

Przetwarzanie obrazów w Javie, czyli OpenCV

Kacper Owczarek



OpenCV

Przebieg prezentacji

- Krótka teoria dotycząca przetwarzania obrazów
- Informacje na temat OpenCV
- Podstawy OpenCV
- Implementacja konwersji barw, podziału na kanały, progowania, binaryzacji, filtracji, detekcji krawędzi, operacji morfologicznych, transformacji, zmiany jasności oraz kontrastu przy użyciu OpenCV

Przetwarzanie obrazów

- Jedna z dziedzin przetwarzania sygnałów cyfrowych oraz grafiki komputerowej. Zajmuje się reprezentacją obrazu w postaci cyfrowej oraz komputerowymi algorytmami przetwarzania i akwizycji obrazów cyfrowych.

(Wikipedia)

- Stosowanie szeregu przekształceń zmieniających lub poprawiających ich jakość, podkreślających ich cechy składowe pod kątem lepszej obserwacji, automatycznej analizy i rozpoznawania oraz doprowadzenia do postaci wygodnej do pomiaru wybranych cech, kodowania czy transmisji.

(Digital Image Processing ~ Rafael C. Gonzalez)

Przetwarzanie obrazów - operacje

Przetwarzanie obrazów obejmuje, m.in. operacje:

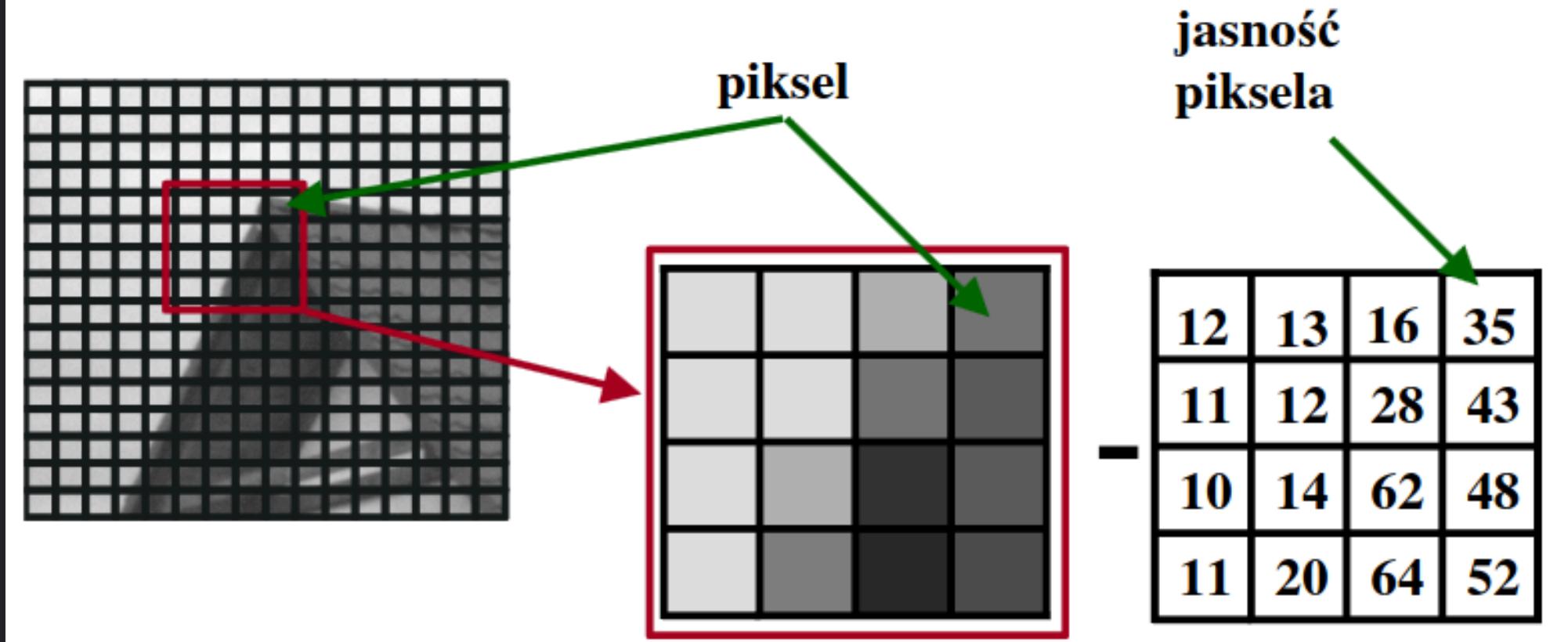
- filtracji,
- binaryzacji,
- segmentacji,
- transformacji geometrycznej,
- transformacji pomiędzy przestrzeniami barw (RGB, grayscale, binary) lub (RGB, CMYK itd.),
- wyrównania histogramu,
- morfologiczne (dylatacja, erozja, otwarcie, zamknięcie),
- arytmetyczne,
- kodowania,
- kompresji.

Przetwarzanie obrazów, a widzenie komputerowe

- **Przetwarzanie obrazu** polega na transformacji obrazu w obraz. Zarówno sygnałem wejściowym, jak i wyjściowym przetwarzania obrazu są obrazy.
- **Widzenie komputerowe** polega na tworzeniu jednoznacznych, znaczących opisów obiektów fizycznych na podstawie ich obrazu. Wynikiem wizji komputerowej jest opis lub interpretacja struktur w scenie 3D.
- **Widzenie komputerowe w znacznym stopniu pokrywa się z przetwarzaniem obrazu!**

Obraz cyfrowy

- Obraz definiuje się jako dwuwymiarową funkcję $F(x,y)$, gdzie x i y są współrzędnymi przestrzennymi, a amplituda F w dowolnej parze współrzędnych (x,y) nazywana jest intensywnością tego obrazu w tym punkt.
- Obraz można zdefiniować za pomocą dwuwymiarowej tablicy ułożonej w wiersze i kolumny.
- Obraz możemy przedstawić w postaci macierzy.



OpenCV

- wieloplatformowa
- dostępna od 2000 roku
- Open Source (Apache License 2)
- zaprojektowana przez Intel
- biblioteka stworzona w C++, ale dostępna w C#, Python, Java
- obecna wersja: 4.9.0
- używana przez: Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota



Możliwości OpenCV

- Odczytywanie i zapisywanie obrazów.
- Przechwytywanie i zapisywanie filmów.
- **Przetwarzanie obrazów (filtrowanie, przekształcanie).**
- Wykonywanie wykrywania cech.
- Wykrywanie określone obiektów, takich jak twarze, oczy, samochody (w filmach lub obrazach).
- Analizowanie wideo, czyli wykrywanie w nim ruchu, odejmowania tła i śledzenie znajdujących się w nim obiektów.

Moduły w OpenCV

Nazwa pakietu	Moduł
org.opencv.core	Podstawowa funkcjonalność (Core Functionality)
org.opencv.imgproc	Przetwarzanie obrazu (Image Processing)
org.opencv.video	Obsługa wideo (Video)
org.opencv.videoio	Obsługa I/O wideo (Video I/O)
org.opencv.calib3d	calib3d
org.opencv.features2d	features2d
org.opencv.objdetect	Detekcja obiektów (Object detection)
*org.opencv.highgui	Podstawowe UI (Simple UI)

* od wersji 3.0 część funkcji w **org.opencv.imgcodecs** i **org.opencv.videoio**

Instalacja biblioteki OpenCV

1. Instalacja "natywna" biblioteki:

EXE -> OpenCV/build/java/opencv-.jar -> Project Structure (Modules/Dependencies) -> Edit -> OpenCV/build/java/x64/opencv_java*.dll*

2. DIY, czyli skompiluj to sam

3. Dodanie zależności do odpowiedniego pliku (Maven, Gradle)

```
<dependency>
    <groupId>org.openpnp</groupId>
    <artifactId>opencv</artifactId>
    <version>4.9.0</version>
</dependency>
```

Inicjalizacja biblioteki OpenCV

```
//W static{} lub main {}
System.loadLibrary(Core.NATIVE_LIBRARY_NAME); //Ładowanie biblioteki natywnej

nu.pattern.OpenCV.loadLocally(); //Ładowanie z lokalnie dostępnych plików biblioteki

nu.pattern.OpenCV.loadShared(); //Ładowanie z zewnętrznego źródła
                                //(użycie natywnych bibliotek) (Java < 12)
```

Przechowywanie obrazów

```
Mat mat = new Mat();
Mat mat = new Mat(int wiersze, int kolumny, int typ); //typ ->
//CV_<bity_głębi>{Unsigned|Signed|Float}C(<liczba_kanałów>), CV_8UC1
Mat mat = new Mat(int wiersze, int kolumny, Scalar skalar);
Mat mat = new Mat(int wiersze, int kolumny, int typ, Scalar skalar);
Mat mat = new Mat(Size rozmiar, int typ);
Mat mat = new Mat(Size rozmiar, int typ, Scalar skalar);
Mat mat = new Mat.ones(Size rozmiar, int typ);

Mat row0 = mat.row(0);
Mat col0 = mat.col(0);
Mat rows_number = mat.rows();
Mat cols_number = mat.cols();
```

Ładowanie obrazów z pliku

```
//Ładowanie biblioteki OpenCV
OpenCV.loadLocally();

//Nowy obiekt pakietu org.opencv.imgcodecs
Imgcodecs imgCodecs = new Imgcodecs();

//Uzyskanie ścieżki do pliku (zamiast Main.class może być getClass())
String path = Main.class.getResource("/pingu.jpg").getPath();

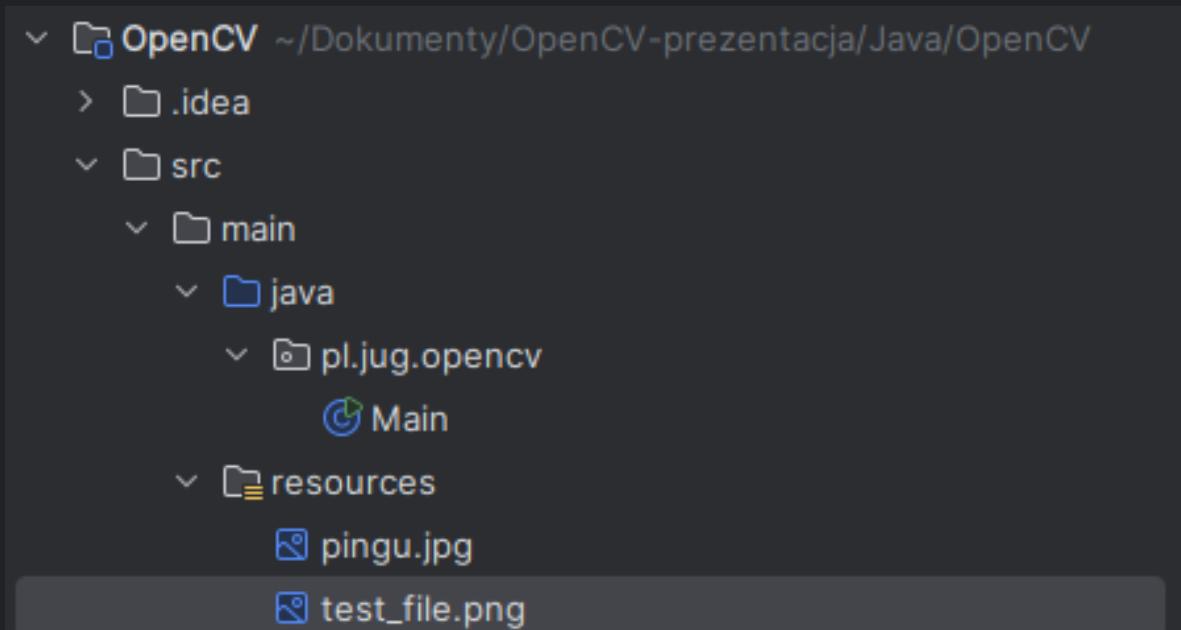
//Ładowanie pliku z zasobów
//Dostępne flagi: IMREAD_COLOR, IMREAD_GRAYSCALE, IMREAD_LOAD_GDAL, IMREAD_ANYCOLOR,
//IMREAD_REDUCED_COLOR_2, IMREAD_REDUCED_COLOR_4, IMREAD_REDUCED_COLOR_8,
//IMREAD_REDUCED_GRAYSCALE_2, IMREAD_REDUCED_GRAYSCALE_4, IMREAD_REDUCED_GRAYSCALE_8,
//IMREAD_UNCHANGED
Mat img = imgCodecs.imread(path, Imgcodecs.IMREAD_COLOR);

System.out.println(img);
```

```
Mat [ 580*718*CV_8UC3, isCont=true, isSubmat=false, nativeObj=0x7f77a01e8660, dataAddr=0x7f77742ce040 ]
```

Zapisywanie obrazów do pliku

```
//Nazwa pliku do zapisu  
String fileName = "src/main/resources/test_file.png";  
  
//Zapisanie obrazu do pliku  
imgCodecs.imwrite(fileName, img);
```



Integracja ze Swing

```
OpenCV.loadLocally();
Imgcodecs imgCodecs = new Imgcodecs();
String path = Main.class.getResource("/pingu.jpg").getPath();
Mat img = imgCodecs.imread(path, Imgcodecs.IMREAD_COLOR);

//Utworzenie macierzy bajtów
MatOfByte matOfByte = new MatOfByte();

//Konwersja macierzy na macierz bajtów
//imencode(ext, image, matOfByte);
imgCodecs.imencode(".jpg", img, matOfByte);

//Konwersja na tablicę bajtów
byte[] byteArray = matOfByte.toArray();

//Przygotowanie obiektu InputStream i BufferedImage
InputStream in = new ByteArrayInputStream(byteArray);
BufferedImage bufImage = ImageIO.read(in);

//Zainicjalizowanie JFrame i ustawienie w nim zawartości
JFrame frame = new JFrame();
frame.getContentPane().add(new JLabel(new ImageIcon(bufImage)));
frame.pack();
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



Integracja z JavaFX

```
WritableImage writableImage = SwingFXUtils.toFXImage(bufImage, null);

//Utworzenie nowego widoku, ustawienie pozycji obrazu, wielkości widoku obrazu
//i współczynnika zachowania widoku obrazu
ImageView imageView = new ImageView(writableImage);
imageView.setX(50);
imageView.setY(25);
imageView.setFitHeight(600);
imageView.setFitWidth(500);
imageView.setPreserveRatio(true);

//Utworzenie obiektu grupy i dodanie do niego widoku obrazu
Group root = new Group(imageView);

//Utworzenie obiektu sceny z obiektem grupy
Scene scene = new Scene(root, 600, 500);

//Ustawienie tytułu okna, dodanie do niego sceny i wyświetlenie okna
stage.setTitle("Testowy obraz");
stage.setScene(scene);
stage.show();
```

Testowy obraz



Konwersja przestrzeni barw

```
//Utworzenie obiektu dla pakietu Imgproc  
Imgproc imgProc = new Imgproc();  
  
//Utworzenie nowej, pustej macierzy  
Mat tmp = new Mat();  
  
//Konwersja obrazu z przestrzeni barw BGR na przestrzeń szarości  
//cvtColor(Mat src, Mat dst, int code)  
//src - macierz obrazu źródłowego  
//dst - macierz, do której ma zostać przetworzony obraz  
//code - wartość liczbową, reprezentująca typ konwersji  
imgProc.cvtColor(img, tmp, Imgproc.COLOR_BGR2GRAY);
```



Podział przestrzeni barw na kanały

```
//Utworzenie macierzy dla poszczególnych składowych
Mat b = new Mat();
Mat g = new Mat();
Mat r = new Mat();

//Podział obrazu wejściowego na poszczególne kanały
//(wraz z zerowaniem wartości innych kanałów)
//NIEBIESKI
ArrayList<Mat> bChannel = new ArrayList<>(3);
Core.split(img, bChannel); //Core.split(Mat src, ArrayList<Mat> dst)
bChannel.set(1, Mat.zeros(bChannel.get(1).size(), CV_8UC1));
bChannel.set(2, Mat.zeros(bChannel.get(2).size(), CV_8UC1));
Core.merge(bChannel, b); //Core.merge(ArrayList<Mat> src, Mat dst)

//ZIELONY
ArrayList<Mat> gChannel = new ArrayList<>(3);
Core.split(img, gChannel); //Core.split(Mat src, ArrayList<Mat> dst)
gChannel.set(0, Mat.zeros(gChannel.get(0).size(), CV_8UC1));
gChannel.set(2, Mat.zeros(gChannel.get(2).size(), CV_8UC1));
Core.merge(gChannel, g); //Core.merge(ArrayList<Mat> src, Mat dst)

//CZERWONY
ArrayList<Mat> rChannel = new ArrayList<>(3);
Core.split(img, rChannel); //Core.split(Mat src, ArrayList<Mat> dst)
rChannel.set(0, Mat.zeros(rChannel.get(0).size(), CV_8UC1));
rChannel.set(1, Mat.zeros(rChannel.get(1).size(), CV_8UC1));
Core.merge(rChannel, r); //Core.merge(ArrayList<Mat> src, Mat dst)
```



Jasność i kontrast

```
double alpha = 1.0; //Współczynnik korekcji kontrastu
int beta = 0; //Współczynnik korekcji jasności (dla wszystkich kanałów)

//Utworzenie nowej, pustej macierzy o rozmiarze macierzy obrazu
Mat out = Mat.zeros(img.size(), img.type());

//Pobranie testowych wartości dla współczynników
try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("* Podaj wartość wskaźnika alfa [0.0-3.0]: ");
    alpha = scanner.nextDouble();
    System.out.print("* Podaj wartość wskaźnika beta [0-100]: ");
    beta = scanner.nextInt();
}

//Zmiana kontrastu i jasności (forma skrócona)
//convertTo(Mat dst, -1, alpha, beta);
img.convertTo(out, -1, alpha, beta);
```

```
//Zmiana kontrastu i jasności (forma rozwinięta)
//Pobranie danych o obrazie wejściowym w postaci tablicy bajtów
byte[] imgData = new byte[(int) (img.total()*img.channels())];
img.get(0, 0, imgData);

//Utworzenie tablicy bajtów dla nowego, przetworzonego obrazu
byte[] outData = new byte[(int) (out.total()*out.channels())];

//Obliczenie nowych wartości po zmianie kontrastu i koloru dla każdego piksela
//(w każdym jego kanale)
for (int y = 0; y < img.rows(); y++) {
    for (int x = 0; x < img.cols(); x++) {
        for (int c = 0; c < img.channels(); c++) {
            //Określenie piksela
            double pixel = imgData[(y * img.cols() + x) * img.channels() + c];

            //Odczytanie odpowiedniej wartości danego piksela
            pixel = pixel < 0 ? pixel + 256 : pixel;

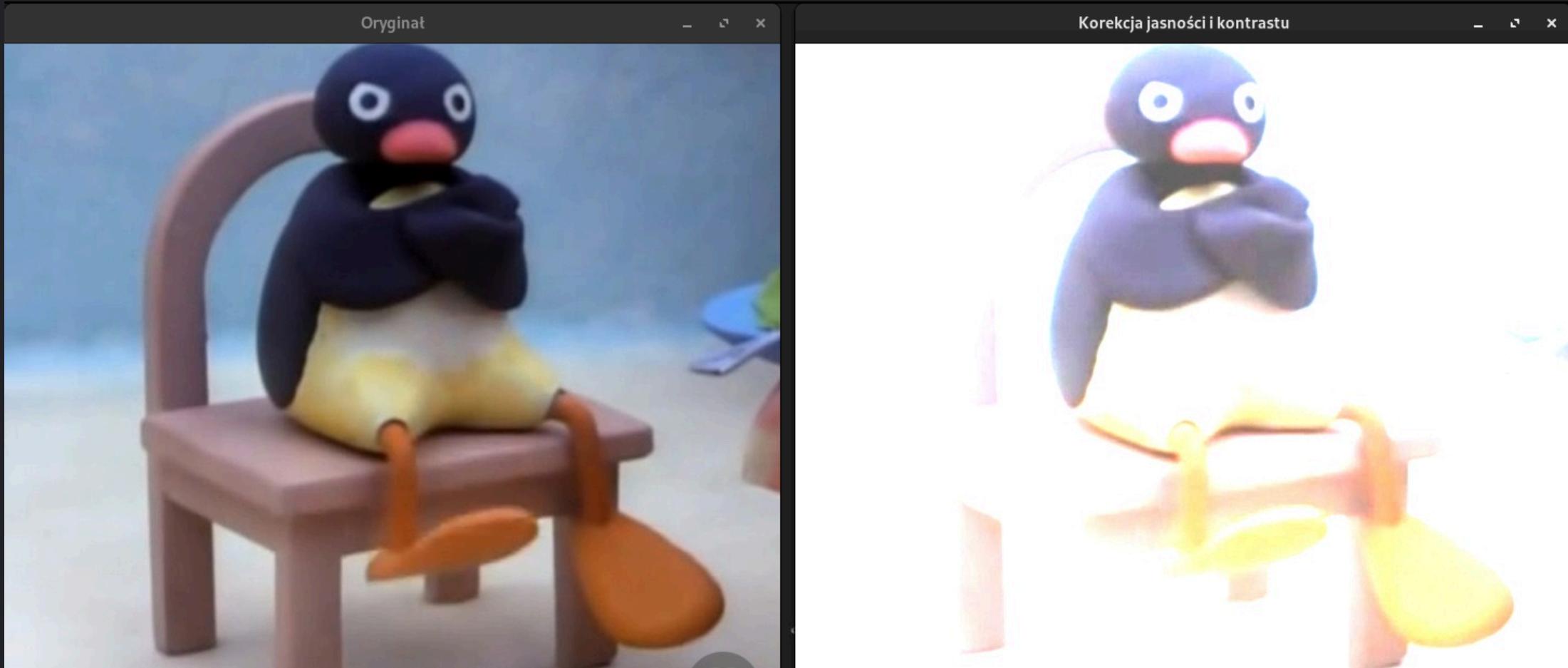
            //Obliczenie nowej wartości dla piksela
            int newVal = (int) Math.round(alpha * pixel + beta);
            newVal = newVal > 255 ? 255 : (Math.max(newVal, 0));

            //Przypisanie danemu pikselowi nowej wartości
            outData[(y * img.cols() + x) * img.channels() + c] = (byte) newVal;

            //C++
            //out.at<cv::Vec3b>(y,x)[c] = cv::saturate_cast<uchar>(alpha*(img.at<cv::Vec3b>(y,x)[c]) + beta);
        }
    }
}

//Umieszczenie tablicy bajtów w macierzy nowego obrazu
out.put(0, 0, outData);
```

Dla $\alpha=2.0$ i $\beta=100$



Progowanie

```
int thresh_type = 0; //Typ progowania
//(0-BINARY, 1-BINARY INV., 2-TRUNCATE, 3-TO ZERO, 4-TO ZERO INV. ...)
int thresh_R = 0;    //Próg dla kanału R
int thresh_G = 0;    //Próg dla kanału G
int thresh_B = 0;    //Próg dla kanału B

try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- PROGOWANIE ---");
    System.out.print("\n* Podaj typ progowania: ");
    thresh_type = scanner.nextInt();
    System.out.print("\n* Podaj wartość progu dla R: ");
    thresh_R = scanner.nextInt();
    System.out.print("\n* Podaj wartość progu dla G: ");
    thresh_G = scanner.nextInt();
    System.out.print("\n* Podaj wartość progu dla B: ");
    thresh_B = scanner.nextInt();
}

ArrayList<Mat> img_channels = new ArrayList<>(3); //Przechowywanie kanałów RGB obrazu wejściowego
ArrayList<Mat> out_channels = new ArrayList<>(3); //Przechowywanie kanałów RGB obrazu wyjściowego

//Dla całego obrazu
//threshold(Mat src, Mat dst, int thresh, int maxval, int type)
//imgProc.threshold(img, out, 100, 255, Imgproc.THRESH_BINARY);

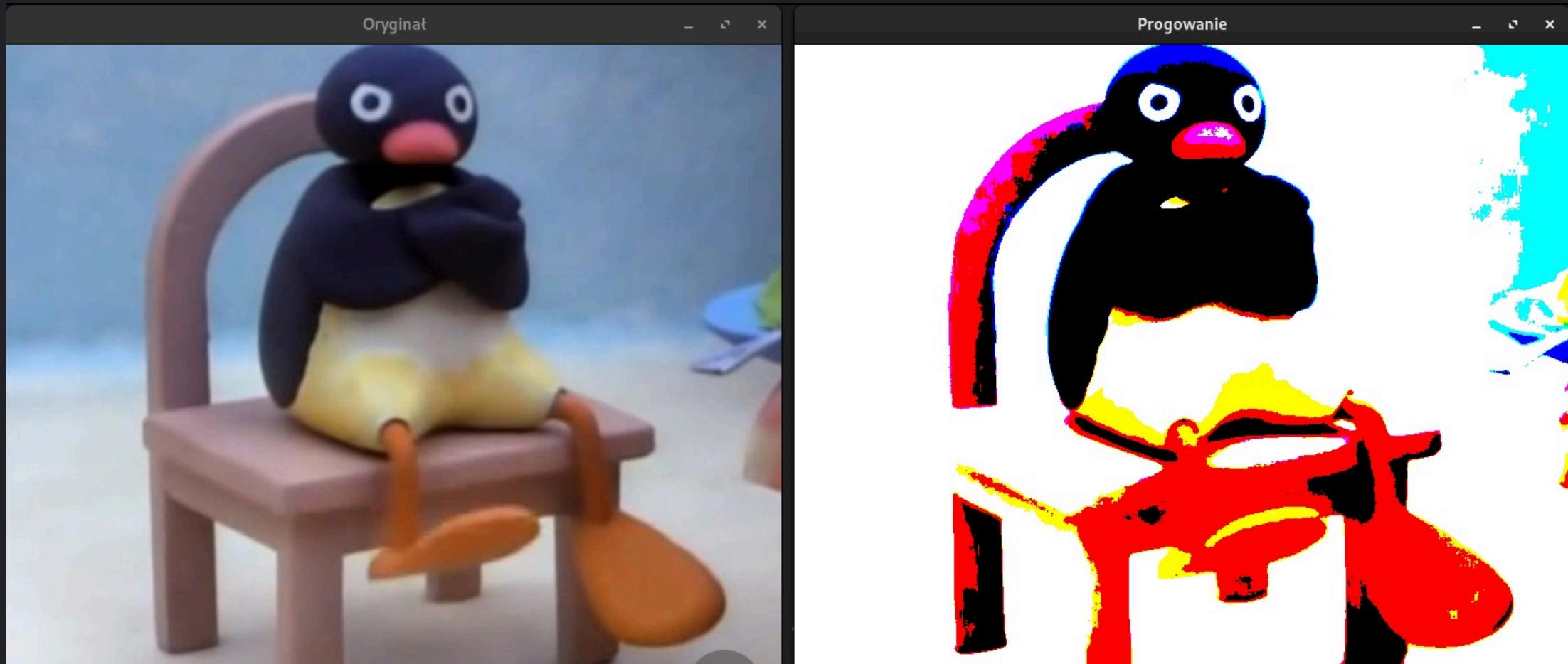
Core.split(img, img_channels); //Podzielenie obrazu wejściowego na kanały RGB

out_channels.add(Mat.zeros(img_channels.get(0).size(), img_channels.get(0).type()));
out_channels.add(Mat.zeros(img_channels.get(1).size(), img_channels.get(1).type()));
out_channels.add(Mat.zeros(img_channels.get(2).size(), img_channels.get(2).type()));

//Ustawienie progowania dla poszczególnych kanałów
imgProc.threshold(img_channels.get(0), out_channels.get(0), thresh_B, 255, thresh_type);
imgProc.threshold(img_channels.get(1), out_channels.get(1), thresh_G, 255, thresh_type);
imgProc.threshold(img_channels.get(2), out_channels.get(2), thresh_R, 255, thresh_type);

//Połączenie zmienionych składowych RGB do jednego obiektu (obrazu wyjściowego)
Core.merge(out_channels, out);
```

Dla typu progowania **BINARY** (0), **thresh_R=100**,
thresh_G=100, **thresh_B=100**



Progowanie adaptacyjne

```
int thresh_type = 0;           //Typ progowania
int adaptive_method = 0;       //Typ metody adaptacyjnego progowania
int block_size = 0;            //Rozmiar bloku sąsiedztwa
double constant = 0;           //Stała

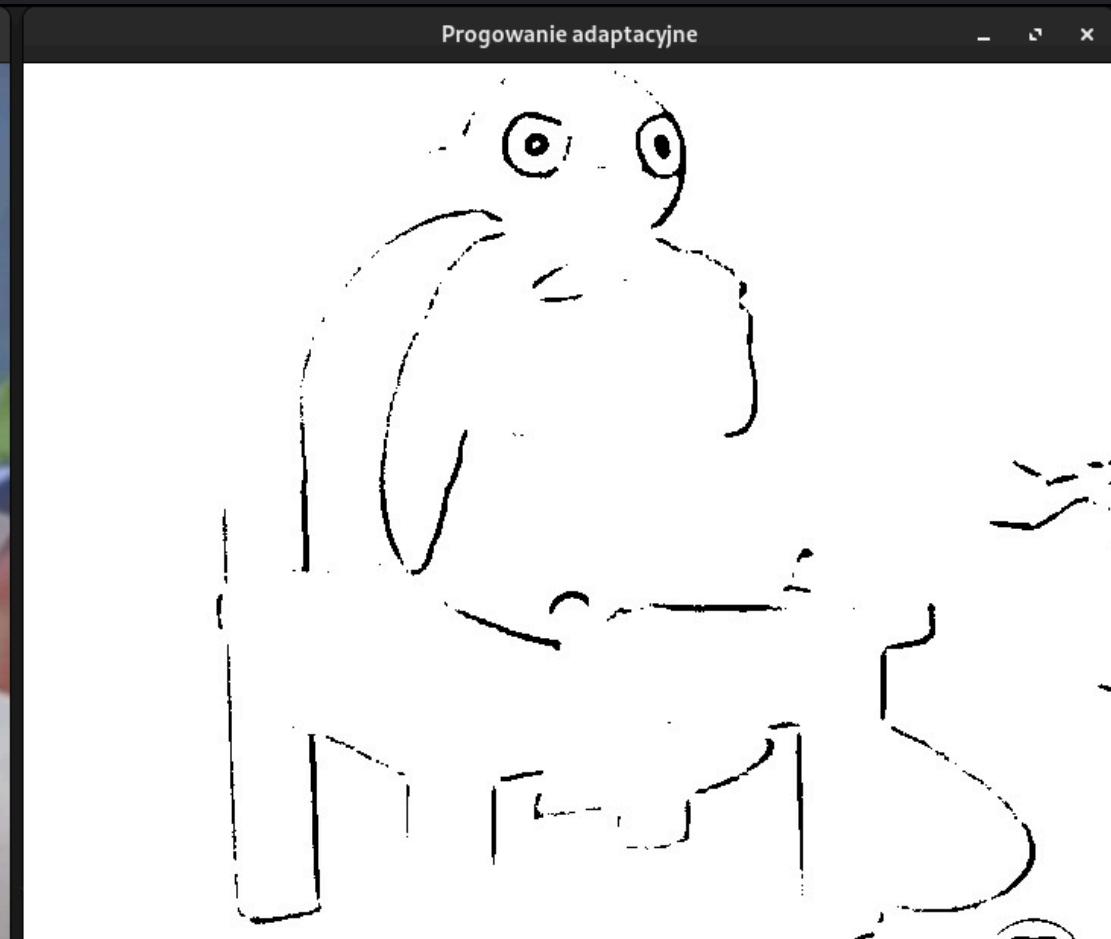
Mat img = new Mat();

imgProc.cvtColor(in, img, Imgproc.COLOR_BGR2GRAY);

try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- PROGOWANIE ADAPTACYJNE ---");
    System.out.print("\n* Podaj typ progowania: ");
    thresh_type = scanner.nextInt();
    System.out.print("* Podaj metodę adaptacyjnego progowania: ");
    adaptive_method = scanner.nextInt();
    System.out.print("* Podaj rozmiar bloku: ");
    block_size = scanner.nextInt();
    System.out.print("* Podaj wartość stałej do obliczeń: ");
    constant = scanner.nextDouble();
}

//adaptiveThreshold(Mat src, Mat dst, int maxval, int adaptiveMethod, int type, int blocksize, double c)
//0-ADAPTIVE_THRESH_MEAN_C, 1-ADAPTIVE_THRESH_GAUSSIAN_C
imgProc.adaptiveThreshold(img, out, 255, adaptive_method, thresh_type, block_size, constant);
```

Dla typu progowania **BINARY** (0), typu metody **MEAN**,
block_size=11, constant=15



Binaryzacja

```
int thresh_type = 0;           //Typ progowania
int bin_Y = 0;                 //Wartość progu binaryzacji

Mat img = new Mat();

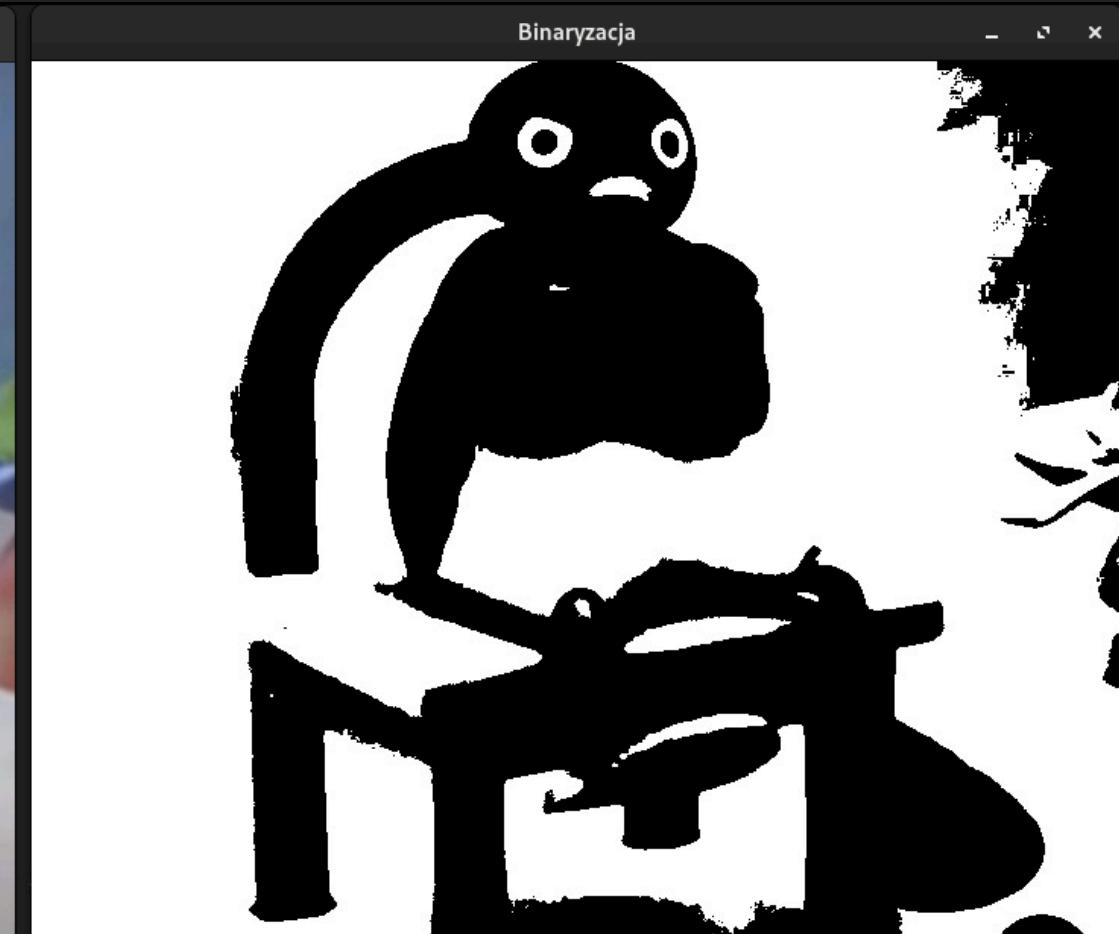
//Konwersja na YCrCb
imgProc.cvtColor(in, img, Imgproc.COLOR_BGR2YCrCb);

try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- BINARYZACJA ---");
    System.out.print("\n* Podaj typ binaryzacji: ");
    thresh_type = scanner.nextInt();
    System.out.print("* Podaj wartość progu binaryzacji: ");
    bin_Y = scanner.nextInt();
}

//Kolekcja pomocnicza do przechowywania kanałów YCrCb
ArrayList<Mat> imgChannels = new ArrayList<Mat>(3);
Core.split(img, imgChannels);

//Dokonanie operacji binaryzacji na kanale Y
imgProc.threshold(imgChannels.get(0), out, bin_Y, 255, thresh_type);
```

Dla typu binaryzacji **BINARY (0)** oraz **bin_Y=126**



Transformacje - rotacja

```
Size size = img.size();    //Rozmiar

System.out.print("\n--- ROTACJA ---");
System.out.print("\n* Podaj kąt obrotu: ");
angle = scanner.nextDouble();

//Przypisanie obiekowi rozmiaru szerokości i wysokości
size.width = img.cols();
size.height = img.rows();

//Wyliczenie punktu obrotu
Point rotationPoint = new Point(size.width/2., size.height/2.);
Mat rotationMatrix = imgProc.getRotationMatrix2D(rotationPoint, angle, 1.0);

//Wywołanie funkcji obrotu
//warpAffine(Mat src, Mat dst, Mat matrix, Size size);
imgProc.warpAffine(img, out, rotationMatrix, size);
```

Transformacje - skalowanie

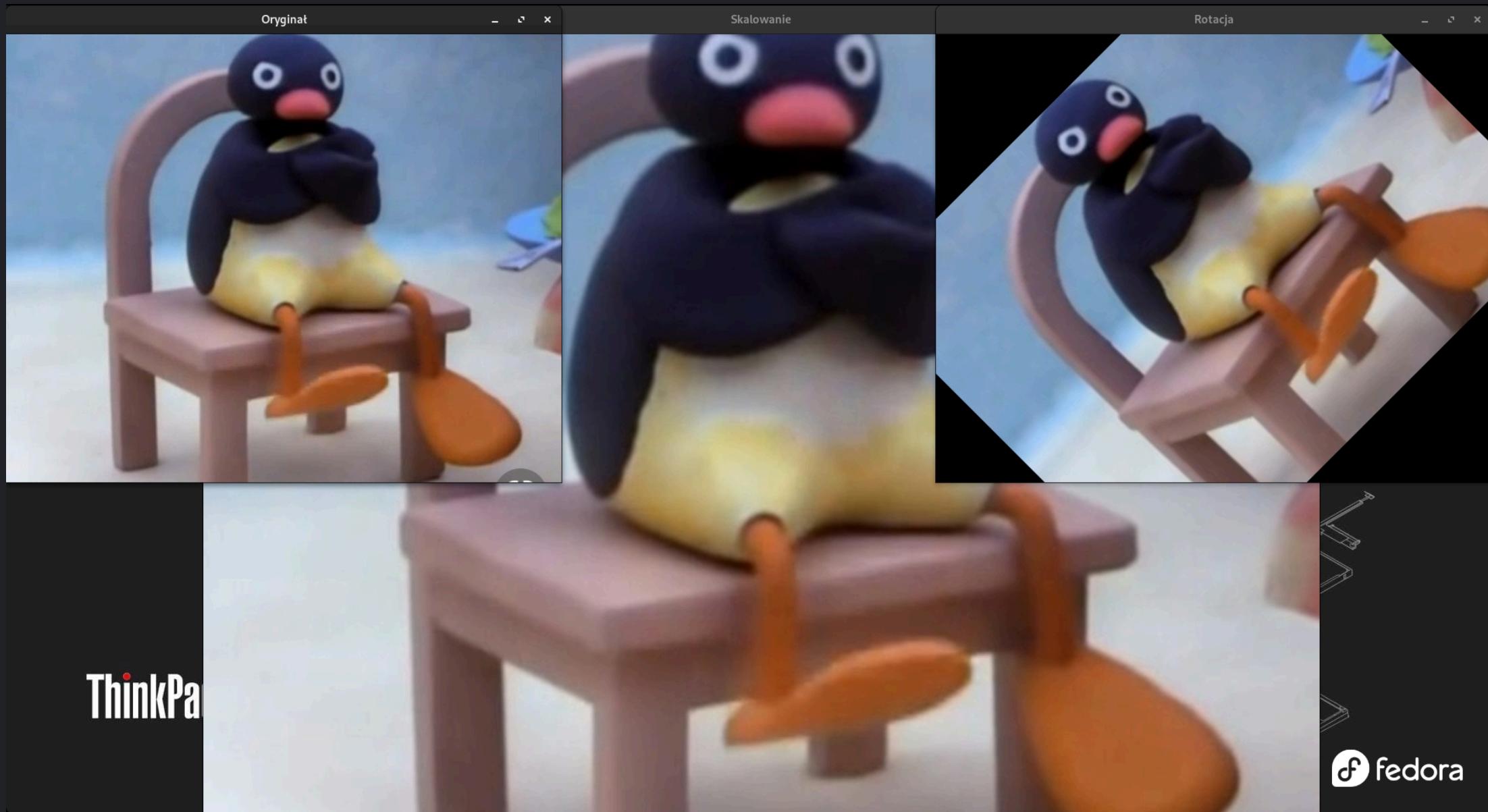
```
System.out.print("\n--- SKALOWANIE ---");
System.out.print("\n* Podaj procent skalowania: ");
scale = scanner.nextDouble();

//Obliczenie poprawnego współczynnika skali (wraz z zabezpieczeniem przed 0)
scale = (scale+1)/100;

Size size=new Size(img.cols()*scale, img.rows()*scale); //Rozmiar

//Wywołanie funkcji skalowania
//resize(Mat src, Mat dst, Size dsize, double fx, double fy, int interpolation));
imgProc.resize(img, out, size, 0, 0, INTER_LINEAR);
```

Dla kąta=45° i skali=200%



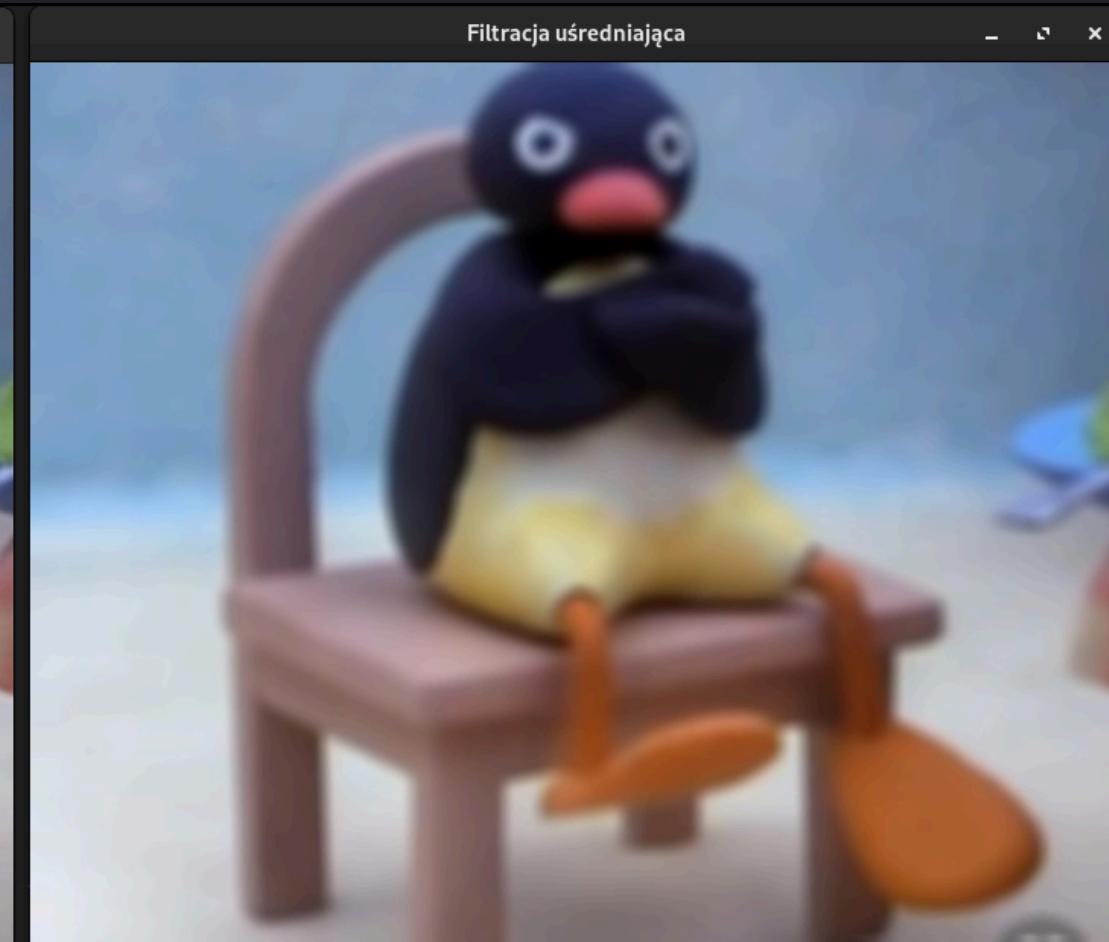
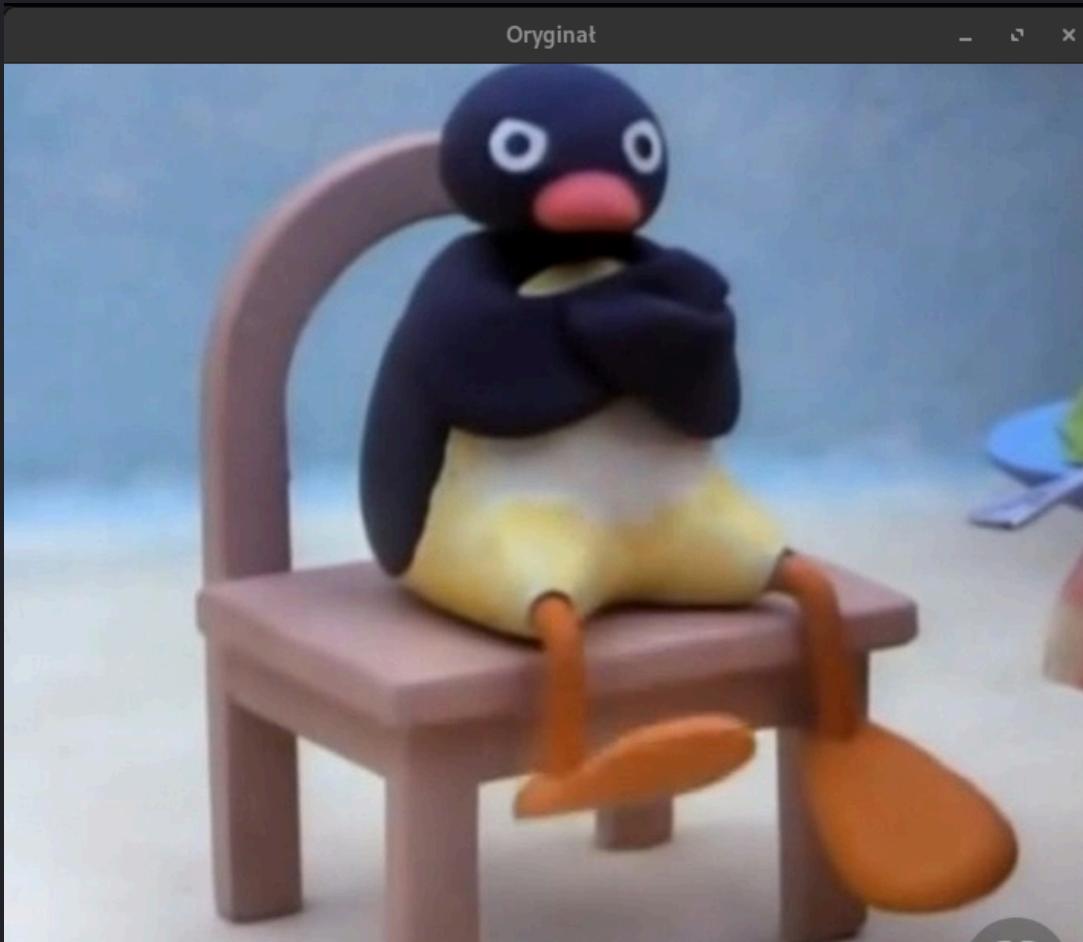
Filtracja (na przykładzie uśredniającej)

```
try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- FILTRACJA---");
    System.out.print("\n* Podaj wielkość okna [1-10]: ");
    windowSize = scanner.nextInt();
}

int filterMask=(2*windowSize)+1;      //Maska filtra (nieparzysta)

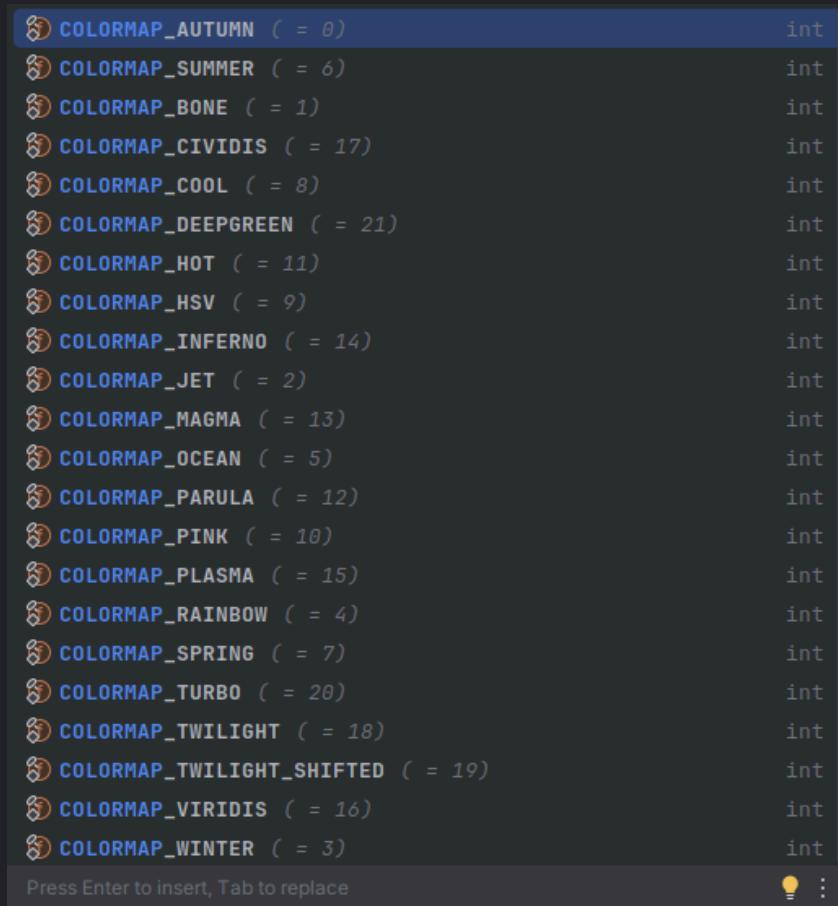
//Wykonanie filtracji uśredniającej
//blur(Mat src, Mat dst, Size kernel);
//GaussianBlur(Mat src, Mat dst, Size kernel, sigmaX, sigmaY);
//medianBlur(Mat src, Mat dst, int filterSize)
imgProc.blur(img, out, new Size(filterMask, filterMask));
```

Dla rozmiaru okna=5

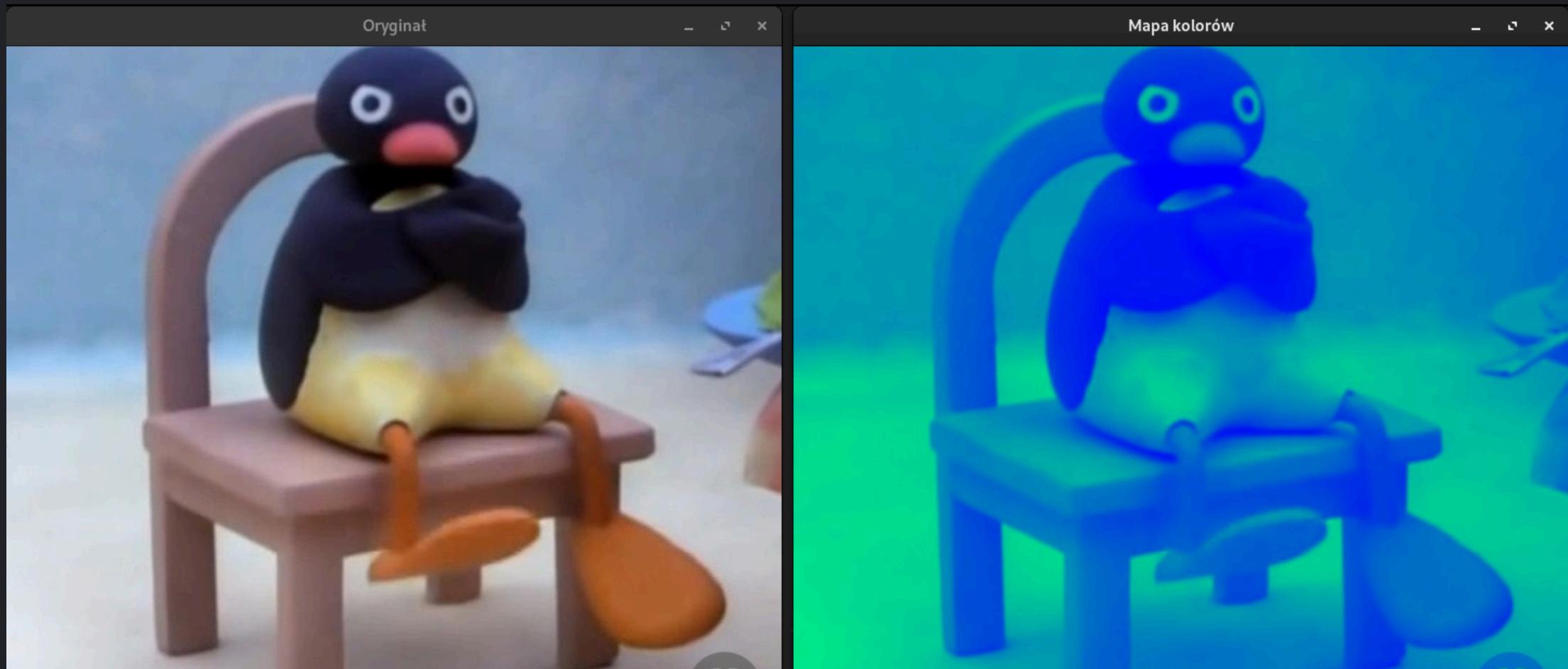


Mapa kolorów

```
//Zastosowanie mapy kolorów na obraz wejściowy  
//applyColorMap(Mat src, Mat dst, int colorMap)  
Imgproc.applyColorMap(img, out, Imgproc.COLORMAP_WINTER);
```



Imgproc.COLORMAP_WINTER



Detekcja krawędzi metodą Canny'ego

```
Mat img = new Mat();
Imgproc.cvtColor(in, img, Imgproc.COLOR_BGR2YCrCb);

try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- DETEKCJA KRAWĘDZI ---");
    System.out.print("\n* Podaj próg minimalny [1-100]: ");
    T_min = scanner.nextInt();
}

//Kolekcja pomocnicza do przechowywania kanałów YCrCb
ArrayList<Mat> imgChannels = new ArrayList<>(3);
Core.split(img, imgChannels); //Podzielenie obrazu wejściowego YCrCb na oddzielne kanały

//Obliczenie wartości maksymalnego progu T_max
T_max = T_min*3;

//Wywołanie filtracji Canny'ego
//Canny(Mat src, Mat edges, double threshold1, double threshold2, int apertureSize)
imgProc.Canny(imgChannels.getFirst(), out, T_min, T_max, 3); //Filtracja Canny'ego
```

Dla $T_{min}=35$



Operacje morfologiczne - erozja

```
try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- EROZJA ---");
    System.out.print("\n* Podaj typ elementu [0-2]: ");
    elementType = scanner.nextInt();
    System.out.print("\n* Podaj wielkość bazową elementu [0-11]: ");
    baseSize = scanner.nextInt();
}

Mat kernel = new Mat();      //Utworzenie obiektu dla kernela

//Sprawdzenie, jaki rodzaj elementu strukturalnego został wybrany i przypisanie go do obiektu kernel
if(elementType== 0)
    //imgProc.getStructuringElement(int shape, Size kernelSize, Point point);
    kernel = imgProc.getStructuringElement(Imgproc.MORPH_RECT, new Size(2*baseSize + 1, 2*baseSize+1), new Point(baseSize, baseSize));
else if(elementType == 1)
    kernel = imgProc.getStructuringElement(Imgproc.MORPH_CROSS, new Size(2*baseSize + 1, 2*baseSize+1), new Point(baseSize, baseSize));
else if(elementType == 2)
    kernel = imgProc.getStructuringElement(Imgproc.MORPH_ELLIPSE, new Size(2*baseSize + 1, 2*baseSize+1), new Point(baseSize, baseSize));

//Funkcja do wykonania erozji
//erode(Mat src, Mat dst, Mat kernel, Point point, int iterations, int borderType);
imgProc.erode(img, out, kernel, new Point(-1,-1), 1, Core.BORDER_CONSTANT);
```

Dla typu **MORPH_CROSS** i wielkości bazowej=9



Operacje morfologiczne - dylatacja

```
try (Scanner scanner = new Scanner(System.in)) {
    System.out.print("\n--- DYLATACJA ---");
    System.out.print("\n* Podaj typ elementu [0-2]: ");
    elementType = scanner.nextInt();
    System.out.print("\n* Podaj wielkość bazową elementu [0-11]: ");
    baseSize = scanner.nextInt();
}

Mat kernel = new Mat();      //Utworzenie obiektu dla kernela

//Sprawdzenie, jaki rodzaj elementu strukturalnego został wybrany i przypisanie go do obiektu kernel
if(elementType== 0)
    //imgProc.getStructuringElement(int shape, Size kernelSize, Point point);
    kernel = imgProc.getStructuringElement(Imgproc.MORPH_RECT, new Size(2*baseSize + 1, 2*baseSize+1), new Point(baseSize, baseSize));
else if(elementType == 1)
    kernel = imgProc.getStructuringElement(Imgproc.MORPH_CROSS, new Size(2*baseSize + 1, 2*baseSize+1), new Point(baseSize, baseSize));
else if(elementType == 2)
    kernel = imgProc.getStructuringElement(Imgproc.MORPH_ELLIPSE, new Size(2*baseSize + 1, 2*baseSize+1), new Point(baseSize, baseSize));

//Funkcja do wykonania dylatacji
//dilate(Mat src, Mat dst, Mat kernel, Point point, int iterations, int borderType);
imgProc.dilate(img, out, kernel, new Point(-1,-1), 1, Core.BORDER_CONSTANT);
```

Dla typu **MORPH_CROSS** i wielkości bazowej=9



Inne operacje przetwarzania obrazów w OpenCV

- wykorzystanie innych rodzajów filtracji (np. Box filter),
- wykorzystanie innych operacji morfologiczne,
- rysowanie po obrazie (np. dopisywanie tekstu, rysowanie prostokątów),
- wykorzystanie innych operatorów (np. operator Scharra),
- wykorzystanie innych metod wykrywania krawędzi (operator Sobela, Prewitta),
- wykorzystanie transformacji Laplace,
- detekcja twarzy na obrazie,
- wyrównanie histogramu,
- wykorzystanie transformacji linii Hougha,
- wykorzystanie innych operacji transformacji geometrycznych,
- obsługa kamery, itd...

Gdzie znaleźć więcej informacji?

- Dokumentacja OpenCV dla Javy:
<https://docs.opencv.org/4.9.0/javadoc/>
- Poradnik OpenCV dla Javy:
<https://www.tutorialspoint.com/opencv/index.htm>
- Poradnik OpenCV dla Javy (pod wersję 3.x):
<https://opencv-java-tutorials.readthedocs.io/en/latest/01-installing-opencv-for-java.html>
- ChatGPT
- StackOverflow
- Google

Dziękuję za uwagę!

- Github: <https://github.com/Chareq1>
- Linkedin: <https://www.linkedin.com/in/chareq1/>
- Prezentacja + pliki: https://github.com/Chareq1/JUG_OpenCV