Sprawozdanie 1. AiPO Mirosław Kołodziej

April 3, 2023

1 Laboratorium 2

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
import cv2
from google.colab.patches import cv2_imshow
```

1.1 a) Proszę wczytać przykładowy obraz

[ ]:
```
image = cv2.imread("/content/drive/MyDrive/AiPO/lab2.jpg")
```

1.2 b) Dokonaj konwersji obrazu do skali szarości. Dlaczego wynikowy obraz nie jest tworzony po prostu przez dodanie składowych R+G+B z wagą 1?

[ ]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

Ponieważ konwersja jest wykonywana metodą bardziej odpowiednią dla ludzkiego oka - modelu YUV. Ludzkie oko jest bardziej wyczulone na kolor zielony, a najmniej na kolor niebieski.

1.3 c) Proszę rozdzielić składowe RGB obrazu i wyświetlić trzy obrazy w skali szarości odpowiadające intensywności pierwotnego obrazu osobno w składowych R, G i B.

[ ]:
```
b,g,r = cv2.split(image)

cv2_imshow(b)

cv2_imshow(g)

cv2_imshow(r)
```

1.4 d) Dokonaj konwersji obrazu do skali HSV i wyświetl trzy obrazy w skali szarości odpowiadające intensywności pierwotnego obrazu osobno w składowych H, S i V.

[ ]:
```
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

h, s, v = cv2.split(hsv)

cv2_imshow(h)

cv2_imshow(s)

cv2_imshow(v)
```

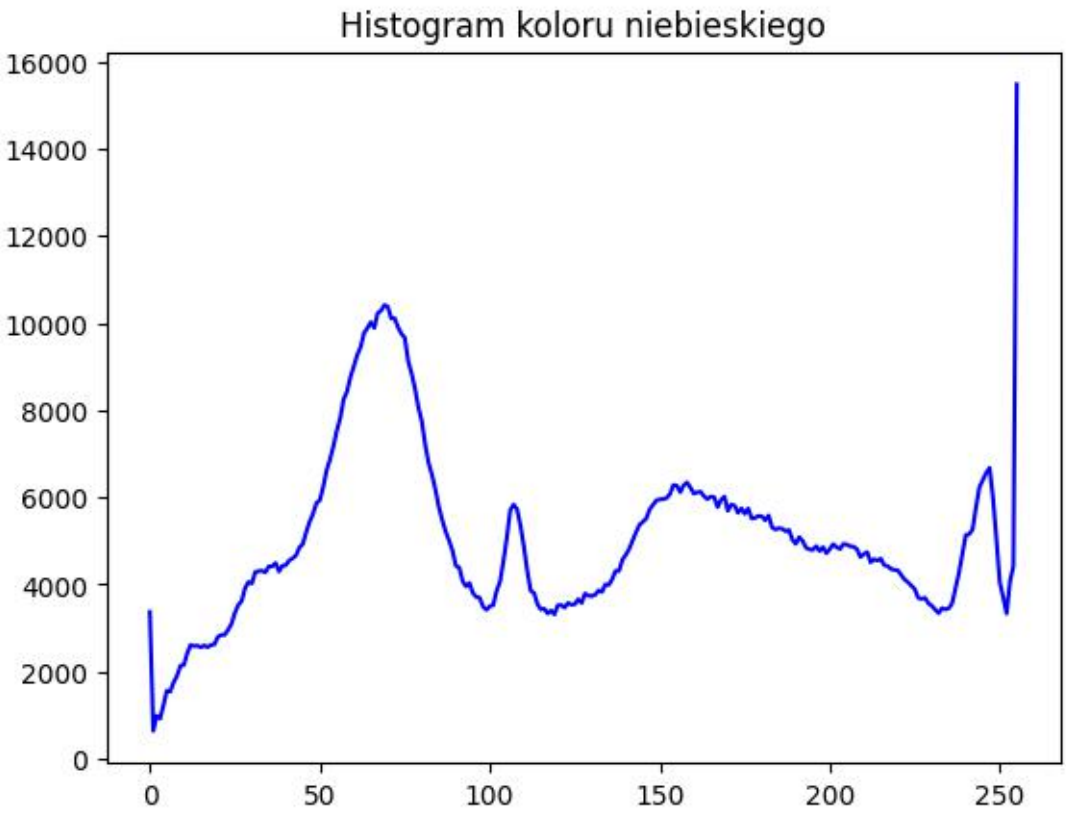
1.5 e) Stwórz obraz na którym widoczny będzie jedynie żółty samochód - to cenne ćwiczenie w zagadnieniach śledzenia obiektów. Realizujemy to w skali HSV- dlaczego?

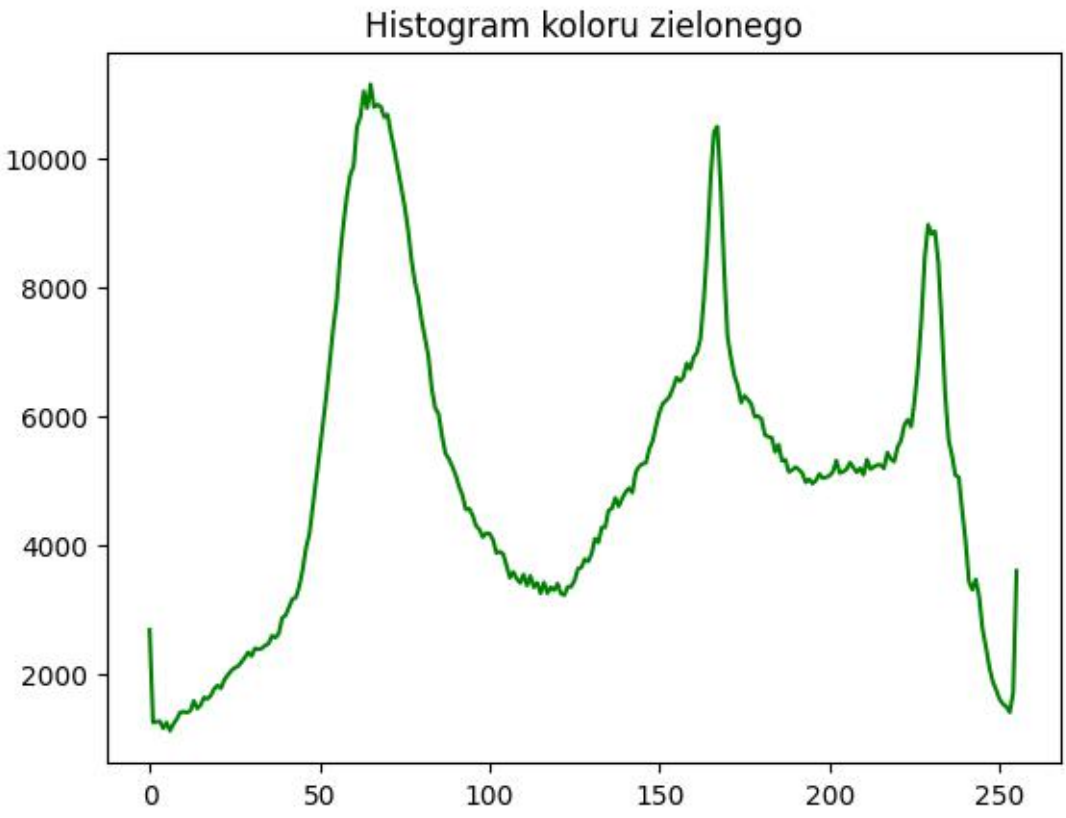
[ ]:
```
mask = cv2.inRange(hsv, (20,100,100), (36, 255, 255))
target = cv2.bitwise_and(image,image, mask=mask)

cv2_imshow(target[350:600,900:1350])
```

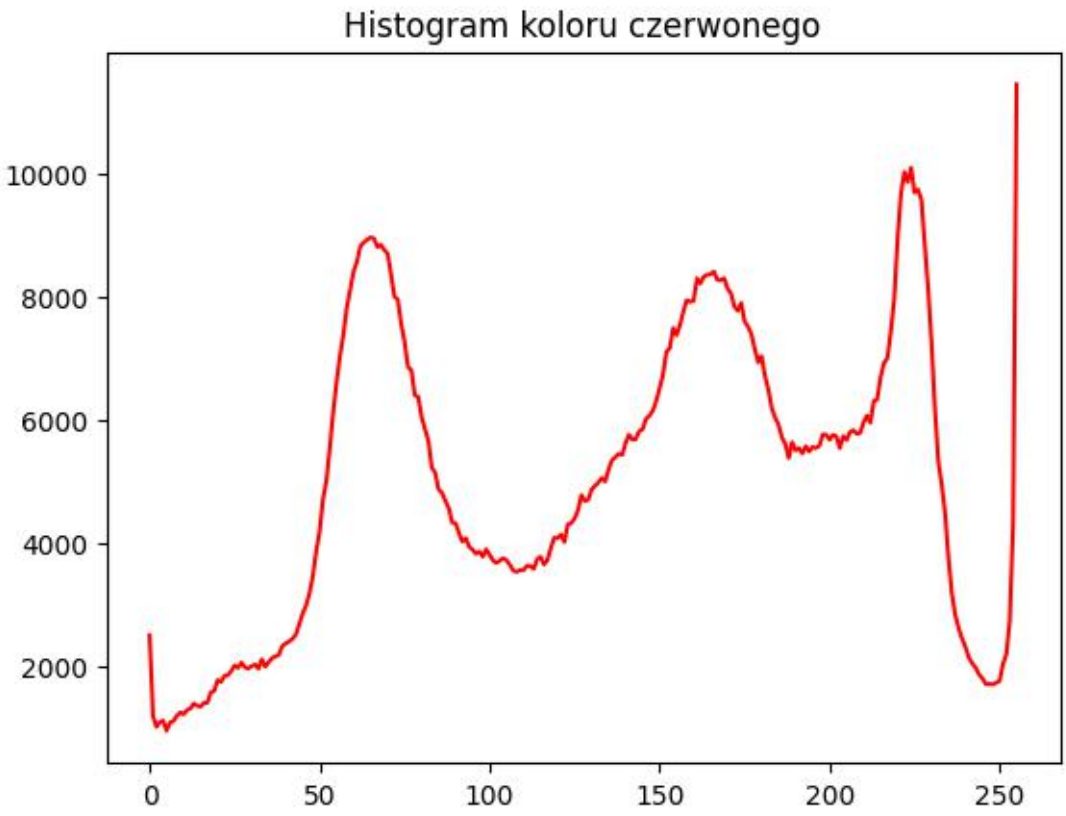
Realizujemy ćwiczenie w modelu HSV, ponieważ można łatwo wyodrębnić kolor żółty. Jak możemy zauważyć na drugim i trzecim kanale modelu HSV, samochód jest obiektem mocno wyróżniającym się z powodu tego, że jest on koloru żółtego. Poza tym, skala HSV jest lepiej widziana przez ludzkie oko niż RGB.

1.6 f) Proszę narysować histogram obrazu dla trzech składowych R,G,B

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt
hist_b = cv2.calcHist([image],[0],None,[256],[0,256])
hist_g = cv2.calcHist([image],[1],None,[256],[0,256])
hist_r = cv2.calcHist([image],[2],None,[256],[0,256])

plt.plot(hist_b, color='b')
plt.title('Histogram koloru niebieskiego')
plt.show()
plt.plot(hist_g, color='g')
plt.title('Histogram koloru zielonego')
plt.show()
plt.plot(hist_r, color='r')
plt.title('Histogram koloru czerwonego')
plt.show()
```


Histogram koloru niebieskiego
16000
14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250


Histogram koloru zielonego
10000
8000
6000
4000
2000
0
50
100
150
200
250

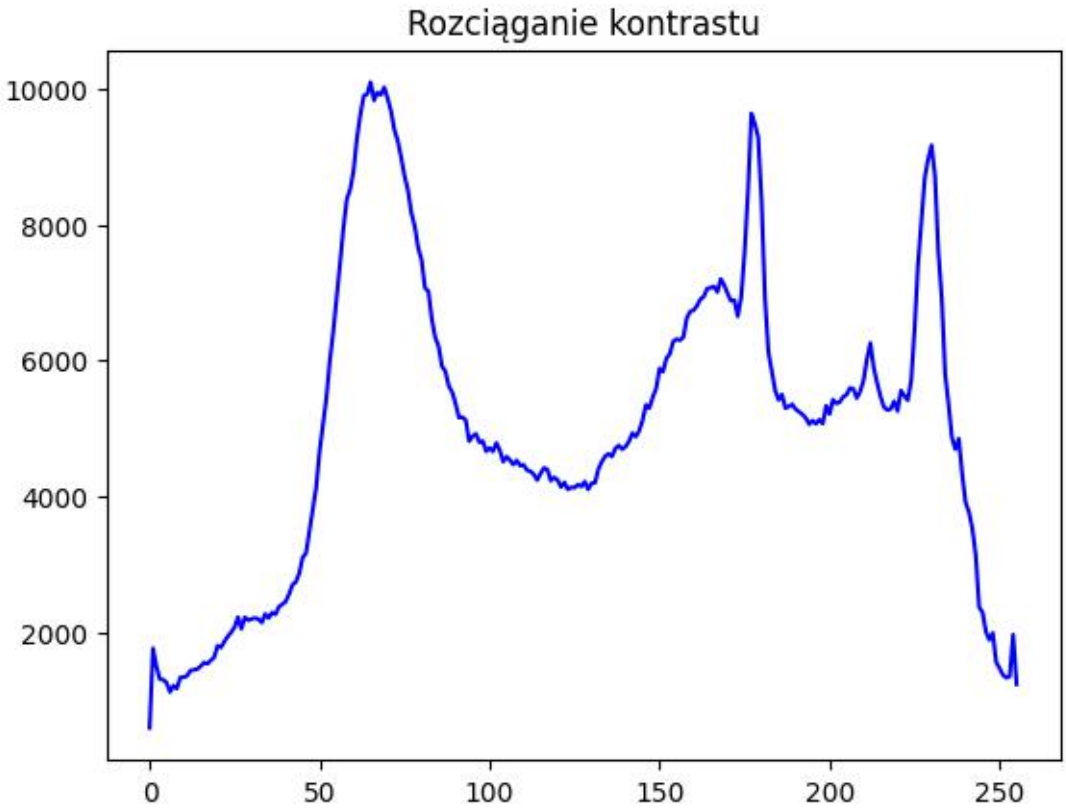


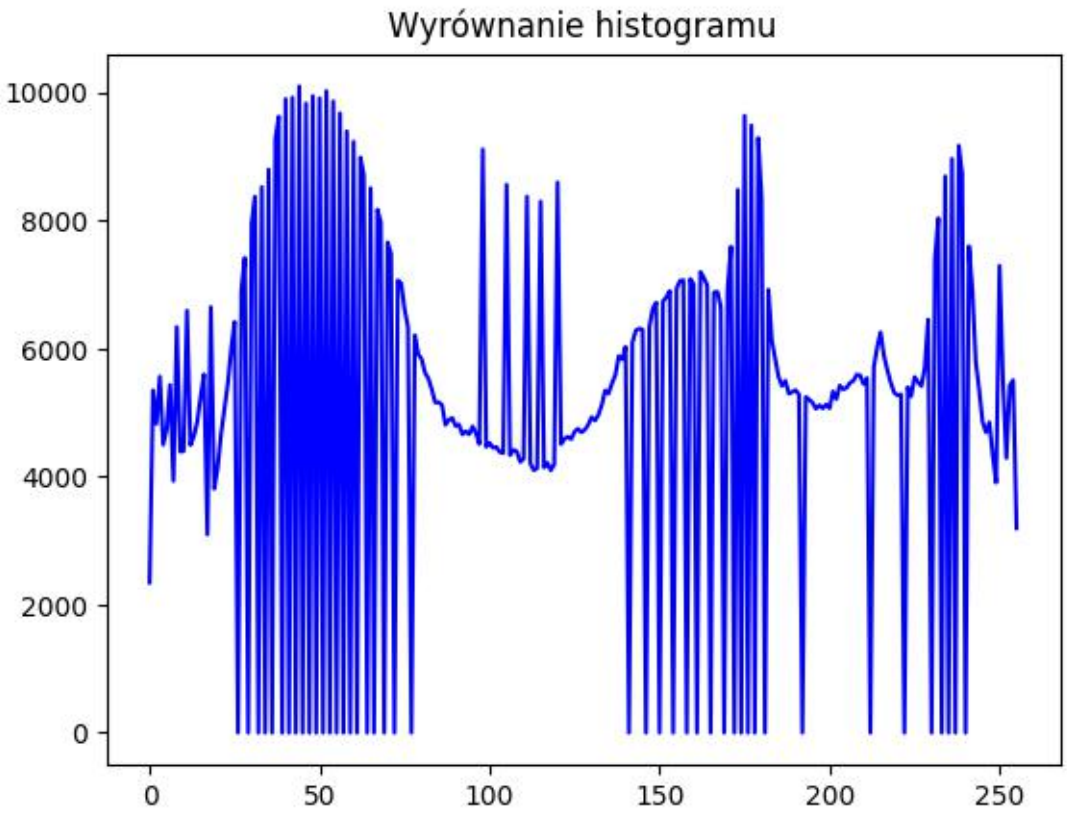
1.7 g) Proszę zrealizować rozciąganie kontrastu i wyrównanie histogramu – dla obrazu zapisanego w skali szarości. Czym różni się wynik tych dwóch operacji?

[ ]:
```
con_strech = gray.copy()
min = np.min(con_strech)
max = np.max(con_strech)
for pixel in con_strech:
  pixel = ((pixel-min)/(max-min))*255
cv2_imshow(con_strech)
```

[ ]:
```
cv2_imshow(cv2.equalizeHist(gray))
```

[ ]:
```
hist_rk = cv2.calcHist([con_strech],[0],None,[256],[0,256])

plt.plot(hist_rk, color='b')
plt.title('Rozciąganie kontrastu')
plt.show()

hist_eh = cv2.calcHist([cv2.equalizeHist(gray)],[0],None,[256],[0,256])

plt.plot(hist_eh, color='b')
plt.title('Wyrównanie histogramu')
plt.show()
```





Wyrównanie histogramu to metoda polegająca na poprawianiu kontrastu analizowanego obrazu z wykorzystaniem jego histogramu. Z jej pomocą wyrównywane są wzniesienia oraz wgłębienia na histogramie akumulacyjnym tworząc obraz, który korzysta bardziej równomiernie ze wszystkich z kolorów.

Rozciąganie kontrastu polega zaś na proporcjonalnym zwiększeniu kontrastu dla wszystkich występujących jasności, od minimalnej do maksymalnej, czyli od najciemniejszej do najjaśniejszej. Rozrzesza po prostu przedział kolorów obrazu tak aby korzystał on z najjaśniejszego oraz najciemniejszego.

W przypadku tego obrazu przy operacji rozciągania kontrastu nie zobaczymy różnicy, ponieważ obraz korzysta już ze maksymalnej oraz minimalnej wartości z przedziału <0, 255>.

Wynik tych dwóch operacji różni się tym, że część wartości na hstogramie rozciągniecia została wyzerowana, a wzniesienia na nim się rozszerzyły. Natomiast w przypadku rozciągania kontrastu nie zaszły żadne zmiany.

2 Laboratorium 3.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
import cv2
from google.colab.patches import cv2_imshow
```

2.1 a) Proszę wczytać przykładowy obraz i skonwertować go do skali szarości.

[ ]:
```
image = cv2.imread("/content/drive/MyDrive/AiPO/lab3_1.jpg")
```

[ ]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

2.2 b) Proszę sprawdzić i porównać wynik progowania ze statycznym ustalonym globalnie progiem zrealizowanego z użyciem każdego z dostępnych w OpenCV trybu progowania (cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO i cv2.THRESH_TOZERO_INV)

[ ]:
```
ret, thresh1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(gray, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(gray, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(gray, 120, 255, cv2.THRESH_TOZERO_INV)

print("THRESH_BINARY")
cv2_imshow(thresh1)
print("THRESH_BINARY_INV")
cv2_imshow(thresh2)
print("THRESH_TRUNC")
cv2_imshow(thresh3)
print("THRESH_TOZERO")
cv2_imshow(thresh4)
print("THRESH_TOZERO_INV")
cv2_imshow(thresh5)
```

```
THRESH_BINARY
```

THRESH_BINARY_INV

THRESH_TRUNC

THRESH_TOZERO

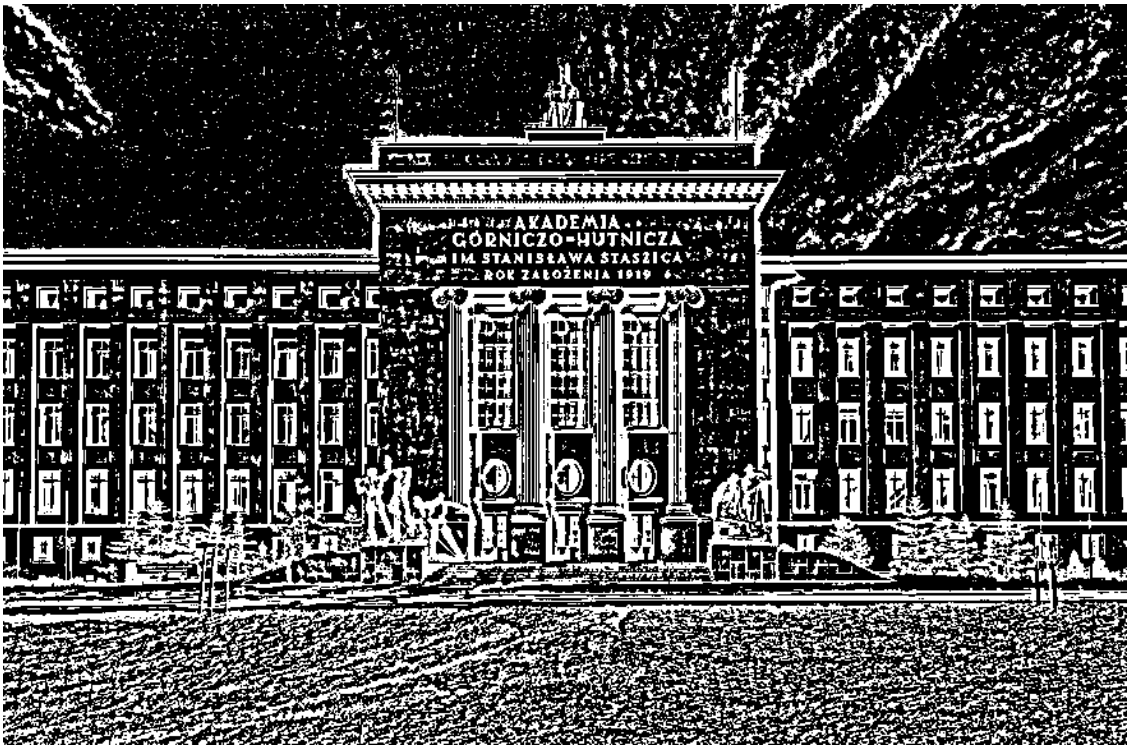
THRESH_TOZERO_INV

2.3 c) Proszę zastosować metody adaptacyjnego doboru progu. Jaki wpływ na wynik ma rozmiar sąsiedztwa? Która z metod daje lepsze rezultaty?

[ ]:
```
ret, th1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.
 ↪THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
 ↪THRESH_BINARY,11,2)
cv2_imshow(th1)
cv2_imshow(th2)
cv2_imshow(th3)
```


AKADEMIA
GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
ROK ZAŁOŻENIA 1919

[ ]:
```
ret, th1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY_INV)
th2 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.
 ↪THRESH_BINARY_INV,11,2)
th3 = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.
 ↪THRESH_BINARY_INV,11,2)
cv2_imshow(th1)
cv2_imshow(th2)
cv2_imshow(th3)
```


AKADEMIA
GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
ROK ZAŁOŻENIA 1919

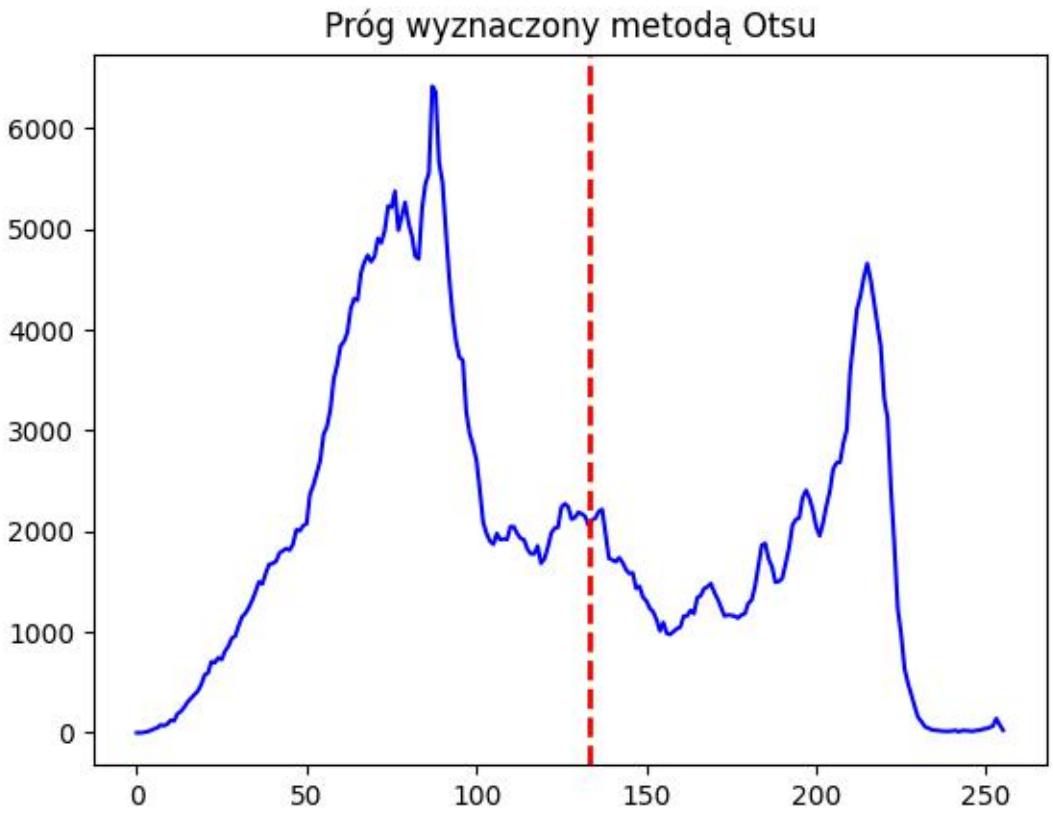
Rozmiar sąsiedztwa może wpłynąć na dokładność progowania. Jeśli zostanie wybrane zbyt duże sąsiedztwo (bądź zbyt małe) otrzymany obraz może być w mniejszym stopniu podobny do oryginału.

Dla metod adaptacyjnych, lepszy wynik zdaje się dawać metoda Gaussa, ponieważ zostawia mniej plam w tle o mniejszych rozmiarach. Napisy również wydają się bardziej wyraźne.

2.4 d) Proszę zastosować metodę Otsu, wykreślić histogram obrazu i zlokalizować na nim uzyskaną wartość progu.

[ ]:
```
ret, thresh1 = cv2.threshold(gray, 120, 255, cv2.THRESH_OTSU)
cv2_imshow(thresh1)
```

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt
hist = cv2.calcHist([gray],[0],None,[256],[0,256])


plt.plot(hist, color='b')
plt.axvline(x=ret, color='r', linestyle='dashed', linewidth=2)
plt.title('Próg wyznaczony metodą Otsu')
plt.show()
print(ret)
```




```
133.0
```

2.5 e) Proszę wczytać drugi z obrazów i dokonać jego segmentacji z użyciem algorytmu k średnich na 2, 4, 8 klastrów. Proszę zweryfikować wizualnie wynik klasteryzacji (kolor każdego piksela to kolor środka klastra do którego go przypisaliśmy) i sprawdzić czy kolory odpowiadają jakimś cechom zdjęcia które analizujemy.

[ ]:
```
import numpy as np
image2 = cv2.imread("/content/drive/MyDrive/AiPO/lab3_2.png")
cv2_imshow(image2)

pixels = image2.reshape((-1, 3))
pixels = np.float32(pixels)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
```

[ ]:
```
_, labels, (centers) = cv2.kmeans(pixels, 2, None, criteria, 10, cv2.
↪KMEANS_RANDOM_CENTERS)

centers = np.uint8(centers)

labels = labels.flatten()
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image2.shape)
cv2_imshow(segmented_image)
```

[ ]:
```
_, labels, (centers) = cv2.kmeans(pixels, 4, None, criteria, 10, cv2.
↪KMEANS_RANDOM_CENTERS)

centers = np.uint8(centers)

labels = labels.flatten()
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image2.shape)
cv2_imshow(segmented_image)
```

[ ]:
```
_, labels, (centers) = cv2.kmeans(pixels, 8, None, criteria, 10, cv2.
↪KMEANS_RANDOM_CENTERS)

centers = np.uint8(centers)

labels = labels.flatten()
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image2.shape)
cv2_imshow(segmented_image)
```

Podzbiory w klasteryzacji wydają się być wyodrębnione prawidłowo. Zwiększanie ilości klastrów powoduje wzrost podobieństwa do oryginału, zatem klasteryzacja wydaje się zachodzić poprawnie. Kolory są podobne do początkowych kolorów z oryginalnego zdjęcia. Dodatkowo, kształty obiektów zostały zachowane.

2.6 f) Proszę samodzielnie zaimplementować metodę Otsu. Czy otrzymany próg różni się od bibliotecznej implementacji? Jeśli tak to dlaczego?

[ ]:
```
# Wyznaczenie histogramu dla obrazka
hist, bin_edges = np.histogram(gray, bins=256)

# obliczenie wartości średniej dla każdej pary sąsiadujących ze sobą kolumn
bin_mids = (bin_edges[:-1] + bin_edges[1:]) / 2.

# Obliczenie prawdopodobieństwa
weight1 = np.cumsum(hist)
weight2 = np.cumsum(hist[::-1])[::-1]

# obliczenie średnich
mean1 = np.cumsum(hist * bin_mids) / weight1
mean2 = (np.cumsum((hist * bin_mids)[::-1]) / weight2[::-1])[::-1]
```

```
inter_class_variance = weight1[:-1] * weight2[1:] * (mean1[:-1] - mean2[1:]) **␣
↪2
index_of_max_val = np.argmax(inter_class_variance)

threshold = bin_mids[:-1][index_of_max_val]
print("Wartość wyznaczona przez implementację algorytmu Otsu:", threshold)
```

```
Wartość wyznaczona przez implementację algorytmu Otsu: 132.947265625
```

Otrzymany próg różni się nieznacznie od funkcji bibliotecznej. Prawdopodobnie jest to związane z tym, że funkcja biblioteczna zaokrągla na koniec otrzymany wynik.

3 Laboratorium 4.

3.1 a) Proszę wczytać przykładowy obraz (pierwszy z laboratorium 3).

[ ]:
```
import cv2
from google.colab.patches import cv2_imshow
```

[ ]:
```
image = cv2.imread("/content/drive/MyDrive/AiPO/lab3_1.jpg")
cv2_imshow(image)
```

3.2 b) Proszę sprawdzić i porównać wynik następującej operacji: zmniejszania rozmiaru obrazu o 50 % - jedną z wybranych metod - a następnie zwiększenie o 50% z użyciem wszystkich dostępnych w OpenCV metod interpolacji. Proszę porównać uzyskany wynik z obrazem pierwotnym.

[ ]:
```
half = cv2.resize(image, (0, 0), fx = 0.5, fy = 0.5)
cv2_imshow(half)
```

[ ]:
```
IL = cv2.resize(image, (image.shape[1]*2, image.shape[0]*2),
                interpolation = cv2.INTER_LINEAR)
cv2_imshow(IL)
```

[ ]:
```
IN = cv2.resize(image, (image.shape[1]*2, image.shape[0]*2),
                interpolation = cv2.INTER_NEAREST)
cv2_imshow(IN)
```

[ ]:
```
IA = cv2.resize(image, (image.shape[1]*2, image.shape[0]*2),
                interpolation = cv2.INTER_AREA)
cv2_imshow(IA)
```

[ ]:
```
IC = cv2.resize(image, (image.shape[1]*2, image.shape[0]*2),
                interpolation = cv2.INTER_CUBIC)
cv2_imshow(IC)
```

[ ]:
```
IL4 = cv2.resize(image, (image.shape[1]*2, image.shape[0]*2),
                 interpolation = cv2.INTER_LANCZOS4)
cv2_imshow(IL4)
```

Dla obrazów wynikowych metod NEAREST oraz AREA można zauważyć "ziarno", dodatkowo jakość powiększenia nie jest zadowalająca. Najlepszy wynik został uzyskany dla metody LINEAR, która dała obraz najbardziej zbliżony do oryginału (pomimo lekkiego rozmycia). Pozostałe metody (CUBIC I LANCZOS4) uzyskują podobne wyniki, które można porównać do metody LINEAR z lekkim szumem.

3.3 c) Proszę nałożyć na obraz filtr uśredniający o macierzy K w rozmiarze 5x5, 10x10 i 15x15. Jaki efekt został zaobserwowany?

[ ]:
```
b1 = cv2.blur(image,(5,5))
cv2_imshow(b1)
```

[ ]:
```
b2 = cv2.blur(image,(10,10))
cv2_imshow(b2)
```

[ ]:
```
b3 = cv2.blur(image,(15,15))
cv2_imshow(b3)
```

Dzięki filtrowi uśredniającemu został zaobserwowany efekt rozmycia zwiększający się wraz z rozmiarem macierzy.

3.4 d) Proszę nałożyć na obraz filtr medianowy o rozmiarze 5x5, 11x11 i 15x15. Jaki efekt został zaobserwowany?

[ ]:
```
mb1 = cv2.medianBlur(image,5)
cv2_imshow(mb1)
```

[ ]:
```
mb2 = cv2.medianBlur(image,11)
cv2_imshow(mb2)
```

[ ]:
```
mb3 = cv2.medianBlur(image,15)
cv2_imshow(mb3)
```

Użycie filtru medianowego pozwoliło na usunięcie zakłóceń oraz szumów. Zwiększenie rozmiarów macierzy spowodowało utratę coraz większej ilości szczegółów.

3.5 e) Proszę nałożyć na obraz filtr gaussowski (o rozmiarze 5) i zaobserwować uzyskane wyniki.

[ ]:
```
gb = cv2.GaussianBlur(image,(5,5),0)
cv2_imshow(gb)
```

Filtr Gaussowski wprowadza wygładzenie obrazu, szczególnie na krawędziach obiektów. Napisy są lepiej widoczne niż w przypadku rozmycia.

3.6 f) Proszę zbinaryzować obraz po przefiltrowaniu go powyższymi metodami i porównać do binaryzacji bez filtracji.

[ ]:
```
#binaryzacja bez filtracji
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
cv2_imshow(thresh1)
```

[ ]:
```
#filtr uśredniający + binaryzacja
b1 = cv2.blur(gray,(5,5))
ret, thresh1 = cv2.threshold(b1, 120, 255, cv2.THRESH_BINARY)
cv2_imshow(thresh1)
```

[ ]:
```
#filtr medianowy + binaryzacja
mb1 = cv2.medianBlur(gray,5)
ret, thresh1 = cv2.threshold(mb1, 120, 255, cv2.THRESH_BINARY)
cv2_imshow(thresh1)
```

[ ]:
```
#filtr gaussowski + binaryzacja
gb = cv2.GaussianBlur(gray,(5,5),0)
ret, thresh1 = cv2.threshold(gb, 120, 255, cv2.THRESH_BINARY)
cv2_imshow(thresh1)
```

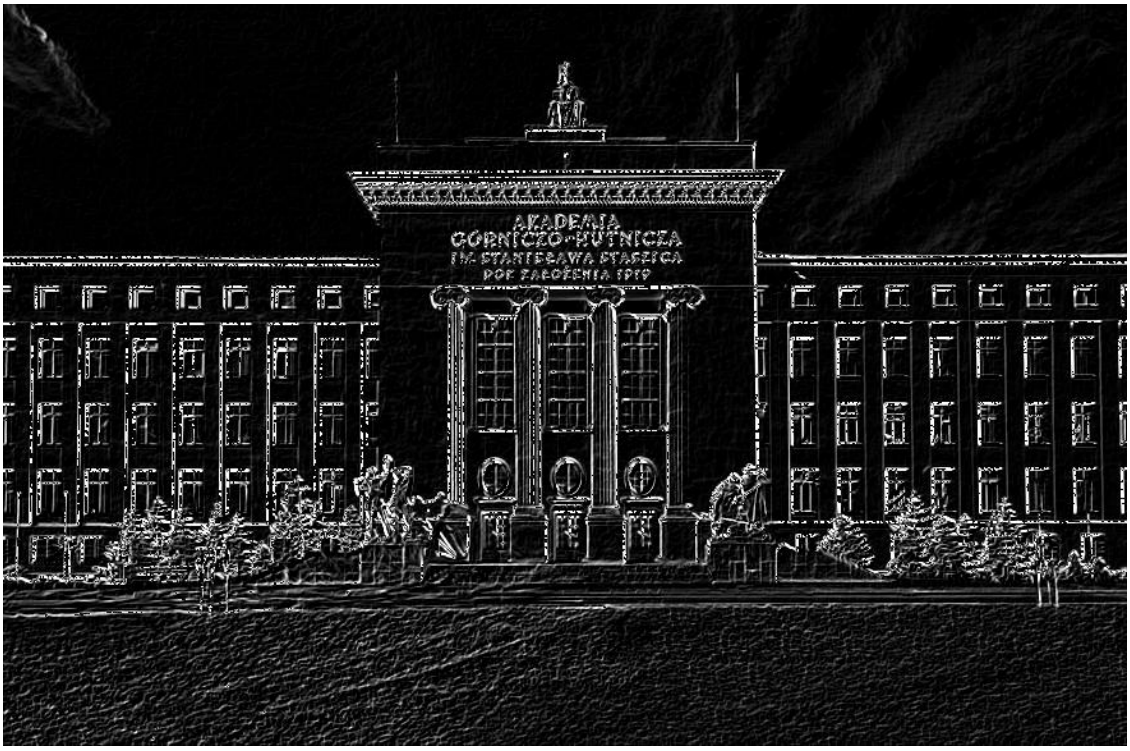
Binaryzacja bez filtracji zachowuje na obrazach więcej szczegółów. Z użytych metod filtracji, najlepsze efekty daje filtr Gaussa, który np. zachował większość krawędzi okien.

3.7 g) Proszę nałożyć na obraz filtry Roberts cross, Prewitta i Sobela

[ ]:
```
import numpy as np
roberts_cross_v = np.array( [[1, 0 ],
                             [0,-1 ]] )

roberts_cross_h = np.array( [[ 0, 1 ],
                             [ -1, 0 ]] )
roberts_image = cv2.filter2D(gray, -1, roberts_cross_v) + cv2.filter2D(gray,␣
 ↪-1, roberts_cross_h)
cv2_imshow(roberts_image)
```

[ ]:
```
prewitt_x = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=np.float32)
prewitt_y = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype=np.float32)
prewitt_image = cv2.filter2D(gray, -1, prewitt_x) + cv2.filter2D(gray, -1,␣
↪prewitt_y)
cv2_imshow(prewitt_image)
```

[ ]:
```
import numpy as np
sobel_v = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

sobel_h = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
sobel_image = cv2.filter2D(gray, -1, sobel_v) + cv2.filter2D(gray, -1, sobel_h)
cv2_imshow(sobel_image)
```



Filtry Sobela i Prewitta dały lepsze wyniki w wyodrębnianiu krawędzi ze względu na to, że krzyż Robertsa służy do wykrywania ukośnych krawędzi, przez co zaznaczone przez niego krawędzie są cienkie.