Dependințele necesare rulării aplicației sunt:

- Python >3.8
- Pachetul de Python PySide2 5.15.1
- Pachetul de Python opencv-python
- Pachetul de Python numpy
- Pachetul de Python pyqtgraph 0.11.0

Instalarea Python este dependentă de sistem și nu va fi acoperită de acest ghid. Este important ca Python și managerul de pachete "pip" să fie disponibile în linia de comandă. Pentru aceasta, utilizatorii trebuie să se asigure că directorul de instalare al Python este adăugat în variabila PATH (în cazul Linux nu ar trebui să existe această problemă).

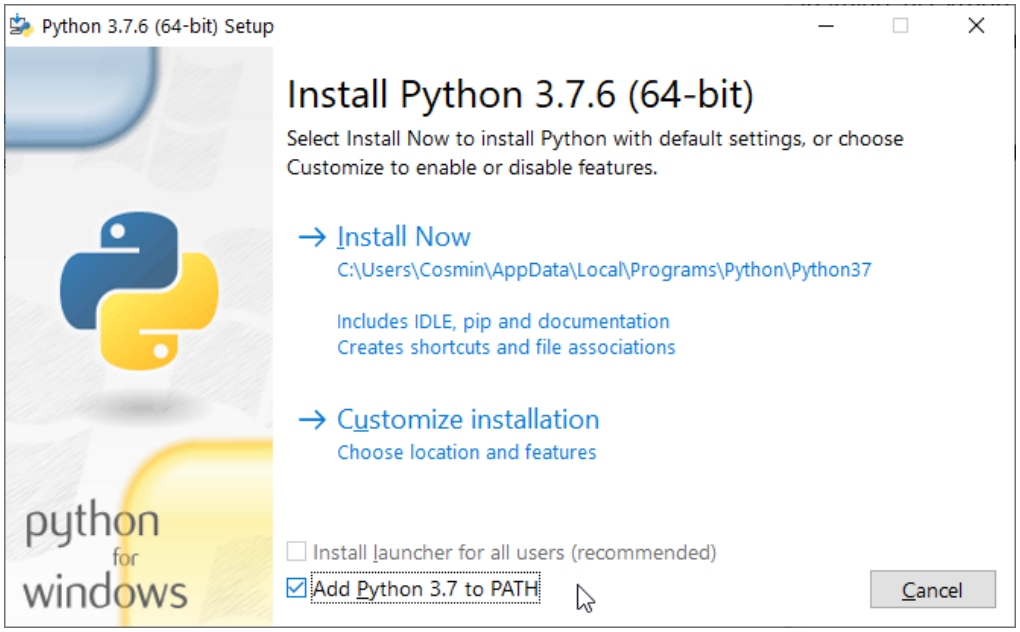

Fig. 1 – adăugarea Python la variabila PATH în installerul de Windows

Pentru instalarea pachetelor de Python necesare, vom folosi un utilitar numit „pip” care este livrat odată cu Python.

După clonarea aplicației, în folderul obținut va fi prezent un fișier numit „requirements.txt”. Acest fișier este necesar pentru instalarea dependințelor.

Pentru instalare, deschideți o consolă (Command Prompt, Terminal, PowerShell, etc.) și navigați în directorul în care se află fișierul „requirements.txt” folosind comanda „cd”. De exemplu:

```
cd C:\Users\Cosmin\Desktop\image_processing_platform
```

Apoi, pentru instalarea pachetelor de Python necesare rulați comanda următoare:

```
pip install --user -r requirements.txt
```

1.1.4 Instalare și rulare aplicație

După dezarhivarea aplicației și instalarea dependințelor, tot ce rămâne de făcut este să rulați aplicația folosind script-ul „ImageProcessingPlatform.pyw”.

În cazul unei instalari corecte de Python în Windows, aplicația ar trebui să pornească dând dublu click pe fișierul „ImageProcessingPlatform.pyw”.

În cazul Mac OS X sau Linux aplicația poate fi rulată navigând întâi în directorul aplicației folosind comanda „cd”, iar apoi executând comanda de mai jos:

```
python ImageProcessingPlatform.pyw
```

Pentru dezvoltarea algoritmilor de procesare de imagine se recomandă rularea constantă a aplicației sub un mediu de depanare (debug pe „ImageProcessingPlatform.pyw”), din motivele menționate în capitolul „1.2.1 Cunoștințe necesare”.

1.2 Ghid de utilizare

1.2.1 Cunoștințe necesare

Aplicația are ca scop scurtarea timpului de dezvoltare al algoritmilor de procesare de imagine în limbajul Python. Drept urmare, se presupune deținerea unor cunoștințe de bază ale limbajului Python (funcții, decoratori, tipuri de date, mod de lucru), numpy (tipul ndarray, mod de lucru), opencv (modul de reprezentare al unei imagini într-o matrice și particularități cum ar fi organizarea RGB în loc de BGR[1]) și în unele cazuri cunoștințe legate de Qt (pentru desenarea peste imagini se folosesc metode ale framework-ului Qt).

Aplicația nu oferă un strat de abstractizare peste toate structurile oferite de biblioteci externe. De multe ori, acesta s-ar fi dovedit reduntant. Folosirea tuturor acestor structuri externe este documentată în continuare (ex. desenarea peste imagini, array-uri numpy).

Aplicația este gândită pentru a fi rulată constant sub un mediu de depanare. Aplicația este un ajutor pentru dezvoltare, iar dezvoltarea este întotdeauna acompaniată de un debugger. Pot apărea erori sau aplicația se poate închide neașteptat în cazul în care codul Python scris de dumneavoastră este invalid sau nu poate fi interpretat. În acest caz, mesajele de eroare vor apărea doar în consola debuggerului folosit.

1.2.2 Implementarea algoritmilor de procesare de imagine

Adițional celor descrise mai jos, exemple de implementări de algoritmi de procesare de imagine pot fi găsite în fișierele:

- Application/ImageProcessingAlgorithms/PointwiseOperations.py
- Application/ImageProcessingAlgorithms/Thresholding.py

[1] Despre reprezentarea BGR în loc de RGB în OpenCV: https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/

Ordinea prezenței decoratorilor menționați mai jos înaintea semnăturii unei funcții nu este importantă.

1.2.2.1 Înregistrarea algoritmilor – RegisterAlgorithm

Algoritmii de procesare de imagine se vor implementa sub formă de funcție într-un fișier denumit modul cu extensia „.py" care se va afla în directorul „Application/ImageProcessingAlgorithms".

Fișierul (modulul[2]) poate face parte și dintr-un pachet[2] (un subdirector) aflat în directorul menționat mai sus, cu condiția ca fișierele (modulele[2]) din pachet[2] să fie importate folosind un fișier de inițializare[2] cu numele „__init__.py". [8] De exemplu, dacă dorim să creăm un pachet[2] (director) cu numele „Binarization" iar în el dorim să avem un fișier (modul[2]) cu numele „SimpleBinarization.py", fișierul de inițializare[2] „__init__.py" din directorul „Application/ImageProcessingAlgorithms/Binarization" va trebui să conțină următoarea linie (pentru ca modulul[2] din pachet să fie încărcat):

```
import SimpleBinarization
```

Platforma va încărca automat toate modulele[2] sau pachetele[2] al căror nume nu începe cu semnul „_" (underline).

În fișierele create se poate scrie orice cod valid Python, nu există limitări (se pot folosi biblioteci externe, clase, etc.).

Algoritmii se scriu sub formă de funcție într-un fișier creat în modul descris mai sus. Un algoritm poate fi spart în mai multe funcții și poate fi scris în orice mod se dorește (folosind chiar clase), singura cerință fiind ca algoritmul să aibă un punct de intrare (funcție principală) decorat[3] cu decoratorul[3] „RegisterAlgorithm". Folosirea decoratorului reprezintă declararea funcției ca punct de intrare al unui algoritm, acesta fiind automat adăugat într-un meniu din interfața grafică.

---

[2] Despre module și pachete Python: https://realpython.com/python-modules-packages/

[3] Despre decoratori în Python: https://realpython.com/primer-on-python-decorators/

Cerințe pentru o funcție validă reprezentând un punct de intrare în algoritm:

- Să fie decorată[3] de decoratorul[3] „RegisterAlgorithm" cu parametri valizi
- Să aibă cel puțin un parametru (numele său nu contează) prin care se va primi imaginea sursă sub formă de numpy array[4] (imaginea primită este o copie a sursei)

Trebuie menționat faptul că imaginea furnizată prin primul parametru este mereu o matrice bidimensională de tip numpy.ndarray dar fiecare element din matrice poate fi:

- int – pentru imaginile grayscale; reprezintă nivelul de gri al pixelului
- tuplu de 3 int – pentru imaginile color; reprezintă cele 3 canale ale pixelului în ordinea R, G, B[5] (complet opus OpenCV, folosirea în algoritmii OpenCV implică transformarea la BGR)

Pentru a putea folosi decoratorul „RegisterAlgorithm" acesta trebuie importat. Această operație se poate face utilizând următoarea secvență:

```
from Application.Utils.AlgorithmDecorators import RegisterAlgorithm
```

Decoratorul „RegisterAlgorithm" primește ca parametri 2 valori:

- Primul parametru reprezintă un nume unic (chiar dacă algoritmul se află în alt meniu acest nume trebuie să fie unic global) al algoritmului care va fi afișat în interfața grafică
- Al doilea parametru reprezintă numele meniului sau calea meniului sub care algoritmul va apărea în interfața grafică
  - Dacă meniul nu există va fi creat automat

[4] Despre numpy arrays: https://realpython.com/numpy-array-programming/

[5] Despre reprezentarea BGR în loc de RGB în OpenCV: https://www.learnopencv.com/why-does-opencv-use-bgr-color-format/

▪ Dacă meniul există, algoritmul va fi adăugat la coada listei din acel meniu

▪ Pentru adăugarea de submeniuri la meniuri, se poate folosi caracterul "/" (slash) pentru trecerea la un subnivel

De exemplu, cu primul parametru „Otsu" și al doilea parametru „Image Segmentation/Thresholding" rezultatul din interfața grafică va fi următorul:

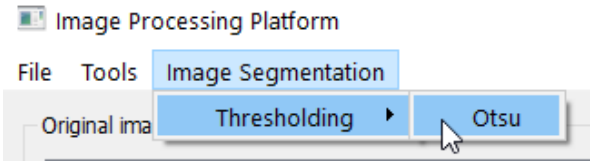

Fig. 2 – exemplu de meniu autogenerat pentru un algoritm

Pe lângă primii 2 parametri obligatorii, decoratorul „RegisterAlgorithm" mai poate primi alți parametri opționali (care trebuie să fie oferiți cu numele complet sub forma „numeparametru = valoare"):

◘ before – poate fi None sau string – dacă meniul în care algoritmul va fi pus e deja populat, în această variabilă se poate specifica înaintea cărei alte intrări să fie plasată intrarea prezentă; dacă intrarea nu există algoritmul va fi pus la finalul listei

◘ fromMainModel – listă de stringuri – dacă sunt necesare date suplimentare pe care aplicația le poate oferi, acestea pot fi cerute folosind acest parametru și primite ca parametri la funcție cu același nume; datele suplimentare existente sunt:

▪ leftClickPosition – va întoarce poziția click-ului dacă ferestrele Plotter sau Magnifier sunt deschise, altfel None – nu se recomandă folosirea în „RegisterAlgorithm"

▪ rightClickLastPositions – întoarce un deque care conține toate pozițiile în care s-a dat click dreapta în poză

Pentru folosirea variabilei „rightClickLastPositions" este necesară selectarea unuia sau mai multor puncte pe imaginea inițială folosind click dreapta. Punctele selectate vor fi marcate printr-un chenar cu un număr de

ordine. Aceste puncte pot fi șterse folosind tasta Esc. Numărul prestabilit de puncte este 4. Aceste puncte ciclează, însemnând că la selectarea celui de-al 5-lea punct, punctul cu numărul 1 este pierdut, celorlalte 3 li se scade poziția cu 1 iar noul punct va fi punctul 4.

Valoarea returnată de funcție trebuie să fie un dicționar care poate conține următoarele chei:

- originalImage – numpy.ndarray – suprascrie imaginea sursă
- processedImage – numpy.ndarray – suprascrie imaginea rezultat
- originalImageOverlayData – tuplu cu 3 obiecte (QPainterPath[6], QColor[7], int[8] – calea, culoarea, lățimea) – reprezintă parametri folosiți de program pentru a desena peste imaginea sursă fără a o modifica
- processedImageOverlayData – identic cu precedentul

Pentru utilizarea funcționalității de desenare peste imagini se vor folosi cheile „originalImageOverlayData" și „processedImageOverlayData" din dicționarul returnat. Aceste variabile au ca valoare un tuplu format din 3 componente:

- Forma care se dorește desenată folosind clasa QPainterPath[6] oferită de Qt; a se consulta documentația Qt pentru ghidul de utilizare al acesteia[6]
- Culoarea care se dorește a fi folosită la desenare[7]
- Lățimea liniei cu care se desenează ca număr natural[8]

Trebuie menționat faptul că algoritmul nu va fi apelat în cazul în care nu există o imagine sursă încărcată în program, fiind afișat un mesaj de avertizare în această privință.

---

[6] Documentație Qt C++ QPainterPath: https://doc.qt.io/qt-5/qpainterpath.html

[7] Documentație Qt C++ QColor: https://doc.qt.io/qt-5/qcolor.html

[8] Folosit pentru a seta lățimea unui QPen folosind setWidth: https://doc.qt.io/qt-5/qpen.html#setWidth

Un scurt exemplu al unui algoritm de folosire a decoratorului „RegisterAlgorithm" este:

```
from Application.Utils.AlgorithmDecorators import RegisterAlgorithm

# acest algoritm nu face nimic
@RegisterAlgorithm('Alg 1', 'Menu')
def test1(image):
  return None

@RegisterAlgorithm('Alg 2', 'Menu', before='Alg 1', fromMainModel=['rightClickLastPositions'])
def test2(image, rightClickLastPositions):
  # image are tipul numpy.ndarray
  # rightClickLastPositions are tipul tuple
  resultImage = ... # operații cu imaginea
  numberOfRightClicks = len(rightClickLastPositions)
  if numberOfRightClicks == 0:
    return {
      'processedImage': resultImage
    }
  else:
    return {}
```

1.2.2.2 Date de intrare suplimentare – InputDialog

Acest decorator poate fi utilizat doar în combinație cu decoratorul „RegisterAlgorithm".

Deseori, algoritmii au nevoie de date de intrare suplimentare unei imagini. Pentru citirea acestora de la tastatură, platforma pune la dispoziție un decorator prin care acestea pot fi cerute.

Modul în care funcționează decoratorul este următorul: la rularea algoritmului din interfața grafică, utilizatorul este întâmpinat de un dialog în care se cerere completarea câmpurilor specificate de programator în decorator. După completarea acestora, câmpurile sunt validate și

transformate în structurile de date cerute (totul este citit ca string). Dacă transformarea în structurile de date are loc cu succes, datele sunt trimise mai apoi la algoritm care le primește ca parametru cu numele cu care au fost declarate în decorator. Altfel este afișat un mesaj de eroare care indică eșecul transformării.

Pentru a putea folosi decoratorul „InputDialog" acesta trebuie importat. Această operație se poate face utilizând următoarea secvență:

```
from Application.Utils.InputDecorators import InputDialog
```

Decoratorul „InputDialog" primește ca parametru un număr nelimitat de valori, după structura următoare (parantezele pătrate indică elemente a căror prezență e opțională):

```
@InputDialog(
  nume_variabilă_1=callable_variabilă_1 [,
  nume_variabliă_2=callable_variabilă_2, ...])
```

Există și o structură alternativă în callable-ul de după egal devine un tuple cu 2 parametri: primul parametru este un string reprezentând textul afișat în dreptul căsuței pentru input, iar al doilea este callable-ul, după cum urmează:

```
@InputDialog(
  nume_variabilă_1=('text', callable_variabilă_1) [,
  nume_variabliă_2=callable_variabilă_2, ...])
```

Numele variabilelor trebuie să respecte convențiile de nume ale limbajului Python[9]. „Callable-ul" variabilelor reprezintă o entitate care poate fi apelată. În Python, totul este un obiect iar distincția dintre clasă și metodă

[9] Convenții de nume în Python: https://docs.python.org/3.7/reference/lexical_analysis.html#identifiers

nu este importantă și poate fi ignorată[10]. O particularitate a limbajului este reprezentată de posibilitatea de a apela clase în loc de a le instanția[10].

Astfel, „callable-ul" de după semnul „=" trebuie să fie orice obiect care acceptă un string ca parametru pentru a fi inițializat (ex. int, float, str, etc.), chiar și clase scrise de utilizator cum ar fi parsere XML.

Un scurt exemplu al unui algoritm de folosire a decoratorului „InputDialog" este:

```
from Application.Utils.AlgorithmDecorators import RegisterAlgorithm
from Application.Utils.InputDecorators import InputDialog

# ordinea decoratorilor nu contează
@InputDialog(requested_value_1=int, requested_value_2=float)
@RegisterAlgorithm(„Test Alg", „Test Menu")
def test(image, requested_value_1, requested_value_2):
  # image are tipul numpy.ndarray
  # requested_value are tipul int
  return {}
```

În urma rulării algoritmului de mai sus suntem întâmpinați cu următorul dialog (valoarea 1 fi transformată din str în int iar 2 din str în float):

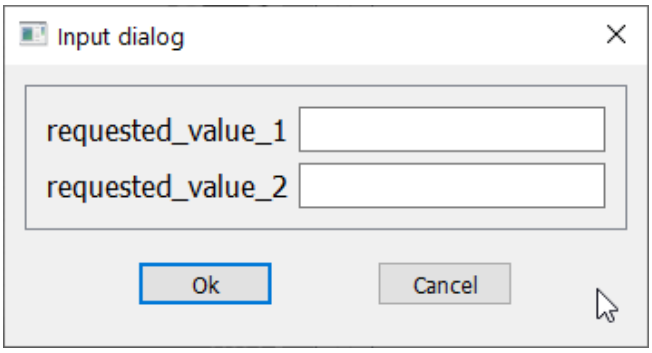

Fig. 3 – exemplu de dialog pentru date de intrare

[10] Clase versus funcții: https://treyhunner.com/2019/04/is-it-a-class-or-a-function-its-a-callable/

1.2.2.3 Mesaje de ieșire - OutputDialog

Acest decorator poate fi utilizat doar în combinație cu decoratorul „RegisterAlgorithm".

Deseori dorim să afișăm diverse mesaje în urma rulării unui algoritm. Fie putem indica o eroare, fie putem afișa o valoare internă care a fost calculată, fie algoritmul analizează un set de date și oferă un rezultat numeric sau un text.

Decoratorul „OutputDialog" primește un singur parametru: titlul ferestrei care va fi afișată.

Decoratorul introduce un nou parametru care poate fi prezent în dicționarul returnat de funcția implementată. Acesta se numește „outputMessage" și are tipul „str". Mesajul poate fi transmis în mod opțional. În cazul în care parametrul menționat mai sus lipsește, fereastra de dialog nu va fi afișată.

Pentru a putea folosi decoratorul „OutputDialog" acesta trebuie importat. Această operație se poate face utilizând următoarea secvență:

```
from Application.Utils.OutputDecorators import OutputDialog
```

Un scurt exemplu al unui algoritm de folosire a decoratorului „OutputDialog” este:

```
from Application.Utils.AlgorithmDecorators import RegisterAlgorithm
from Application.Utils.InputDecorators import InputDialog
from Application.Utils.OutputDecorators import OutputDialog

# ordinea decoratorilor nu contează
@InputDialog(number=int)
@OutputDialog(„Dialog Title”)
@RegisterAlgorithm(„Test Alg”, „Test Menu”)
def test(image, number):
  # image are tipul numpy.ndarray
  # number are tipul int
  if number == 0:
    return {
      „outputImage”: image
    }
  else:
    #mesajul se afișează doar dacă numărul primit nu este 0
    return {
      „outputImage”: image
      „outputMessage”: „a longer test message”
    }
```

În urma rulării algoritmului de mai sus și a introducerii valorii 1 în dialogul de intrare suntem întâmpinați cu următorul dialog de ieșire:

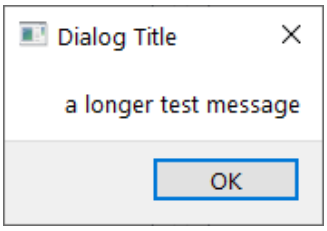

Fig. 4 – exemplu de dialog pentru date de ieșire

1.2.3 Implementarea algoritmilor de plotting – PlotterFunction

Adițional celor descrise mai jos, exemple de implementări de algoritmi de plotting pot fi găsite în fișierul „Application/PlottingAlgorithms/Basic.py".

Unul din modurile de a reprezenta informație extrasă dintr-o imagine este plotarea datelor sau a unor funcții bazate pe aceste date. Un plot este o reprezentare a unui set de date pe un plan cu un sistem de coordonate carteziene ca ghid. Platforma oferă această capabilitate prin intermediul unui decorator numit „PlotterFunction".

Algoritmii de plotting se vor implementa sub formă de funcție într-un fișier denumit modul cu extensia „.py" care se va afla în directorul „Application/PlottingAlgorithms".

Algoritmilor de plotting li se aplică aceleași condiții de structurare în pachete și inițializare a acestora ca în cazul algoritmilor de procesare de imagine prezentați în capitolul „1.2.2.1 Înregistrarea algoritmilor – RegisterAlgorithm".

De asemenea, la fel ca în cazul algoritmilor de imagine, algoritmii de plotting sunt detectați automat și introduși în interfața grafică în unealta de plotting descrisă în capitolul „1.2.6 Utilizarea plotterului", cu diferența folosirii unui decorator numit „PlottingFunction" și a unei structuri de date numita „PlottingData".

Cerințe pentru o funcție validă reprezentând un punct de intrare în algoritm:

- Să fie decorată de decoratorul „PlottingFunction" cu parametri valizi
- Să aibă cel puțin un parametru (numele său nu contează) prin care se va primi imaginea sursă sub formă de numpy array (imaginea primită este o copie a sursei)
- Să returneze None sau o listă care conține obiecte de tip „PlottingData"

Pentru a putea folosi decoratorul „PlotterFunction” și tipul de date „PlottingData” acestea trebuie importate. Această operație se poate face utilizând următoarea secvență:

```
from Application.Models.PlottingData import PlottingData
from Application.Utils.PlotterDecorators import PlotterFunction
```

Decoratorul „PlotterFunction” primește un singur parametru: numele funcției de plotting ca str.

Pe lângă primul parametru obligatoriu, decoratorul „PlotterFunction” mai poate primi alți parametri opționali (care trebuie să fie oferiți cu numele complet sub forma „numeparametru = valoare”):

- fromMainModel – listă de stringuri – dacă sunt necesare date suplimentare pe care aplicația le poate oferi, acestea pot fi cerute folosind acest parametru și primite ca parametri la funcție cu același nume; datele suplimentare existente sunt:
  - leftClickPosition – va întoarce poziția pe care s-a dat click stânga cât timp fereastra de plotting este deschisă
  - rightClickLastPositions – întoarce un deque care conține toate pozițiile în care s-a dat click dreapta în imagine

- computeOnClick – bool – în cazul în care este True, funcția de plotting va fi recalculată la fiecare click stânga efectuat pe imagine în timp ce fereastra de plotting este deschisă
- computeOnImageChanged – bool – în cazul în care este True, funcția de plotting va fi calculată de fiecare dată când se încarcă o imagine nouă, click stânga neavând niciun efect

Valoarea returnată de funcție trebuie să fie None sau un dicționar care poate conține doar cheia „plottingDataList” cu valoarea o listă care conține elemente de tip „PlottingData”.

Fiecare element de tip „PlottingData” reprezintă o nouă linie sau curbă desenată pe plot. Această clasă se instanțiază utilizând următoarele câmpuri:

- name – str – numele corespunzător liniei sau curbei curente
- y – Iterable – un obiect de tip iterabil (list, tuple, dict, ndarray, etc.) care conține valorile corespondente axei Y
- x – Iterable – identic cu Y dar pentru axa X – este un parametru opțional; în lipsa sa se presupune că avem o listă cu elementele naturale din intervalul [0, len(y)-1]
- pen – str, tuple sau altele[11] – descrie culoarea cu care se desenează curba
  - pentru str: poate avea una din valorile următoare: r, g, b, c, m, y, k, w
  - pentru tuple: este un tuplu cu 3 sau 4 elemente reprezentând forma R, G, B și opțional A (opacitate) a unei culori

Algoritmul de plotting va fi apelat de 2 ori, o dată pentru fiecare imagine (sursă și rezultat).

Funcțiile de plotting sunt apelate doar la deschiderea ferestrei de plotting (pentru primul algoritm din listă), la schimbarea algoritmului sau la apăsarea unui click în cazul în care algoritmul este setat să reacționeze astfel prin parametrul „computeOnClick”. Acest proces se numește „lazy initialization” și îmbunătățește performanța aplicației .

O diferență majoră constă în faptul că în cazul algoritmilor de plotting, parametrul „image” poate fi None. Aceasta este o alegere conștientă de design din următorul motiv: acelasi algoritm este apelat pentru fiecare din cele 2 imagini din aplicație, iar imaginea sursă este disponibilă în timp ce imaginea rezultat poate nu a fost generată încă. Este responsabilitatea programatorului să verifice dacă a fost primită o imagine.

[11] PyQtGraph Colors: http://www.pyqtgraph.org/documentation/functions.html#pyqtgraph.mkColor

Tipul de date și formatul imaginii primite sunt identice cu cele ale algoritmilor de procesare de imagine.

Exemplu de algoritm care plottează valorile de pe rândul selectat cu click stânga în ambele imagini (dacă sunt disponibile) doar dacă imaginea încărcată e în format grayscale:

```
from Application.Models.PlottingData import PlottingData
from Application.Utils.PlotterDecorators import PlotterFunction

@PlotterFunction(
  name="Plot row values if gray",
  fromMainModel=["leftClickPosition"],
  computeOnClick=True
)
def plotRowValuesGray(image, leftClickPosition):
  plotDataItemsList = []

  # dacă imaginea nu există sau dacă nu s-a dat click stânga sau dacă imaginea
  # nu are 2 dimensiuni (x și y pentru grayscale, RGB pentru color)
  if image is None or leftClickPosition is None or len(image.shape) != 2:
    return {
      'plottingDataList': []
    }

  plotName = 'Gray level'
  plottingData = PlottingData(plotName, image[leftClickPosition.y], pen='r')
  plotDataItemsList.append(plottingData)

  return {
    'plottingDataList': plotDataItemsList
  }
```

Rezultatul algoritmului de mai sus în fereastra de plotting folosind 2 imagini arbitrare ca sursă și rezultat:

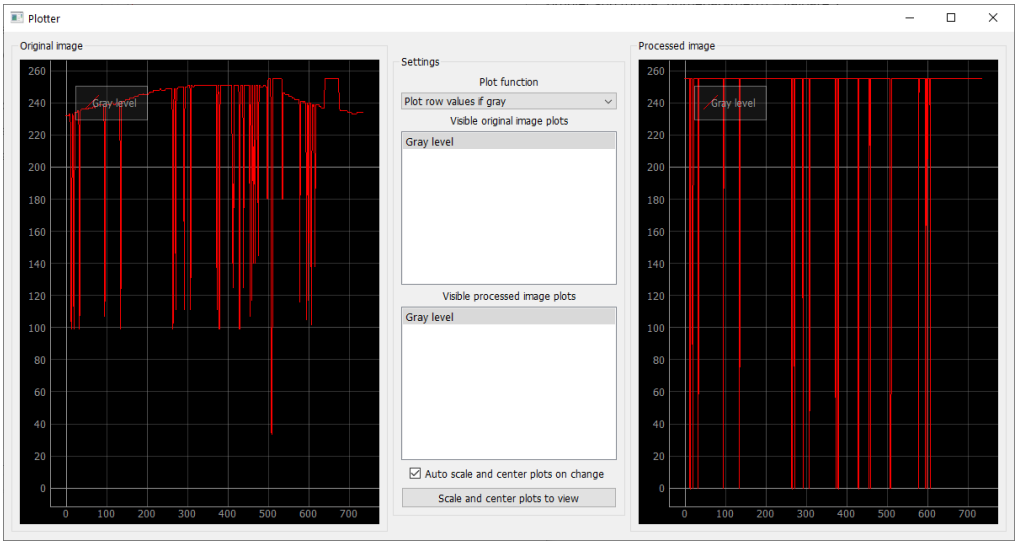

Fig. 5 – fereastra de plotting rulând algoritmul de la pagina anterioară

1.2.4 Încărcarea și procesarea imaginilor

Primul pas pentru procesarea unei imagini constă în încărcarea unei imagini sursă. Acest lucru se poate face folosind opțiunile „Load grayscale image" și „Load color image" din meniul „File" al aplicației.

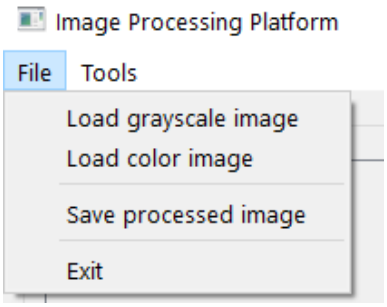

Fig. 6 – meniul File al aplicației

În urma selectării uneia din cele două opțiuni va fi afișat un dialog de alegere al imaginii sursă. În functie de opțiunea de încărcare selectată, imaginea va fi încărcată astfel:

- Pentru "Load grayscale image": imaginea va fi încărcată ca imagine în tonuri de gri pe 8 biți (imaginile color vor fi automat transformate); structura internă a imaginii va fi de tipul numpy.ndarray cu 1 valoare pentru fiecare element al matricei
- Pentru "Load color image": imaginea va fi încărcată ca imagine RGB pe 8 biți (imaginile grayscale vor fi automat transformate); structura internă a imaginii va fi de tipul numpy.ndarray cu 3 valori pentru fiecare element al matricei (componentele color în ordinea RGB)

Formatele de imagine suportate la încărcare sunt:

- Bitmap (.bmp, .dib)
- JPEG (.jpeg, .jpg, .jpe)
- JPEG 2000 (.jp2)
- PNG (.png)
- WebP (.webp)
- Netpbm (.pbm, .pgm, .ppm)
- Sun Microsystems Raster (.ras, .sr)
- TIFF (.tiff, .tif)

Pentru procesarea imaginii sursă se poate selecta acum unul din algoritmii disponibili în bara de meniu din partea de sus a aplicației.

Pentru încărcarea imaginii rezultat ca imagine sursă se va folosi opțiunea „Save as original image" din partea dreaptă a meniului aplicației.

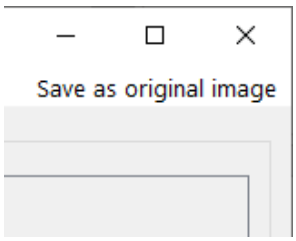

Fig. 7 – butonul de salvare a imaginii rezultat ca imagine sursă

Pentru salvarea imaginii rezultat se va folosi opțiunea „Save processed image" din meniul „File". Formatele de imagine suportate pentru salvare sunt aceleași ca cele pentru încărcare enumerate mai sus.

1.2.5 Utilizarea lupei (magnifier)

Platforma pune la dispoziția utilizatorului o unealtă de tip lupă cu care se pot inspecta imaginile la nivel de pixel. Pentru deschiderea acesteia se va selecta meniul „Tools" iar apoi opțiunea „Magnifier".

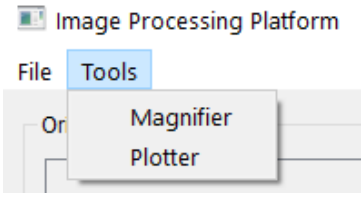

Fig. 8 – meniul Tools al aplicației

Pentru ca unealta să afișeze informații, în aplicație trebuie să fie încărcată cel putin imaginea sursă. Apoi, cu fereastra „Magnifier" deschisă, este necesară apăsarea click stânga pe una dintre imagini pentru a fi afișat conținutul din locul apăsării la nivel de pixel. Locul apăsării se va marca peste fiecare imagine disponibilă, pentru a fi cunoscută zona analizată.

Unealta poate afișa valorile pixelilor în 5 moduri de culoare:

- RGB (max. 255, 255, 255)
- HSL (max. 359°, 100%, 100%) – H=-1 pentru culori acromatice[12]
- HSV (max. 359°, 100%, 100%) – H=-1 pentru culori acromatice[13]
- CMYK (max. 359°, 100%, 100%)
- Grayscale (max. 255)

[12] Detalii de implementare QColor pentru HSV: https://doc.qt.io/qt-5/qcolor.html#the-hsv-color-model

[13] Detalii de implementare QColor pentru HSL: https://doc.qt.io/qt-5/qcolor.html#the-hsl-color-model

La închiderea uneltei marcajul zonei analizate va fi eliminat de pe imaginile din aplicație.

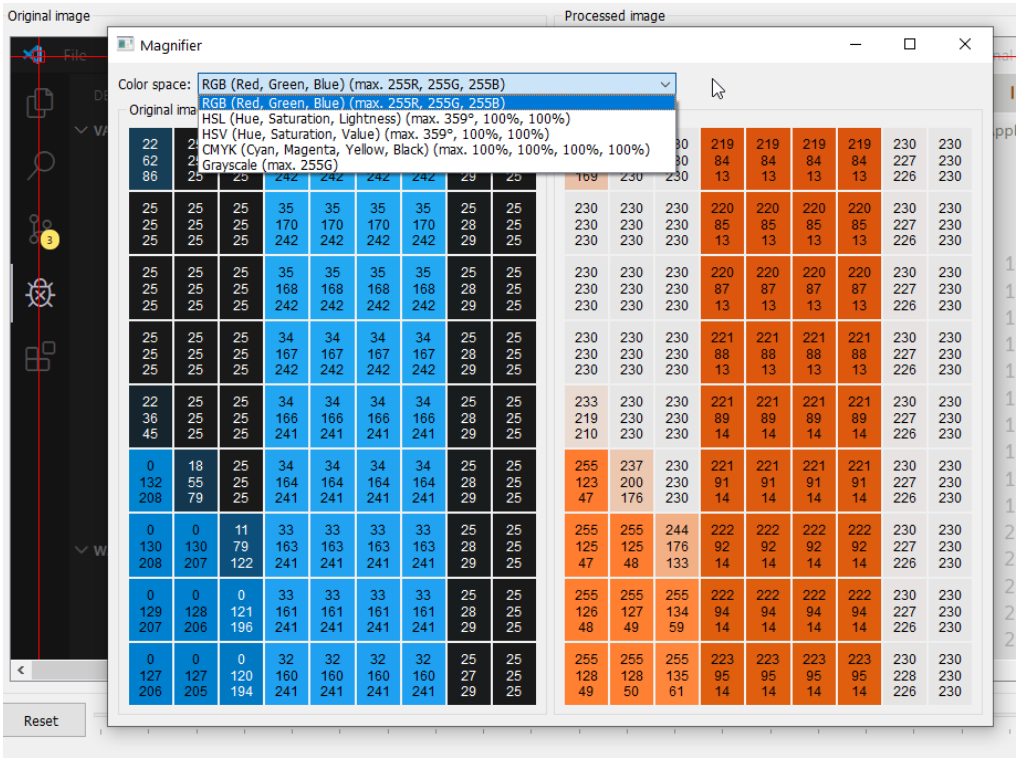

Fig. 9 – unealta de tip lupă cu opțiunile sale

1.2.6 Utilizarea plotterului

Platforma pune la dispoziția utilizatorului un plotter cu algoritmi care pot fi definiți conform capitolului „1.2.3 Implementarea algoritmilor de plotting – PlotterFunction".

O fotografie a ferestrei poate fi observată în Fig. 5 de la pagina 20.

Plotterul oferă următoarele funcționalități:

- Axele nu au o limită finită a valorilor reprezentabile
- Posibilitatea schimbării algoritmului de plotting utilizat
- Posibilitatea afișării selective a anumitor ploturi folosind meniul central

- Scalare și deplasare automată în zona valorilor de pe grafic cu posibilitatea de dezactivare și rulare manuală
- Zoom (folosind rotița mouse-ului) și navigare în 4 direcții (prin tragere)

În funcție de implementare, unii algoritmi necesită un click suplimentar pe imaginea sursă sau cea rezultată. Pentru a oferi aceste date, este necesară apăsarea pe una din imagini în timp ce fereastra „Plotter" este deschisă. Zona selectată va fi evidențiată printr-un marcaj care va dispărea la închiderea ferestrei.

1.2.7 Caracteristici minore

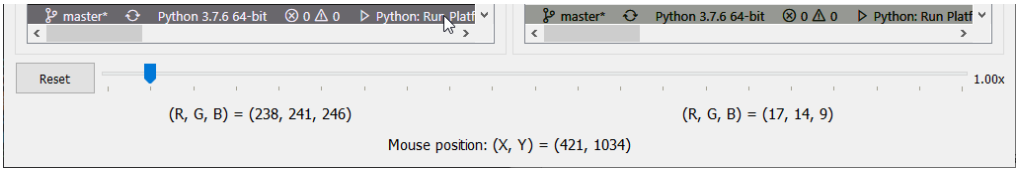

Fig. 10 – zona caracteristicilor minore

Aplicația poate mări sau micșora pozele încărcate prin sliderul din josul ferestrei. Acest efect poate fi resetat folosind butonul „Reset".

În timpul acțiunii de „hover" (plutire) a cursorului deasupra uneia din imaginile încărcate se vor afișa coordonatele curente precum și valoarea pixelului curent.

O altă caracteristică prezentă este sincronizarea barelor de scroll ale imaginilor în cazul în care imaginile au aceeași dimensiune (mai precis se verifică lățimile între ele și înălțimile între ele, nu numărul total de pixeli).

De asemenea, o parte din caracteristicile aplicației pot fi modificate prin intermediul fișierului cu setări „Application/Settings.py".

În fereastra principală putem modifica:

- zoomMinimumValue – valoarea minimă a zoom-ului pe imagine
- zoomMaximumValue - valoarea maximă a zoom-ului pe imagine

- zoomSingleStep – valoarea cu care zoom-ul va fi incrementat/decrementat pentru un pas efectuat cu săgețile tastaturii
- zoomPageStep – valoarea cu care zoom-ul va fi incrementat/decrementat pentru un pas efectuat cu butoanele PageUp/PageDown ale tastaturii
- zoomDefaultValue – valoarea inițială a zoom-ului
- zoomTicksInterval – intervalul de afișare al ghidurilor vizuale de-a lungul sliderului

În fereastra uneltei tip lupă putem modifica:

- gridSize – mărimea matricei de pixeli (număr pozitiv, impar)
- textThreeRowsHeightPadding – spațierea textului pe pixel în cazul prezenței a 3 valori (RGB, HSL, HSV)
- textFourRowsHeightPadding – spațierea textului pe pixel în cazul prezenței a 4 valori (CMYK)
- textFontSize – mărimea fontului în puncte

Funcționalitatea de click dreapta pe imaginea inițială poate fi și ea ajustată folosind următoarele variabile:

- aroundClickSquareSize – mărimea laturii pătratului (în pixeli) care marchează zona click-ului
- numberOfClicksToRemember – numărul de click-uri care se memorează și afișează
- showClickOrder – decide dacă click-urile sunt afișate pe imagine
- clickOrderFontSize – mărimea fontului pentru scrierea numărului de ordine

1.2.8 Îmbunătățirea performanței algoritmilor

1.2.8.1 Python List Comprehension[14]

Suplimentar modului clasic de a itera prin colecții de date și a genera rezultate pe baza acestora folosind structuri repetitive imbricate, Python a introdus în versiunea 3.0 capabilitatea de a scrie acest cod într-un mod succint și mult mai ușor de înțeles prin „list comprehensions".

Această nouă sintaxă oferă și o imbunătățire a timpului de rulare, prin prisma folosirii unor funcții care execută cod C compilat.

Spre exemplu, scrierea clasică a algoritmului pentru extragerea multiplilor de 3 dintr-o listă este următoarea:

```
def extract_mod_3(input):
  result = []
  for number în input:
    if number % 3 == 0:
      result.append(number)
  return result
```

Scrierea cu list comprehension este următoarea:

```
def extract_mod_3_lc(input):
  return [number for number în input if number % 3 == 0]
```

Timpul de execuție al fiecărui algoritm cu n=10000000 elemente în input:

- Fără list comprehension: 0.69284 secunde
- Cu list comprehension: 0.56702 secunde

[14] Python List Comprehension: https://realpython.com/list-comprehension-python/

1.2.8.2 Vectorizare cu numpy[15]

Alegerea utilizării bibliotecii numpy a fost făcută din considerentul sporului de performantă obținut în momentul prelucrării unor seturi de date foarte mari folosind această unealtă.

Majoritatea operațiilor și operatorilor numpy sunt implementați în limbajul C în cod optimizat, acest cod utilizând capacitățile SIMD ale procesoarelor moderne descrise în capitolul „**Error! Reference source not found. Error! Reference source not found.**".

Luăm ca exemplu următorul algoritm: să se însumeze două liste cu „n" elemente; prima listă conține primele „n" pătrate ale numerelor naturale începând de la 0, iar a doua listă conține primele „n" cuburi ale numerelor naturale începând de la 0. [9]

În cazul în care „n" este suficient de mare (de exemplu, 1.000.000) diferența de performanța este de ordinul zecilor de ori în favoarea codului numpy.

Codul Python folosind list comprehension:

```
def square_and_root_sum_py(n):
  a = [i ** 2 for i în range(n)]
  b = [i ** 3 for i în range(n)]
  return [a[i] + b[i] for i în range(n)]
```

Codul Python folosind numpy:

```
def square_and_root_sum_numpy(n):
  a = numpy.arange(n) ** 2
  b = numpy.arange(n) ** 3
  return a + b
```

Timpul de rulare al implementărilor pentru n=1000000 este:

[15] Vectorizare cu numpy: https://realpython.com/numpy-array-programming/#what-is-vectorization

- square_and_root_sum_py: 0.8356481 secunde
- square_and_root_sum_numpy: 0.0110970 secunde

Se poate observa astfel o îmbunătățire a timpului de execuție de 80 de ori. Această diferență va crește pe măsură ce „n" devine mai mare.

1.2.9 Limitări ale aplicației

Pe parcursul dezvoltării aplicației au fost luate în calcul multe variante de design. În urma analizei acestora s-a decis introducerea cu bună știință a unor limitări în aplicație pentru facilitatea implementării design-ului și pentru a oferi utilizatorului un mediu cât mai simplu de dezvoltare. Este important ca aceste limitări să fie cunoscute pentru a înțelege comportamentul aplicației.

Prima limitare este lipsa suportului pentru transparență în imaginile încărcate. Această limită este impusă din lipsa necesității unei astfel de funcționalități în mediul pentru care a fost dezvoltată aplicația. Prezența canalului „alpha" în matricele interne nu ar fi oferit niciun beneficiu didactic în raport cu programa actuală a cursului. Lipsa acestui suport a crescut viteza de dezvoltare a aplicației și a redus dificultatea utilizării sale.

A doua limitare majoră este suportul scăzut pentru capturarea și afișarea erorilor. Decizia de a oferi posibilitatea „injectării" de cod Python direct în aplicație crește riscul apariției greșelilor de sintaxă sau de logică. Acest cod greșit se poate afla în imposibilitatea de a fi interpretat și astfel poate duce la închiderea neașteptată a aplicației.

O parte din erorile introduse prin codul scris nu pot fi „capturate" folosind design-ul curent, aceste erori manifestându-se la nivelul interpretorului Python. Astfel, orice încercare de corectare a erorii este în zadar deoarece un interpretor închis nu mai poate evalua codul corector. Folosirea bibliotecilor implementate în limbajul C (numpy, OpenCV) agravează acest efect, aceste biblioteci fiind mult mai predispuse la erori fatale în timpul rulării prin încercarea procesării unor date nevalidate.