

Przetwarzanie obrazów w Javie, czyli OpenCV

Kacper Owczarek



OpenCV

Przebieg prezentacji

- 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

OpenCV

- wieloplatformowa,
otwartoźródłowa
- dostępna od 2000 roku
- zaprojektowana przez Intel
- biblioteka stworzona w
języku C++, ale dostępna w
C#, Python, Java, Mathlab
- używana przez:
Google, 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.

Najważniejsze moduły w OpenCV

Nazwa pakietu	Moduł
<code>org.opencv.core</code>	Podstawowa funkcjonalność (Core Functionality)
<code>org.opencv.imgproc</code>	Przetwarzanie obrazu (Image Processing)
<code>org.opencv.video</code>	Obsługa wideo (Video)
<code>org.opencv.calib3d</code>	calib3d
<code>org.opencv.features2d</code>	features2d
<code>org.opencv.objdetect</code>	Detekcja obiektów (Object detection)
<code>*org.opencv.highgui</code>	<code>org.opencv.videoio</code>
	Obsługa wejścia/wyjścia dla obrazów

** od wersji 3.0 zaprzestano wsparcia tego pakietu (przez Qt)*

Instalacja biblioteki OpenCV

1. Instalacja "natywna" biblioteki
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();
```

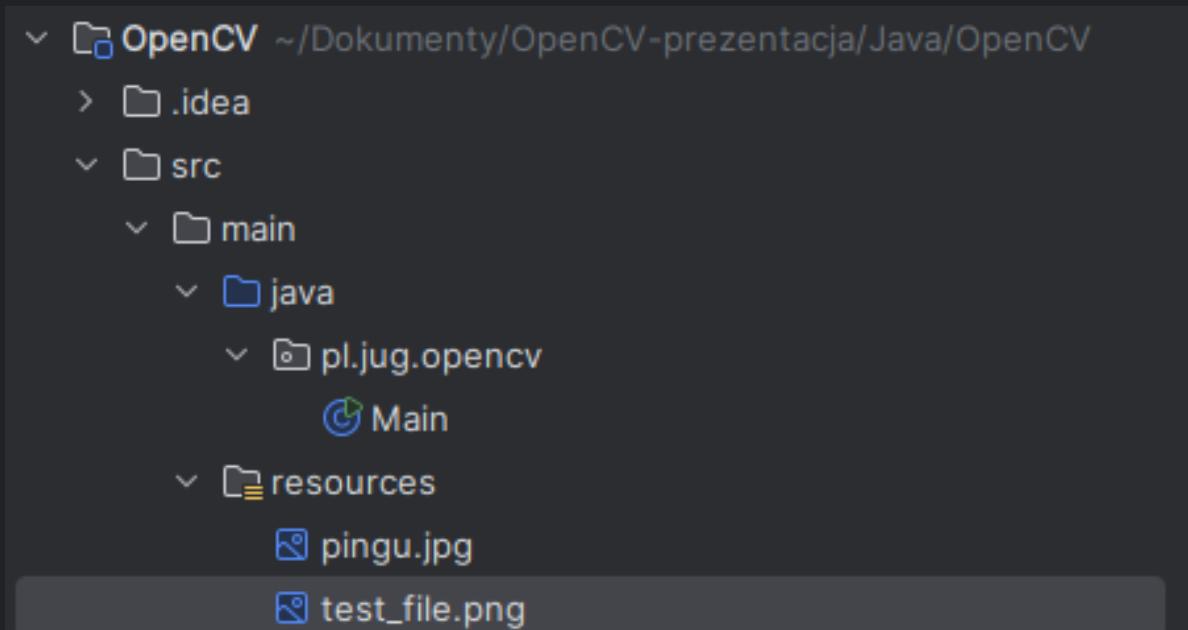
Ładowanie obrazów z pliku

```
//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

```
//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 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 liczbowa, 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)

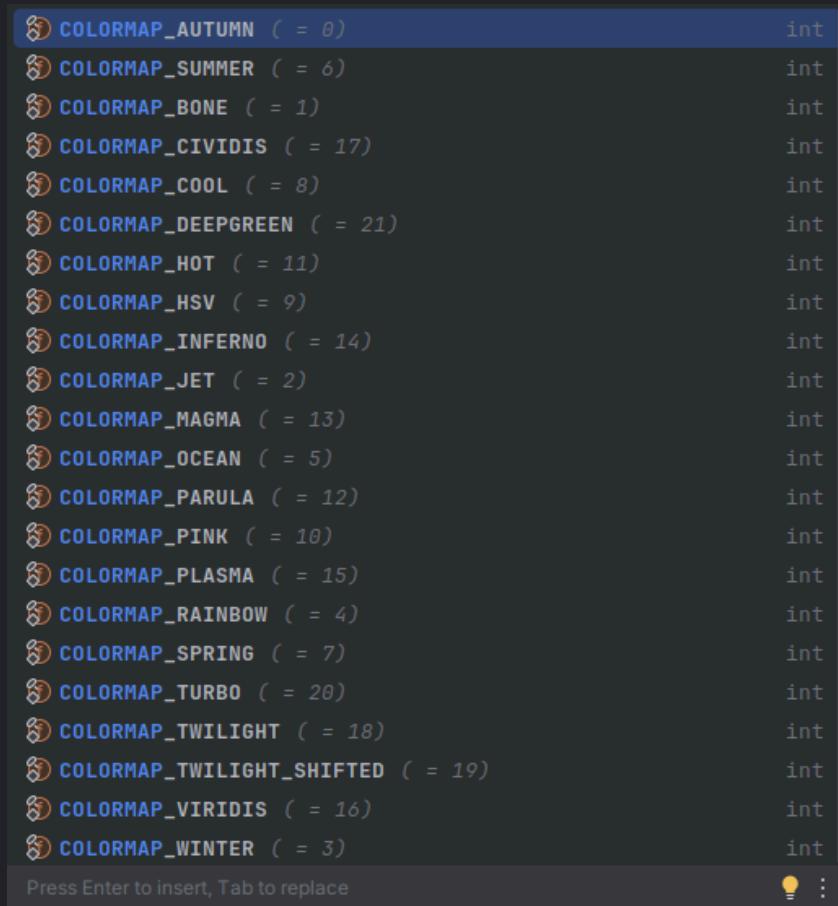
//ZIELONY
ArrayList<Mat> gChannel = new ArrayList<>(3);
Core.split(img, gChannel); //Core.split(Mat src, ArrayList<Mat> dst)
gChannel.set(0, Mat.zeros(gChannel.get(0).size(), CV_8UC1));
gChannel.set(2, Mat.zeros(gChannel.get(2).size(), CV_8UC1));
Core.merge(gChannel, g); //Core.merge(ArrayList<Mat> src,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

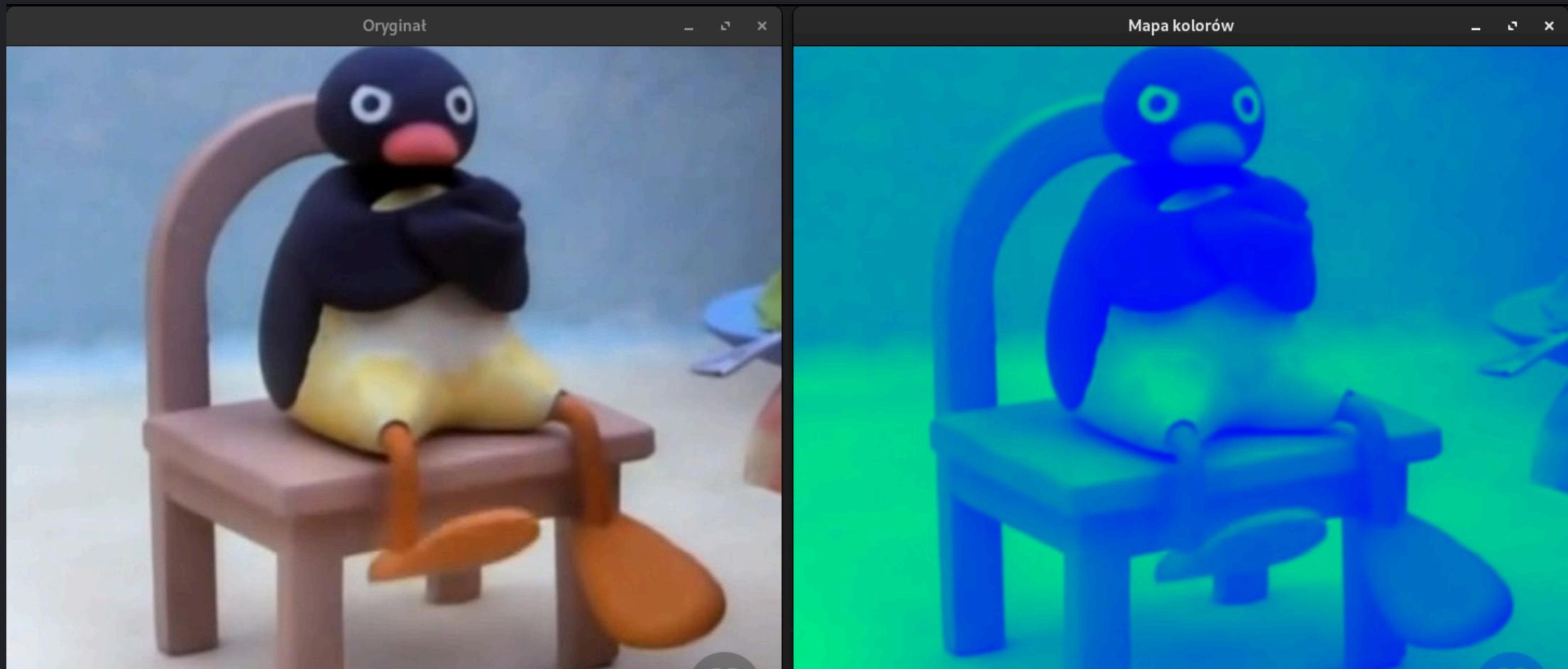


Mapa kolorów

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



Imgproc.COLORMAP_WINTER



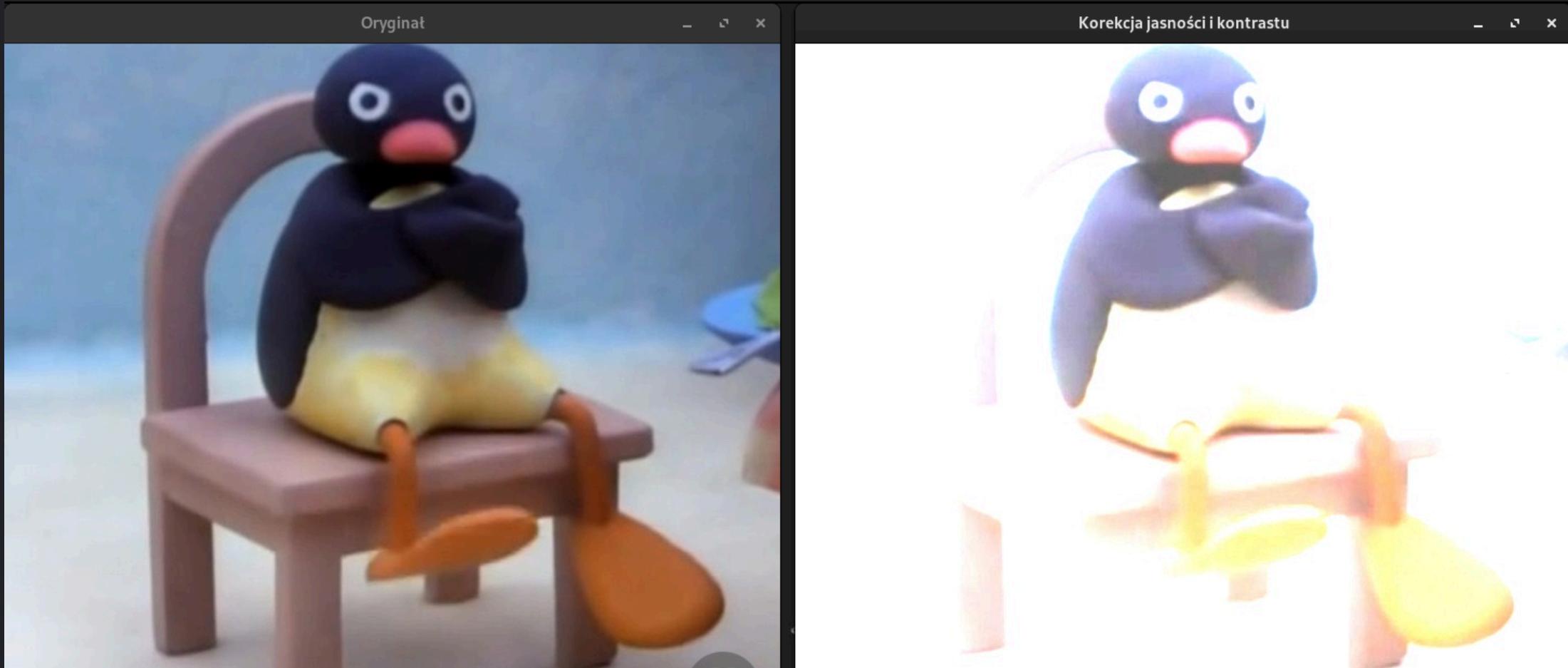
Jasność i kontrast

```
double alpha = 2.0; //Współczynnik korekcji kontrastu (min.0)
int beta = 100; //Współczynnik korekcji jasności (dla wszystkich kanałów) (min. 0)

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

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

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

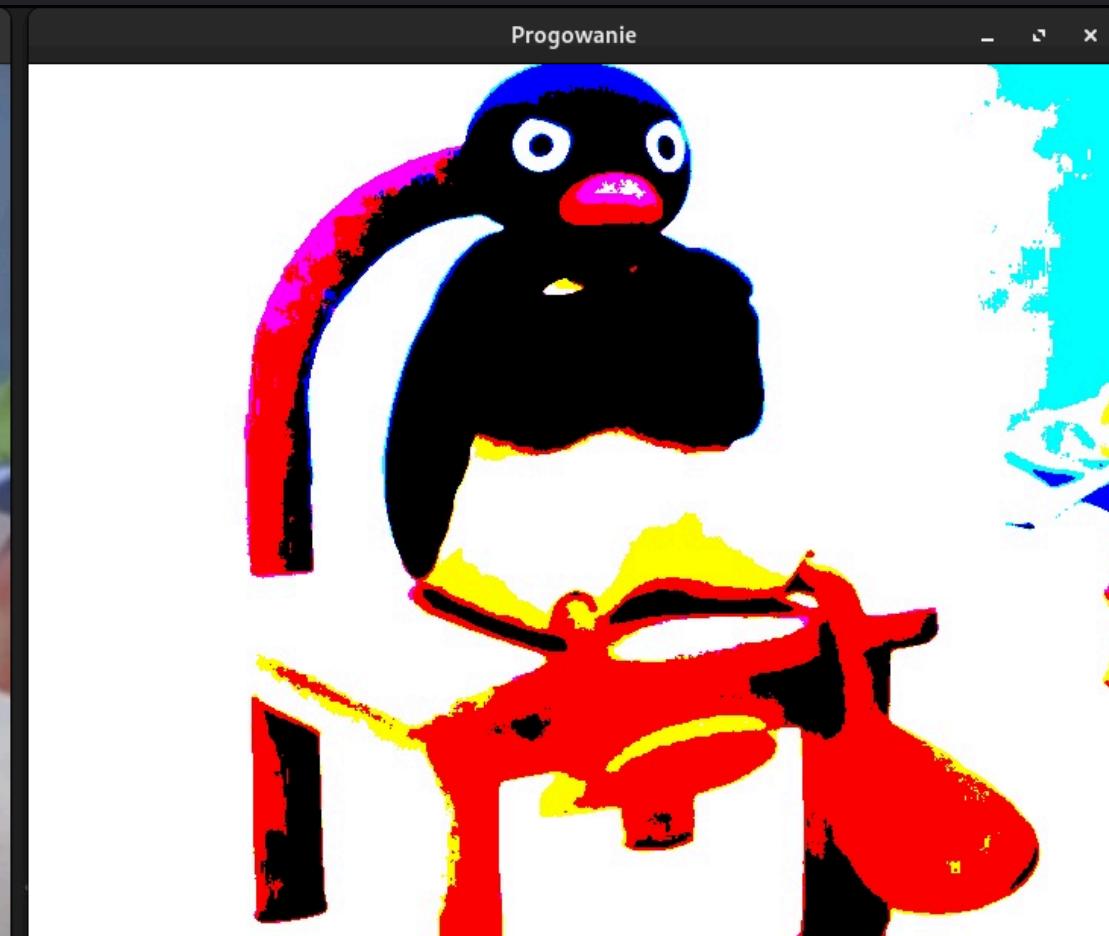
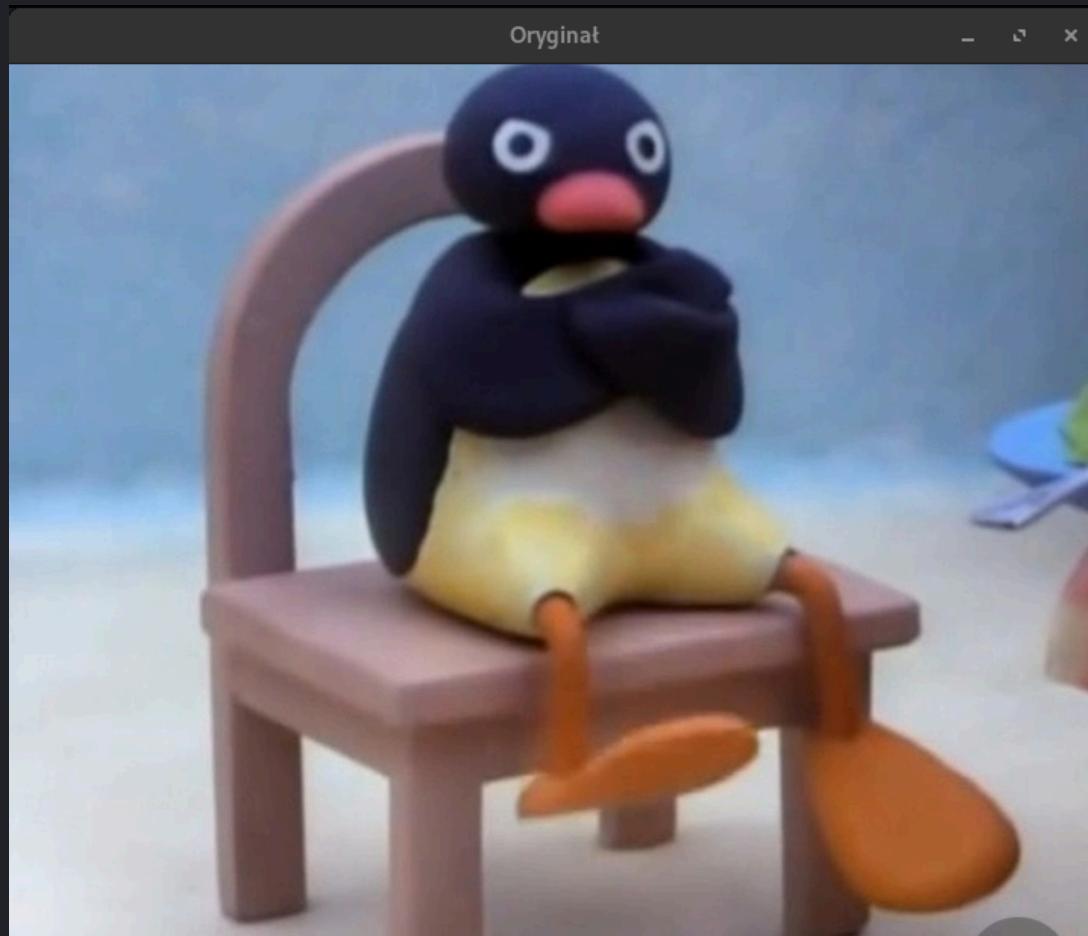


Progowanie

```
int thresh_type = 0; //Typ progowania
//(0-BINARY, 1-BINARY INV., 2-TRUNCATE, 3-TO ZERO, 4-TO ZERO INV. ...)
int thresh = 100;

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

Dla typu progowania **BINARY (0)**, $\text{thresh}=100$



Binaryzacja

```
int thresh_type = 0;           //Typ progowania
int bin_Y = 126;              //Wartość progu binaryzacji

Mat img = new Mat();

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

//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**



Progowanie adaptacyjne

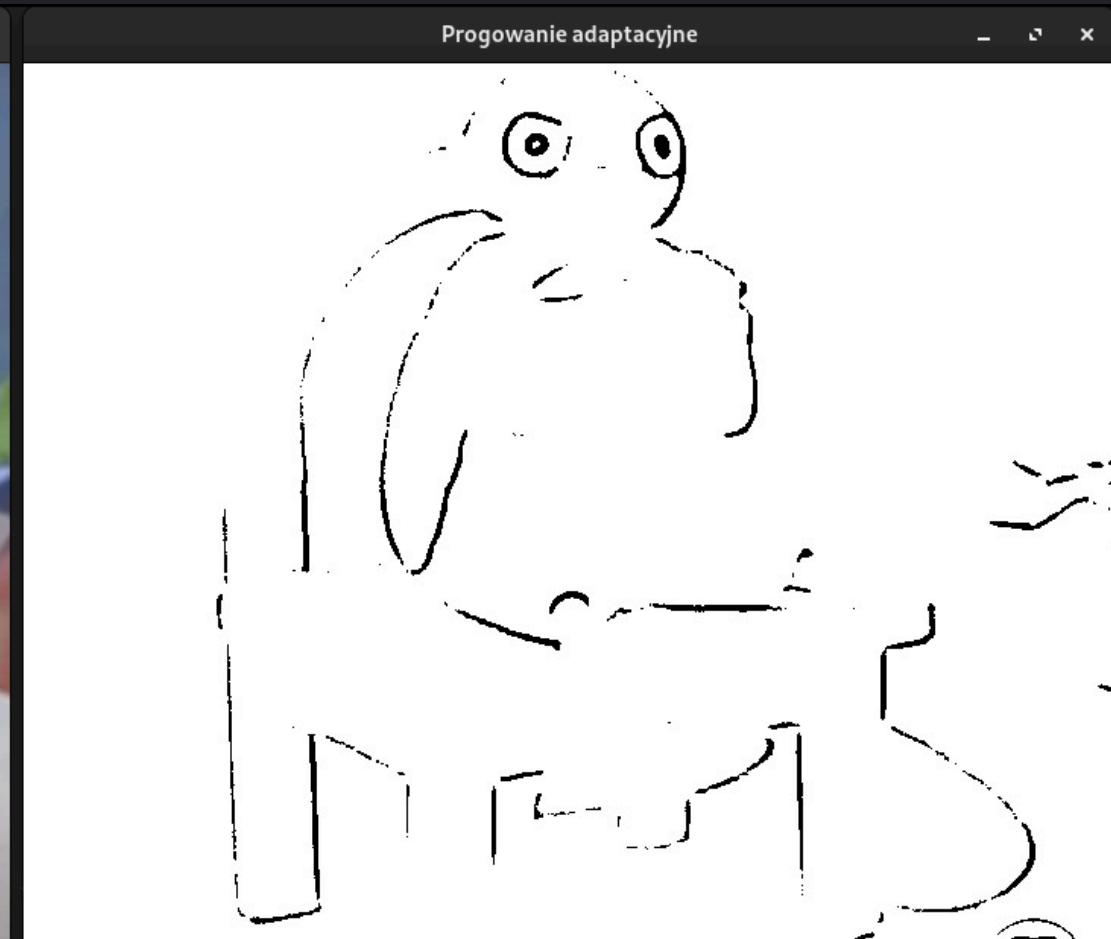
```
int thresh_type = 0;          //Typ progowania
int adaptive_method = 0;      //Typ metody adaptacyjnego progowania
int block_size = 11;          //Rozmiar bloku sąsiedztwa
double constant = 15;         //Stała

Mat img = new Mat();

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

//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



Transformacje - rotacja

```
double angle = 45;
Size size = img.size();    //Rozmiar

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

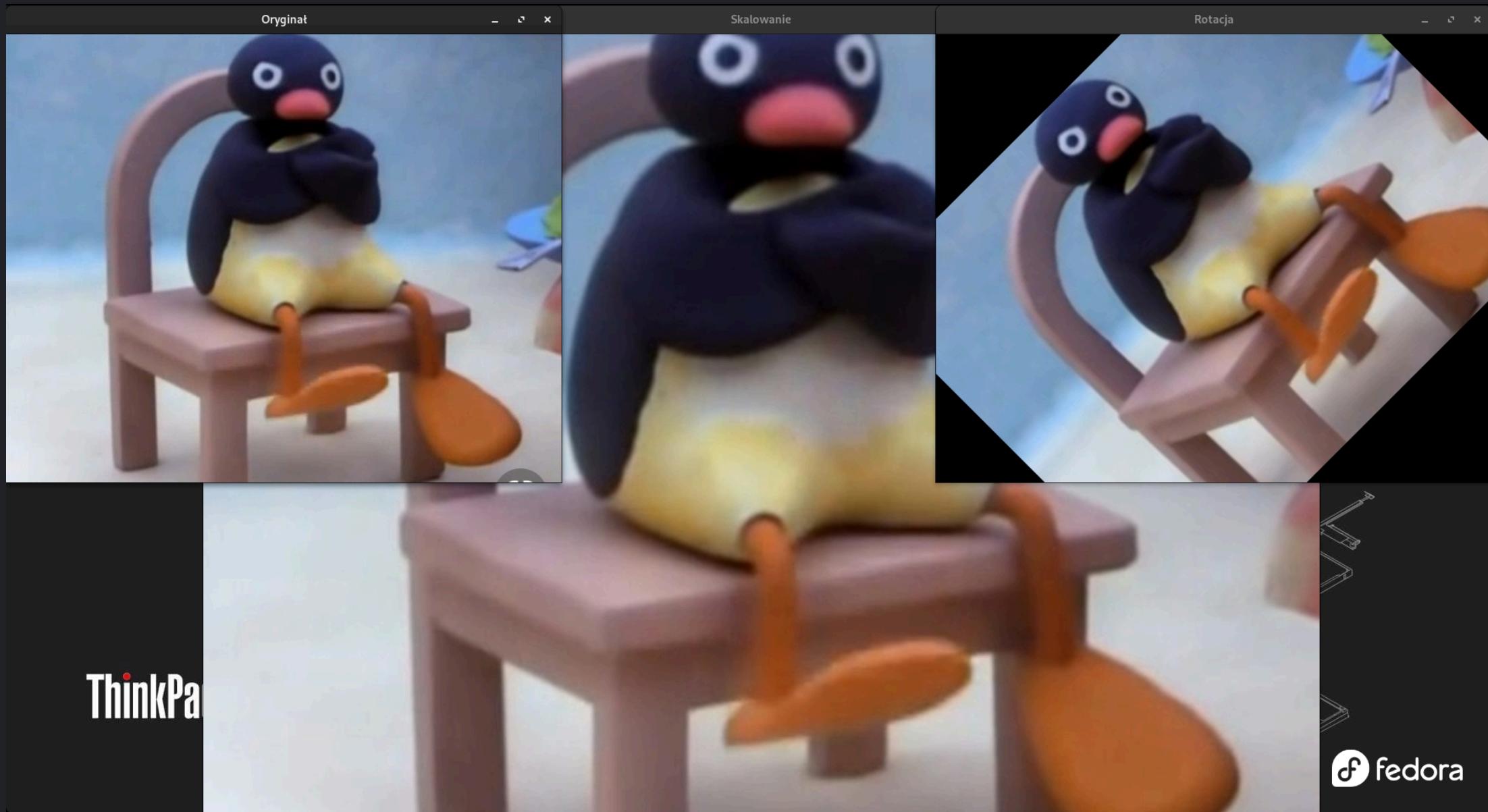
//Wyliczenie punktu obrotu
Point rotationPoint = new Point(size.width/2., size.height/2.);
Mat rotationMatrix = imgProc.getRotationMatrix2D(rotationPoint, angle, 1.0);

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

Transformacje - skalowanie

```
double scale = (200+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%



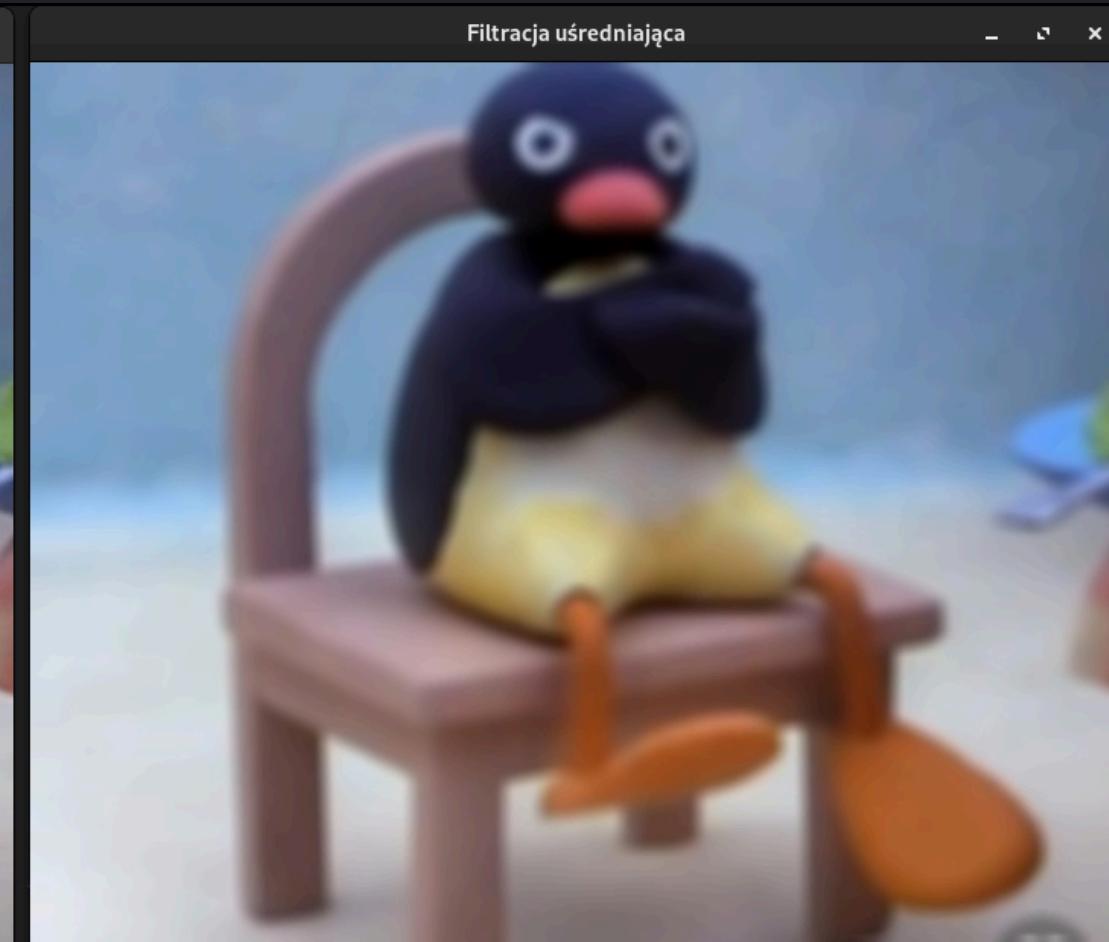
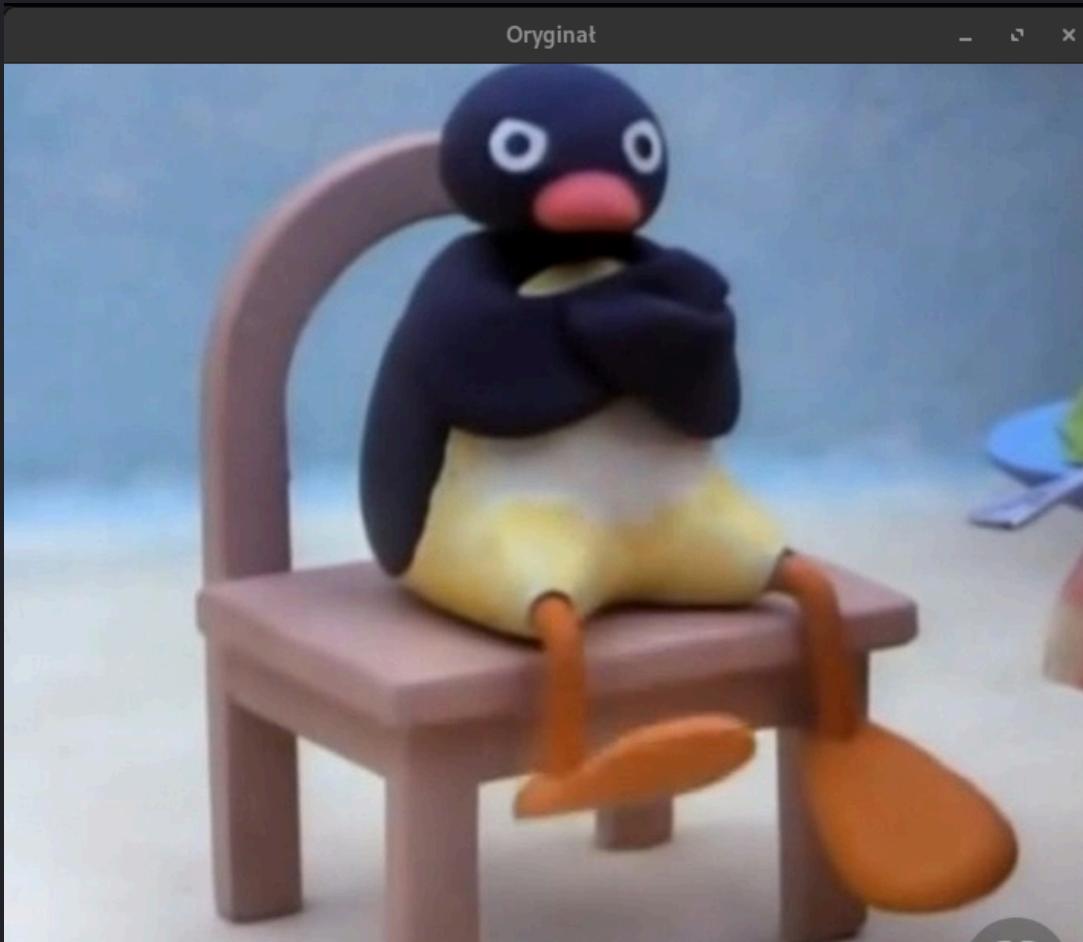
Filtracja (na przykładzie uśredniającej)

```
int windowHeight = 5;

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

//Wykonanie filtracji uśredniającej
imgProc.blur(img, out, new Size(filterMask, filterMask));
```

Dla rozmiaru okna=5



Detekcja krawędzi metodą Canny'ego

```
Mat img = new Mat();
int T_min = 35;          //Próg minimalny (min. 0)
T_max = T_min*3;        //Obliczenie wartości maksymalnego progu T_max

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

//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 i dylatacja

```
int elementType = 1;
int baseSize = 9;

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);
imgProc.dilate(img, out2, kernel, new Point(-1,-1), 1, Core.BORDER_CONSTANT);
```

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



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 i geometrycznych,
- rysowanie po obrazie (np. dopisywanie tekstu, rysowanie prostokątów),
- wykorzystanie innych operatorów (np. operator Scharra),
- wykorzystanie innych metod wykrywania krawędzi,
- wykorzystanie transformacji Laplace,
- detekcja twarzy na obrazie,
- wyrównanie histogramu,
- wykorzystanie transformacji linii Hougha,
- 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

