Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

Ghid de utilizare
Framework destinat procesării imaginilor digitale

Autor: Lixandru Andreea-Bianca
Pepene Adina-Florentina

Braşov, 2021

Cuprins

ABSTRACT

Framework-ul a fost dezvoltat pentru a oferi studenţilor de la facultatea de Matematică şi Informatică, din cadrul Universităţii Transilvania, Braşov posibilitatea să dezvolte algoritmi pentru procesarea imaginilor digitale într-un mod cât mai rapid.

Acest framework oferă multiple posibilităţi de a vizualiza date despre imagini, precum: imaginea originală şi cea procesată, imaginea redimensionată, nivelul de culoare/gri din imagine. Mai mult de atât, aplicaţia conţine şi o fereastră de magnifier prin intermediul căreia pot fi observaţi pixelii dintr-o zonă selectată de către utilizator.

Design-ul aplicaţiei a fost gândit cât mai simplu, astfel încât să îi permită utilizatorului să adauge uşor şi rapid butoanele în meniu, algoritmii implementaţi, să poată încărca simplu imaginile şi să le redimensioneze.

Capitolul 1

Prezentarea framework-ului

1.1 Cum descărcăm aplicaţia?

Framework-ul se află într-un repository public la adresa: Framework PI, de unde se poate clona sau descărca un folder ZIP ce conţine framework-ul. Aplicaţia este cross-platform, fiind realizată în limbajul de programare C#. Cerinţele minime necesare pentru a putea folosi aplicaţia sunt: utilizarea unui IDE ce permite folosirea limbajului de programare C#. Pentru acest framework nu există nici un fel de dependinţă.

1.2 Arhitectura framework-ului

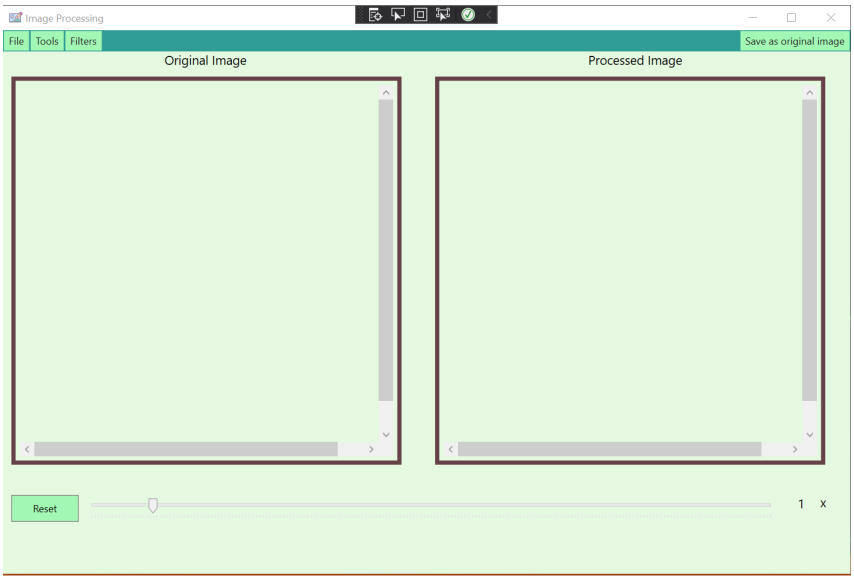
Această aplicaţie cuprinde două proiecte: *ImageProcessingFramework* şi *ImageProcessingAlgorithms*. Primul proiect conţine implementarea framework-ului, iar al doilea proiect este destinat utiliztaorului pentru a putea crea algoritmii doriţi.

Pentru proiectul *ImageProcessingFramework* a fost folosită o arhitectură de tipul MVVM(Model-View-ViewModel). Acest tip de arhitectură facilitează separarea dezvoltării interfeţelor grafice(View), de dezvoltarea logicii(Model), astfel încât vizualizarea să nu depindă de logică. View-Model este responsabil pentru expunerea obiectelor din model în aşa fel încât obiectele să fie uşor de gestionat şi de prezentat.

În cadrul celui de-al doliea proiect, *ImageProcessingAlgorithm*, se află folder-ul şi clasa Tools ce permit adăugarea algoritmilor implementaţi de către utilizator sub forma unor metode. Pentru a putea folosi algoritmul şi pentru a vedea rezultatul filtrului aplicat este nevoie să adăugăm un MenuItem în meniul aplicaţiei principale, care se află în proiectul *ImageProcessingFramework*, iar acest procedeu va fi detaliat în capitolul următor.

1.3 Design

În următoarea imagine este prezentă pagina principală a framework-ului. Iar în cele ce urmează vor fi prezentate, pe rând, butoanele din această fereastră.



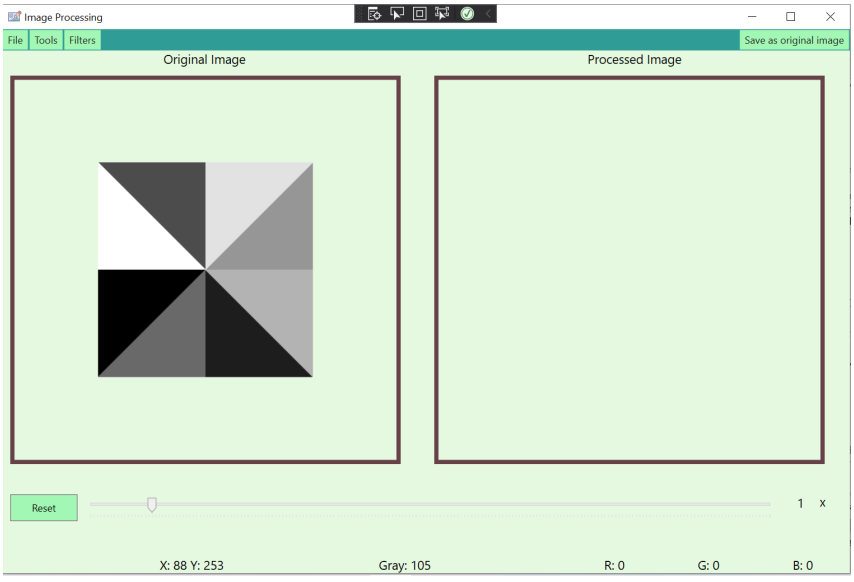
Pentru început, se pot remarca cele două zone ale ferestrei în care vor apărea imaginea originală şi cea procesată. Acestea sunt de fapt două Label-uri peste care a fost pus un ScrollView-er şi un Canvas.

Atunci când este încărcată o imagine în această fereastră, în partea de jos vor apărea valorile x şi y ale poziţiei mouse-ului pe imagine şi valoarea de gri, respectiv cea de RGB pentru o imagine color, corespunzătoare locului unde se află mouse-ul la acel moment.

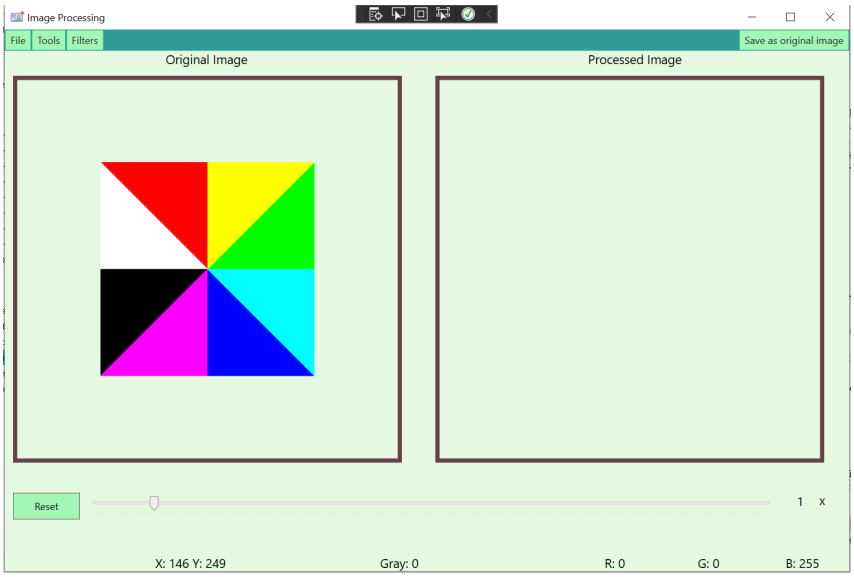
Se mai pot observa butoanele principale:

1. File

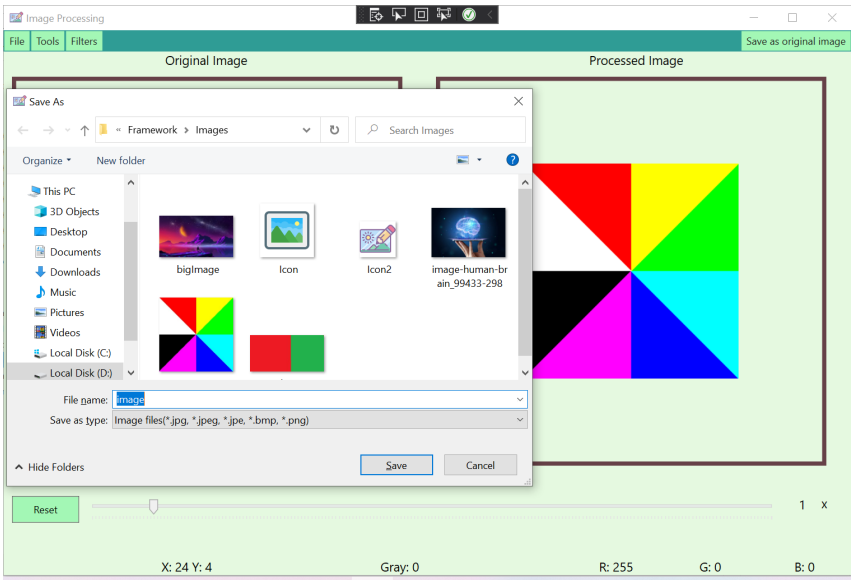
    - Load grayscale image - din fereastra Explorer se selectează imaginea ce se doreşte a fi încărcată in varianta alb-negru;



    - Load color image - din fereastra Explorer se poate încărca o imagine color spre a fi procesată;



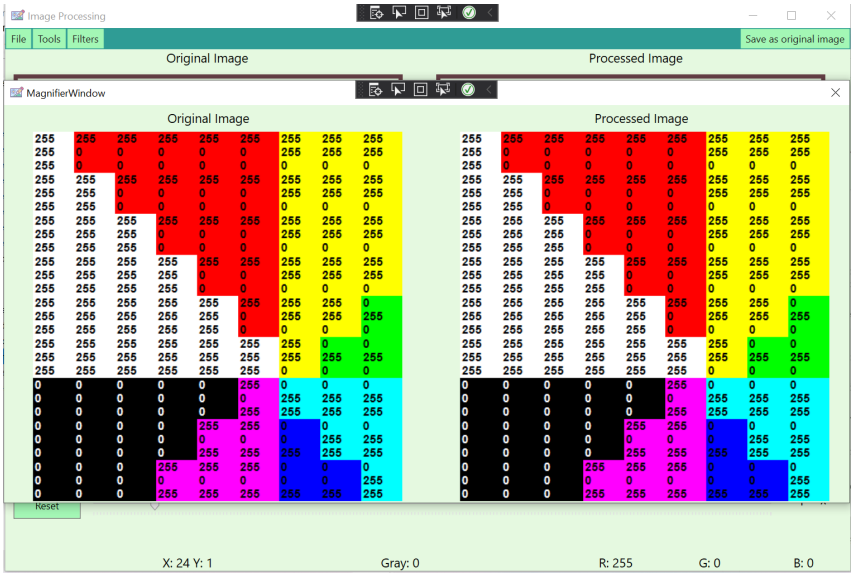
- Save processed image - imaginea procesată cu ajutorul framework-ului poate fi salvată în calculatorul utilizatorului;



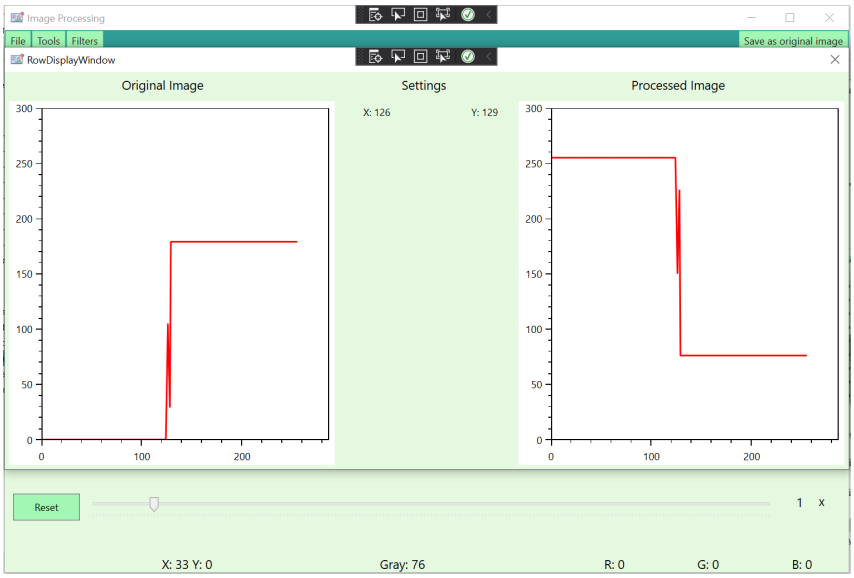
- Exit - va închide fereastra principală precum şi celelalte ferestre deschise posibil anterior.

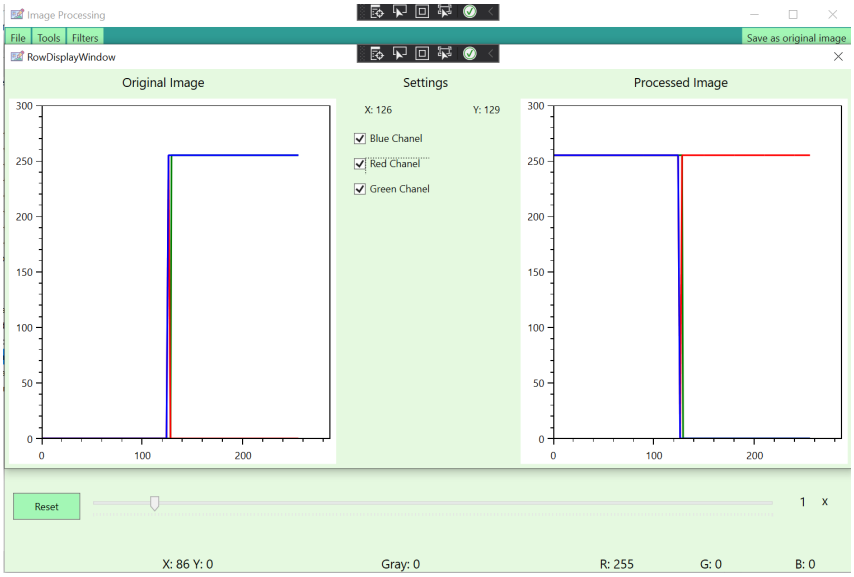
2. Tools

- Magnifier - va deschide o nouă fereastră care va oferi detalii despre imaginea color sau cea gri, precum nuanţa culorii şi valoarea acesteia, pentru un dreptunghi de 9x9 creat în jurul punctului unde utilizatorul a dat click cu mouse-ul;

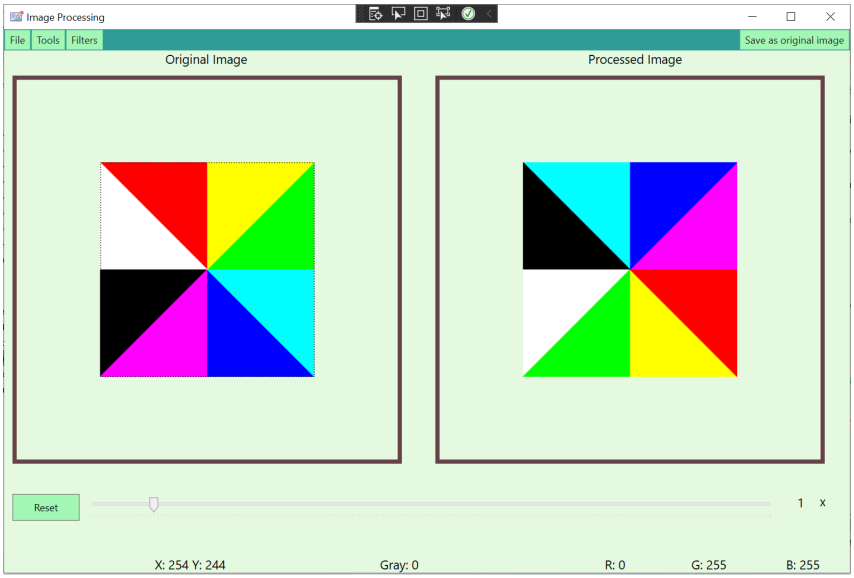


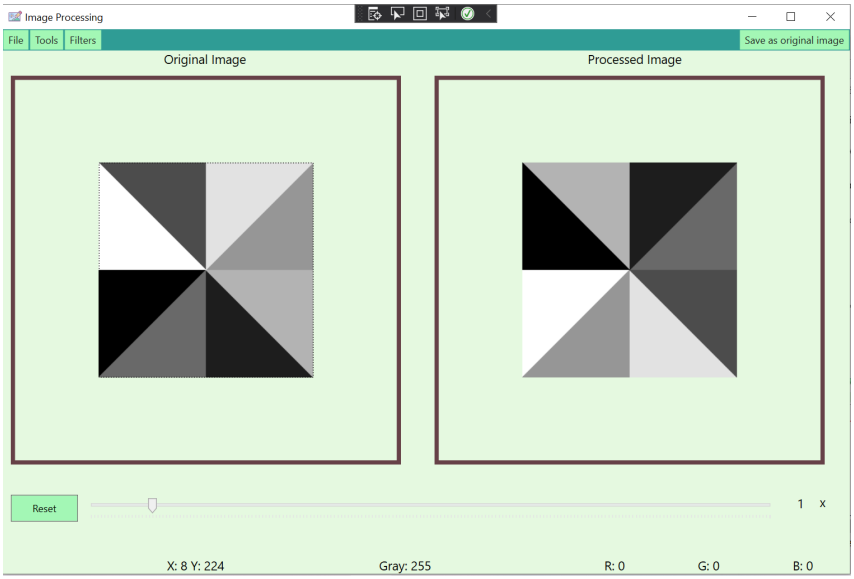
- GLevelsRow - deschide o nouă fereastră în care se desenează un grafic cu nivelurile de gri, respectiv RGB, de pe linia orizontală creată acolo unde a fost dat click pe imagine;



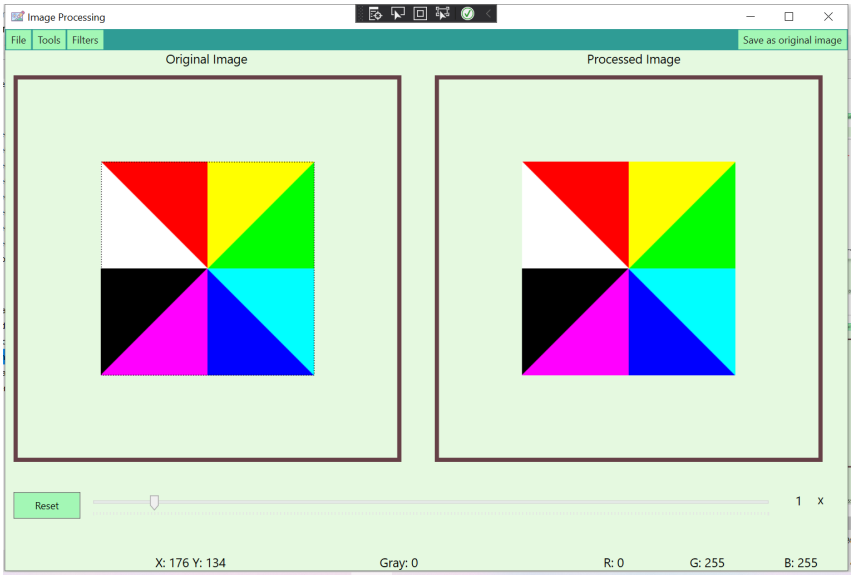


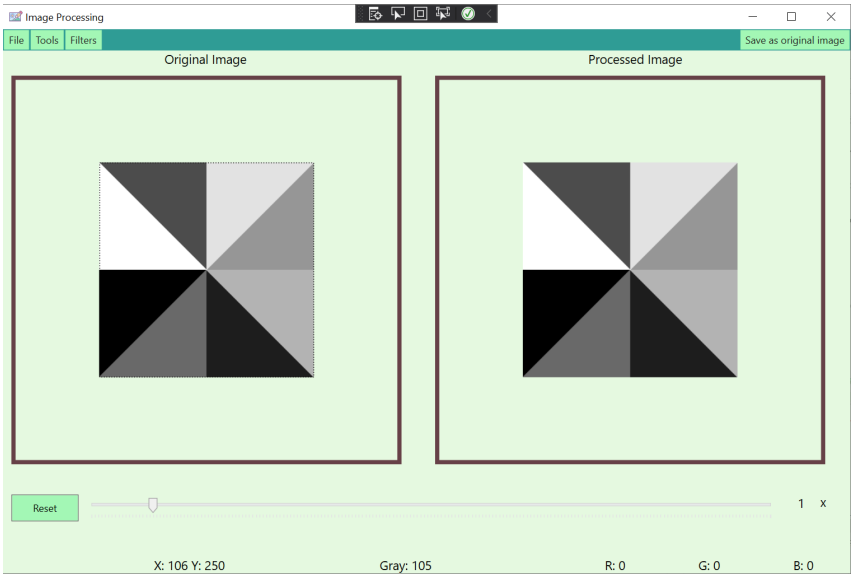
- Invert - imaginea procesată va avea valoarea de pe fiecare pixel egală cu 255 minus valoarea pixelului, respectiv din imaginea originală;



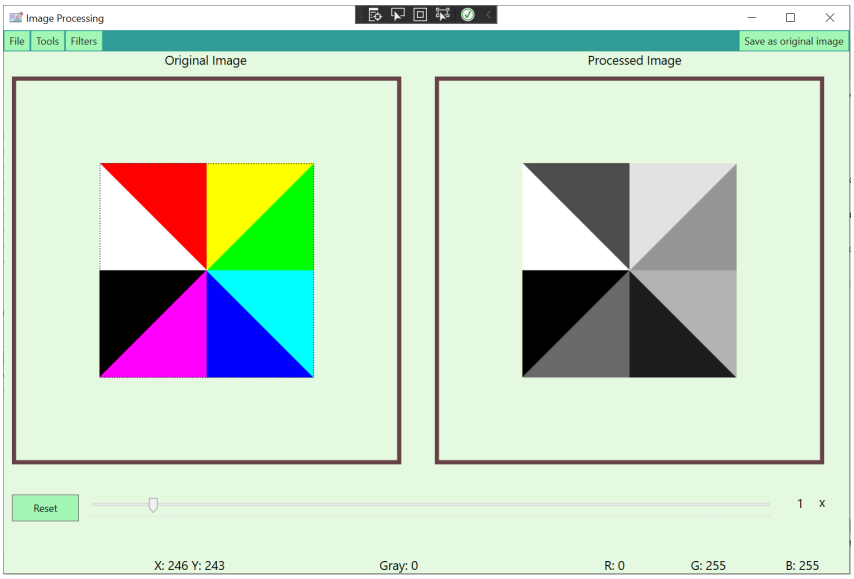


- Copy - copiază imaginea originală şi o plasează ca fiind imagine procesată;





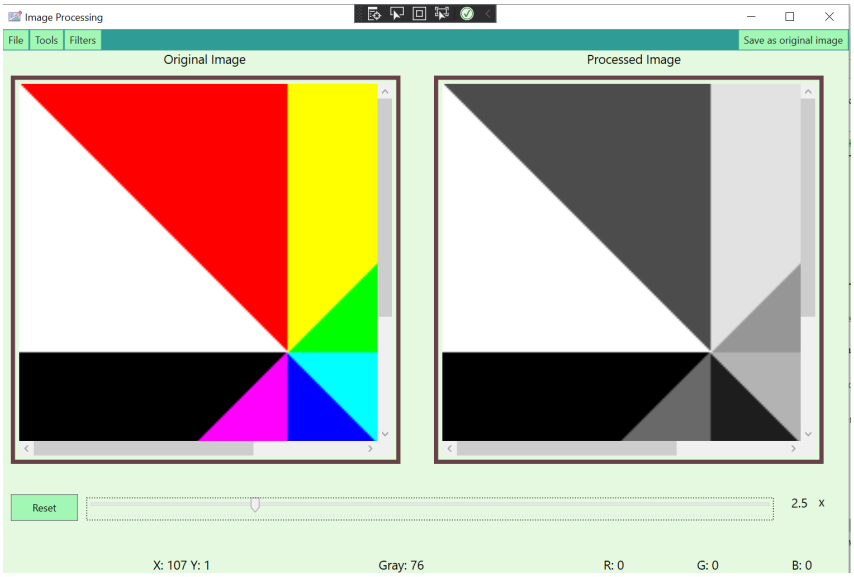
- Transform to gray image - această opţiune disponibilă doar pentru imaginea color, oferă posibilitatea de a transforma pixelii RGB în pixeli gri, pentru o prelucrare ulterioară a imaginii rezultate.



3. Filters - în această secţiune a meniului, utilizatorul va adăuga butoane cu care se vor putea aplica algoritmii implementaţi.

4. Save as original image - acest buton mută imaginea procesată în locul celei originale pentru a se putea aplica în continuare alte filtre.

5. Reset - acest buton setează dimensiunea imaginii la dimensiunea originală.



Capitolul 2

Folosirea framework-ului

2.1 Adăugarea unui algoritm

Pentru a putea adăuga un nou algoritm trebuie să parcurgem următoarele etape:

1. Trebuie să adăugăm o metodă ce conţine implementarea algoritmului în proiectul ImageProcessingAlgorithms, folder-ul Tools, clasa Tools. Metoda trebuie să fie publică şi statică pentru a putea fi accesată oricând şi oriunde în interiorul proiectului ImageProcessingFramework.

2. Trebuie să ne ducem în proiectul ImageProcessingFramework în fereastra MainWindow, fişierul MainWindow.xaml şi să căutăm meniul.

```
<MenuItem Header="File"
          Background="#A3F7B5">
    <MenuItem Header="Load grayscale image"
              Command="{Binding Path=AddGrayImage}"
              CommandParameter="{Binding ElementName=SliderZoom}"/>
    <MenuItem Header="Load color image"
              Command="{Binding Path=AddColorImage}"
              CommandParameter="{Binding ElementName=SliderZoom}"/>
    <Separator/>
    <MenuItem Header="Save processed image"
              Command="{Binding Path=Save}"/>
    <Separator/>
    <MenuItem Header="Exit"
              Command="{Binding Path=Exit}"/>
</MenuItem>

<MenuItem Header="Tools"
          Background="#A3F7B5">
    <MenuItem Header="Magnifier"
               Command="{Binding Path=Magnifier}"/>
    <MenuItem Header="GLevelsRow"
              Command="{Binding Path=RowDisplay}"/>
    <MenuItem Header="Invert"
              Command="{Binding Path=Invert}"/>
    <MenuItem Header="Copy"
              Command="{Binding Path=Copy}"/>
    <MenuItem Header="Transform to gray image"
              Command="{Binding Path=ConvertToGrayImage}" />
</MenuItem>

<MenuItem Header="Filters"
          Background="#A3F7B5">
</MenuItem>
<MenuItem Header="Save as original image"
          Background="#A3F7B5"
          HorizontalAlignment="Right"
          Command="{Binding Path=SaveAsOriginalImage}"/>
```

3. Acum că am găsit meniul aplicaţiei, este necesar să adăugăm un nou MenuItem în secţiunea în care dorim sau să adăugăm o nouă secţiune.

   Acest lucru se poate face cu ajutorul următoarei secvenţe de cod:

<MenuItem Header="nume_secţiune_sau_nume_algoritm"
Command="Binding Path=nume_comandă />

4. În momentul de faţă avem: implementarea algoritmului nostru şi butonul cu care vom porni algoritmul. Totuşi, lipseşte conexiunea dintre buton şi algoritm. Pentru a realiza această conexiune trebuie sa implementăm o comandă în clasa HomeCommands din folder-ul ViewModel.

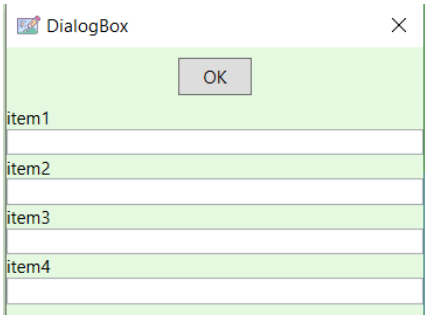
   Pentru a face acest lucru vom adăuga în clasa HomeCommands un membru privat *m_nume_comandă*, o metodă care va apela noua metodă din clasa Tools şi o proprietate publică de tipul ICommand cu numele *nume_comandă* ales la pasul anterior.

```
private ICommand m_nume_comanda;

1 reference
private void metoda_mea(object parameter)
{
    //apelarea metodei din clasa Tools in functie de tipul imaginii gray/color
}

0 references
public ICommand nume_comanda
{
    get
    {
        if (m_nume_comanda == null)
            m_nume_comanda = new RelayCommand(metoda_mea);
        return m_nume_comanda;
    }
}
```

După toti aceşti paşi algoritmul adăugat ar trebui să funcţioneze. Trebuie acordată o atenţie sporită atunci când implementăm *metoda_mea*, pentru a nu crea bug-uri atunci când avem un algoritm pentru imagini alb-negru, care nu funcţionează pentru imagini color sau invers. Aceste situaţii pot fi uşor tratate cu ajutorul variabilei de tip bool *m_isColorImage*, care va fi setată pe true atunci când avem o imagine color şi pe fals atunci când încărcăm imagini alb-negru.

2.2 Adăugare unei căsuţe de dialog

Pentru algoritmii realizaţi există posibilitatea să avem nevoie să folosim si parametrii. În acest sens, în proiectul ImageProcessingFramework am creat în folder-ul View, o altă fereastră numită DialogBox, folosită pentru a putea introduce unul sau mai mulţi parametrii ce pot fi folosiţi pentru algoritmi.



Pentru a putea folosi această fereastră se va crea în comanda pentru algoritm un obiect de tipul DialogBox, de unde se va putea extrage valoarea sau valorile introduse în căsuţele text. Imaginea de mai jos, ajută la înţelegerea modului de folosire a căsuţei de dialog. Textul introdus va reprezenta parametrul care va fi folosit ulterior pentru algoritm.

```
var dialogBox = new DialogBox();
var prop = new List<string>();
prop.Add("item1");
prop.Add("item2");
prop.Add("item3");
prop.Add("item4");

dialogBox.CreateDialogBox(prop);
if (dialogBox.ShowDialog() == true)
{
    //do something
}

var response = dialogBox.GetResponseTexts();
```

Clasa DialogBox conţine metoda *CreateDialogBox(List<string> texts)* care creează dinamic casuţele pentru fiecare parametru. Fiecare string va genera un TextBlock şi un TextBox, unde TextBlock-ul va reprezenta numele parametrului, iar TextBox-ul va fi parametrul. Pentru a putea lua valorile acestor parametrii am conceput metoda *GetResponseTexts()* care va întoarce un *List<string>*, reprezentând parametrii doriţi. Aceştia vor veni în ordinea în care a fost dată lista cu numele parametrilor.

În cazul în care în fereastră nu se văd foarte bine TextBox-urile şi TextBlock-urile se poate ajusta înălţimea elementelor în metoda *CreateDialogBox(List<string> texts)*, din clasa DialogBox.

2.3 Adăugarea unei ferestre de plotare a doi vectori

În cazul în care dorim să adăugăm o nouă fereastră în care desenăm un grafic cu doi vectori vom folosi clasa *GraphWindow*. Această clasă primeşte ca parametrii cele două liste de valori si le desenează. Prima listă de valori va fi desenată cu culoarea roşie, iar cea de-a doua va fi desenată cu culoarea albastru. Bineînţeles, acest lucru poate fi modificat la dorinţa utilizatorului.

Pentru a crea o fereastră de plotare pentru doi vectori vom realiza o comandă precum am folosit şi în cazul ferestrei de dialog. Un astfel de exemplu este prezentat în imaginea de mai jos:

```
List<double> firstList = new List<double> { -1, -2, -3, -4, -5, -6, -7 };
List<double> secondList = new List<double> { 7, 6, 5, 4, 3, 2, 1, 0 };

var graph = new GraphWindow(firstList, secondList);
graph.Show();
```

2.4 Folosirea clasei DrawHelper

Prin intermediul clasei *DrawHelper* venim în sprijinul utilizatorului şi îl ajutăm pe acesta să deseneze forme pe imagini. Astfel, avem următoarele metode:

1. *public static void SetPixelGray(Image imageSource, Image¡Gray, byte¿ inputImage, int x, int y, int gray)* - permite setarea unui pixel de culoare gri la o valoare dorită;

```
DrawHelper.SetPixelGray(InitialImageUi, GrayInitialImage, (int)MousePosition.X, (int)MousePosition.Y, 0);
DrawHelper.SetPixelGray(ProcessedImageUi, GrayProcessedImage, (int)MousePosition.X, (int)MousePosition.Y, 0);
```

2. *public static void SetPixelColor(Image imageSource, Image¡Bgr, byte¿ inputImage, int x, int y, int red, int green, int blue)* - permite setarea unui pixel de culoare la o valoare dorită;

```
DrawHelper.SetPixelColor(InitialImageUi, ColorInitialImage, (int)MousePosition.X, (int)MousePosition.Y, 0, 0, 0);
DrawHelper.SetPixelColor(ProcessedImageUi, ColorProcessedImage, (int)MousePosition.X, (int)MousePosition.Y, 0, 0, 0);
```

3. *public static Line DrawLine(Canvas canvas, int startX, int startY, int endX, int endY, int thickness, Brush color)* - permite desenarea unei linii de la punctul de start(*startX*, *startY*) la punctul de end(*endX*, *endY*);

```
InitialLine = DrawHelper.DrawLine(canvasOriginalImage, (int)LastPosition.X, (int)LastPosition.Y,
    (int)MousePosition.X, (int)MousePosition.Y, 2, Brushes.Red);
```

4. *public static Rectangle DrawRectangle(Canvas canvas, int leftTopX, int leftTopY, int rightBottomX, int rightBottomY, int thickness, Brush color)* - permite desenarea unui dreptunghi bazându-ne pe punctul stânga sus (*leftTopX*, *leftTopY*) şi punctul dreapta jos end(*rightBottomX*, *rightBottomY*);

```
InitialRectangle = DrawHelper.DrawRectangle(canvasOriginalImage, (int)LastPosition.X, (int)LastPosition.Y,
    (int)MousePosition.X, (int)MousePosition.Y, 2, Brushes.Red);
```

5. *public static Ellipse DrawElipse(Canvas canvas, int centerX, int centerY, int width, int height, int thickness, Brush color)* - permite desenarea unei eclipse de dimensiuni *width* si *height* de culoarea *color* cu centrul (*centerX*, *centerY*);

```
InitialEllipse = DrawHelper.DrawElipse(canvasOriginalImage, (int)MousePosition.X, (int)MousePosition.Y,
    40, 50, 2, Brushes.Red);
```

6. *public static Polygon DrawPolygon(Canvas canvas, PointCollection vectorOfPoints, int thickness, Brush color)* - permite desenarea unui poligon pe canvas-ul *canvas* de grosime *thickness* si culoarea *color* bazat pe click-urile utilizatorului;

```
InitialPolygon = DrawHelper.DrawPolygon(canvasOriginalImage, VectorOfMousePosition, 2, Brushes.Red);
```

VectorOfMousePosition este un vector care păstrează toate poziţiile selectate de mouse cu click stânga. Pentru a şterge una dintre formele de mai sus sau pentru a elibera valorile reţinute în vectorul de poziţii vom apăsa click dreapta.

Capitolul 3

Ce se întâmplă dacă întâlnim probleme legate de framework?

Pe parcursul lucrului în framework-ul prezentat, dacă se întâlnesc dificultăţi în adăugarea unui algoritm sau se produc diverse erori la compilare, este recomandat ca prim pas să se trimită un report pe repo, iar apoi să se trimită un mesaj la unul dintre autori, în care să se descrie situaţia dorită şi care este scenariul prin care s-a ajuns la problemă.