

# **Sprawozdanie z przedmiotu**

# **Podstawy Cyfrowego Przetwarzania Obrazów**

Krzysztof Ciszek  
czerwiec 2023

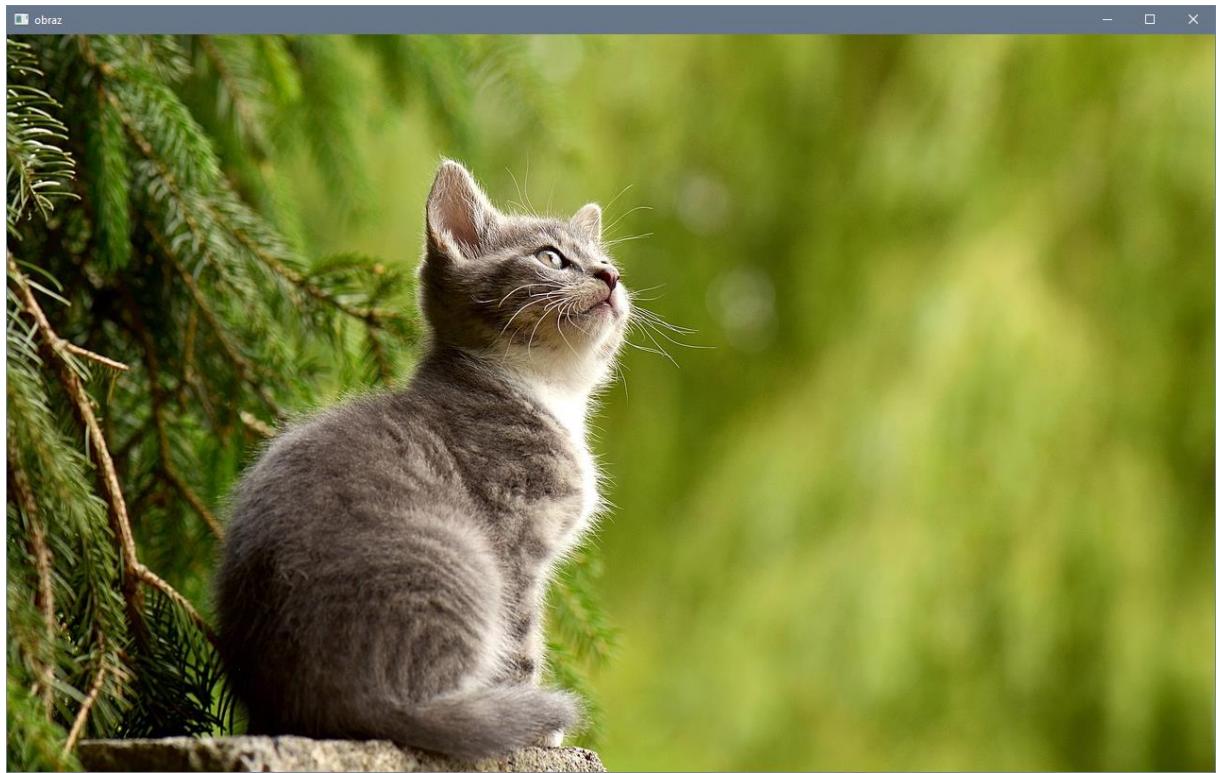
Opisywane poniżej zagadnienia to tylko część zadań wykonanych na przedmiocie PCPO. Kod źródłowy pozwalający na uzyskanie odpowiednich efektów załączam jako osobne pliki. W niektórych miejscach, jeśli kod był krótki, dodałem też screeny kodu, lub jego najważniejsze fragmenty.

## **1. Wczytywanie/zapis obrazu oraz operacje na barwach**

- Kod w folderze **p1** i **p4**
- Plik **main\_p1.py** i **main\_p4\_histogarmy\_binaryzacja.py**

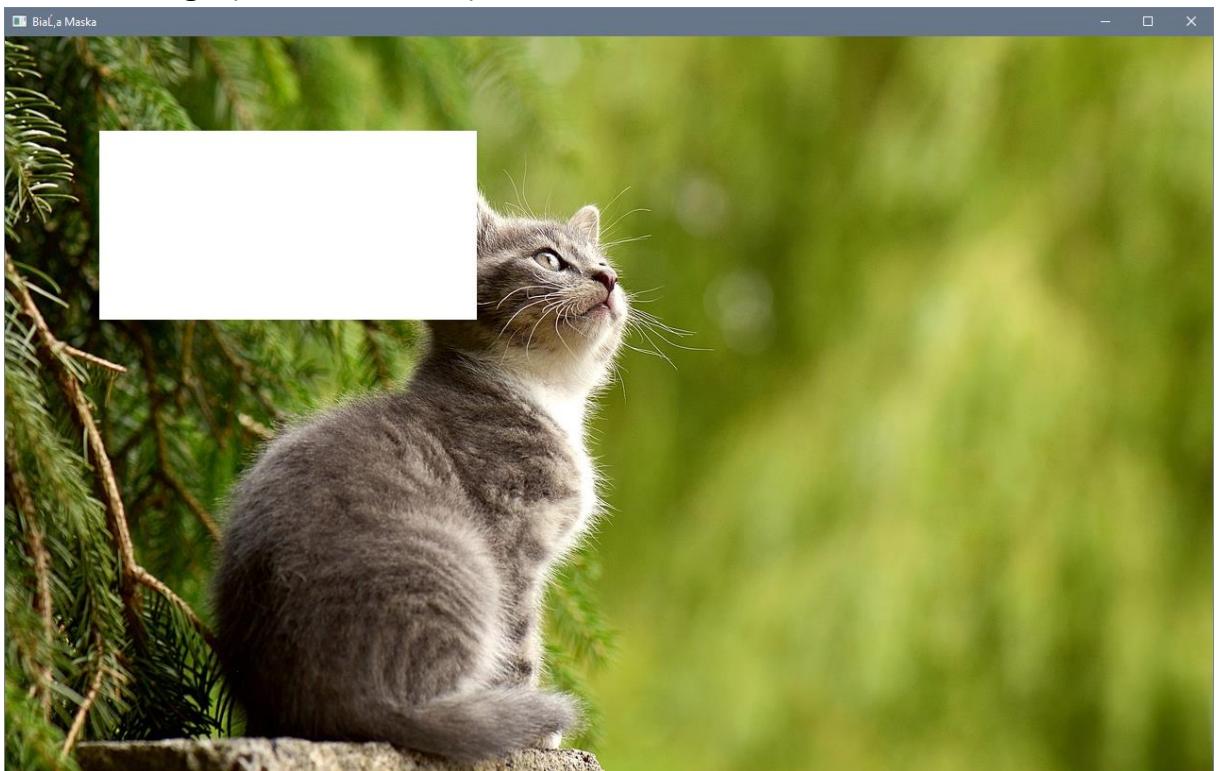
W opencv obraz wczytujemy za pomocą funkcji cv2.imread(). Obraz jest reprezentowany za pomocą macierzy numpy (o odpowiednich wymiarach, w zależności od liczby kanałów obrazu), w której każda liczba oznacza wartość intensywności piksela w danym miejscu dla danego kanału. Obraz wyświetlamy za pomocą funkcji cv2.imshow() w połączeniu z cv2.waitKey(). Obraz (macierz) zapisujemy za pomocą funkcji cv2.imwrite(). Jako, że obraz jest reprezentowany jako macierz, to możemy przeprowadzać na obrazie operacje typowe dla macierzy, m.in. slice'owanie i przypisywanie nowych wartości kolorów. Możemy też użyć funkcji takich jak cv2.add() lub cv2.subtract() w celu zmiany saturacji kolorów. Takich różnych funkcji w bibliotece opencv jest dużo.

## Oryginalny obraz:



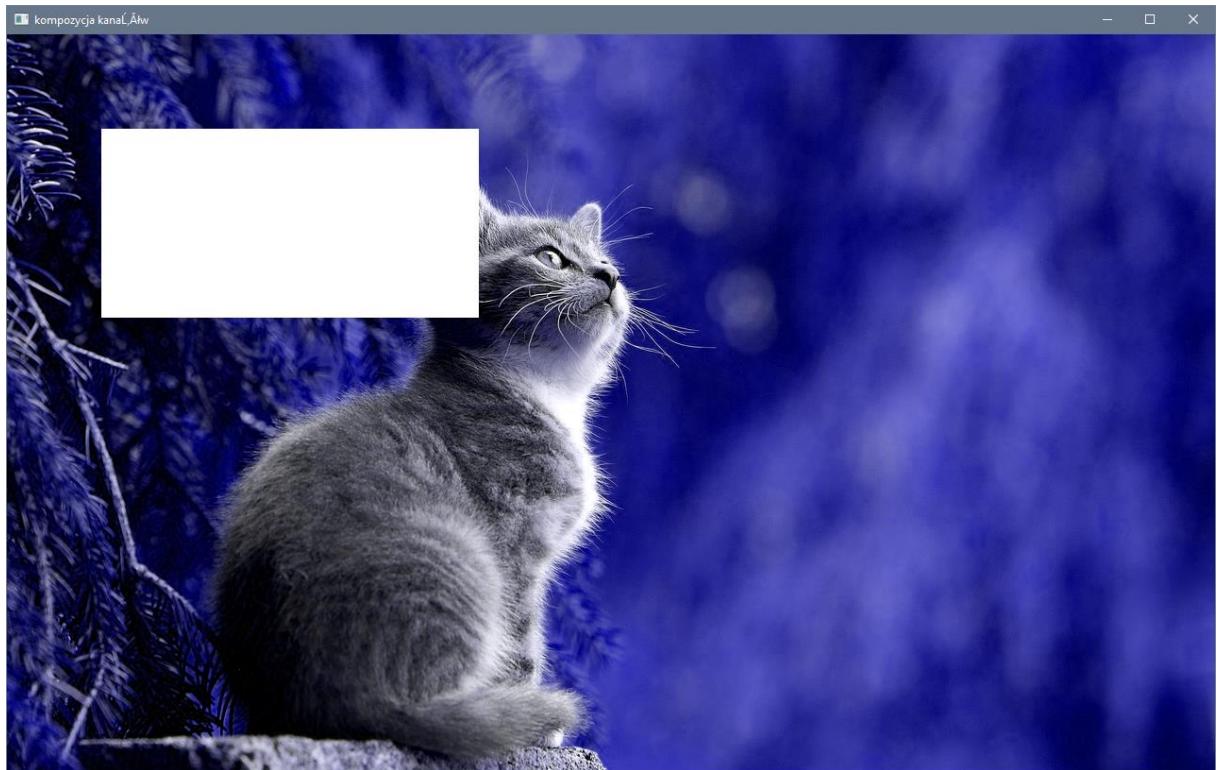
```
print(cv2.__version__)
img_filepath = r"C:\SEM6\PCPO\p1\cat.jpg"
img2_path = r"C:\SEM6\PCPO\p1\butterfly.jpg"
img_savefilepath = r"C:\SEM6\PCPO\p1\wynik.jpg"
img = cv2.imread(img_filepath, flags=-1) # 0 monochromatyczne 1 rgb -1 w/alpha
# cv2.namedWindow("obrazdd", cv2.WINDOW_NORMAL)
cv2.imshow(winname="obraz", mat=img) # wyświetlenie okna
key = cv2.waitKey(0) # oczekiwanie na wcisnięcie przycisku przez użytkownika
if key == 27:
    cv2.destroyAllWindows() # usunięcie okna z obrazem i innych, jeżeli były stworzone
else: # elif key in [83, 115]: # S or s
    cv2.imwrite(img_savefilepath, img)
    cv2.destroyAllWindows()
```

Obraz po zastąpieniu wartości kolorów w wycinku obrazu przez wartości koloru białego ([255, 255, 255]):



```
img[100:300, 100:500] = [255, 255, 255]
```

## Kompozycja kanałów:



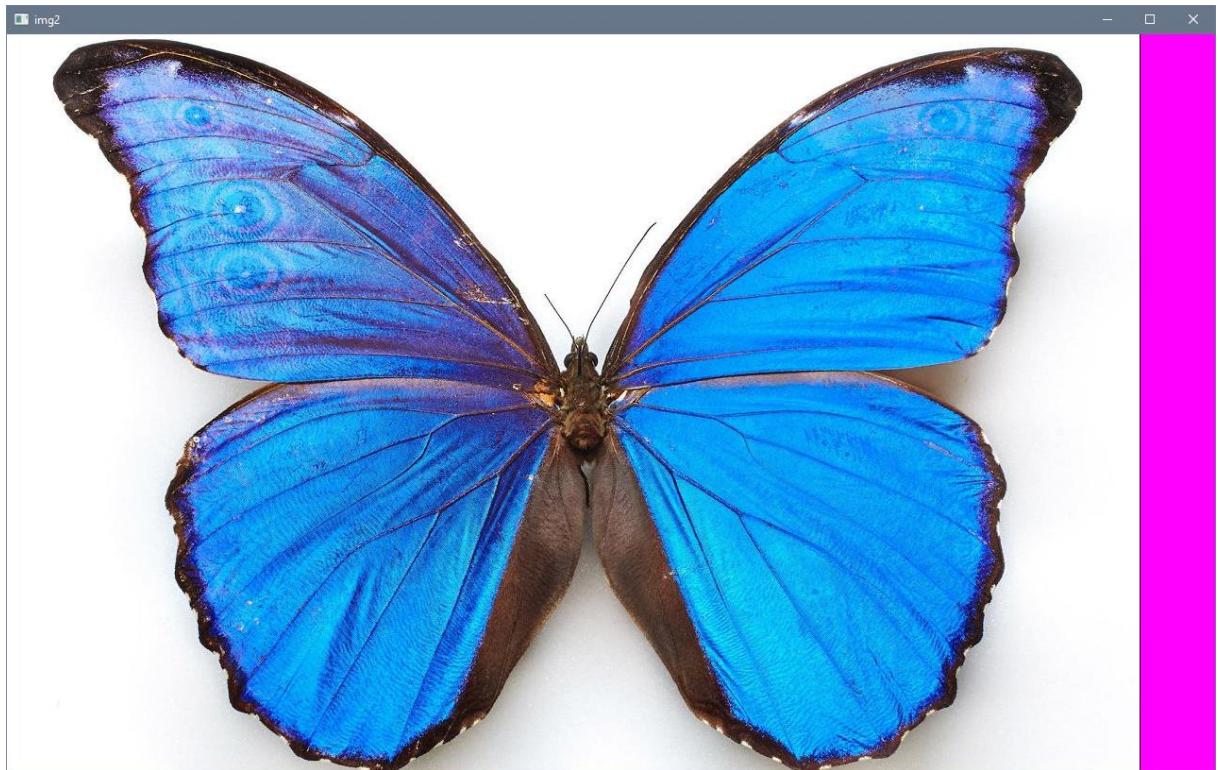
```
b, g, r = cv2.split(m=img) # macierze kanałów B G R # print(g)
obr = cv2.merge((g, b, b))
cv2.imshow(winname="kompozycja kanałów", mat=obr)
```

Fragment obrazu:



```
fragment = obr[100:1200, 150:700]
cv2.imshow(winname="fragment", mat=fragment)
```

Obraz złożony z fragmentu innego obrazu i części wypełnionej manualnie:

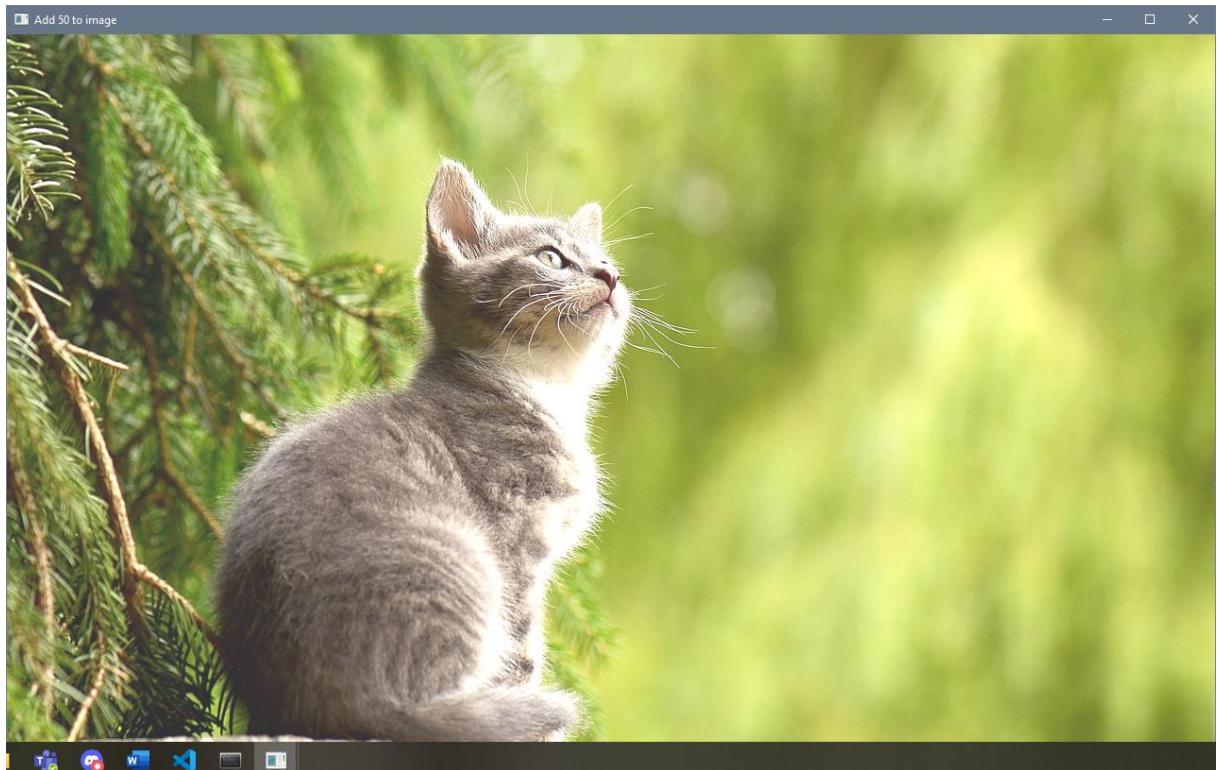


Kompozycja obrazów:

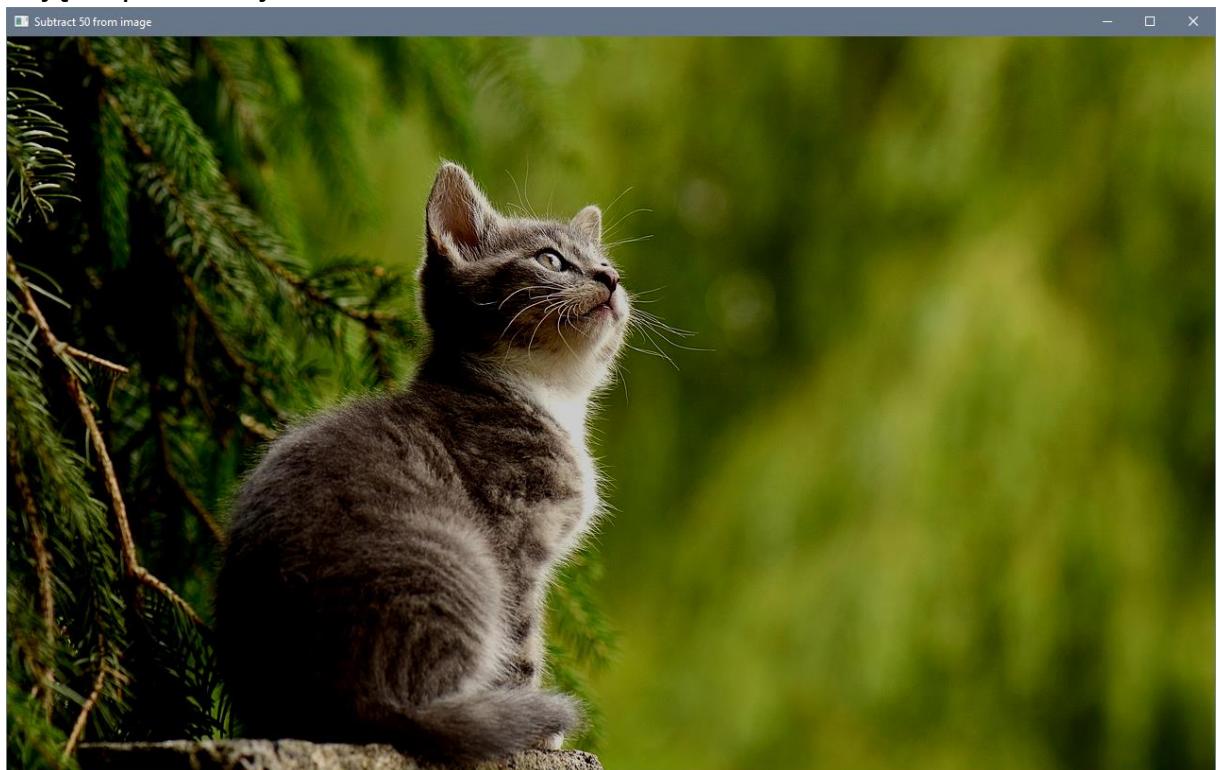


```
new_img = cv2.addWeighted(src1=img, alpha=0.6, src2=fragment2, beta=0.4, gamma=0)
```

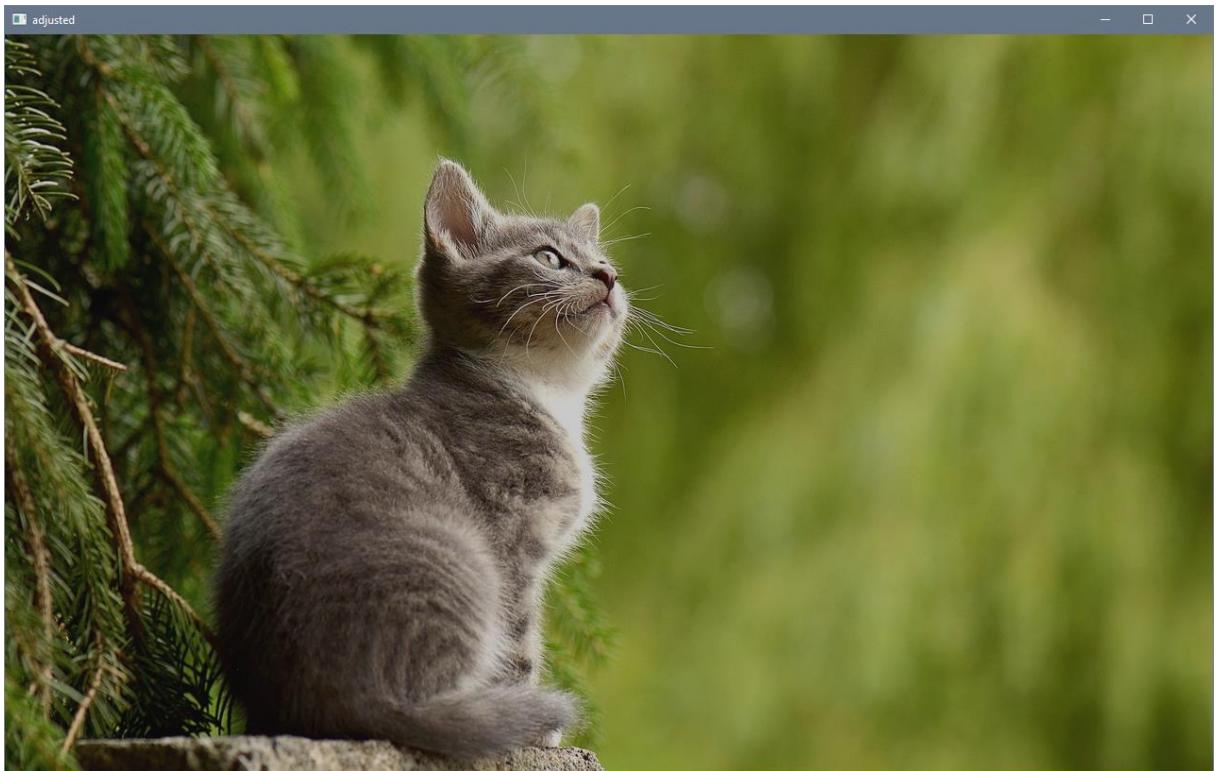
Zdjęcie po zwiększeniu wartości kolorów o 50:



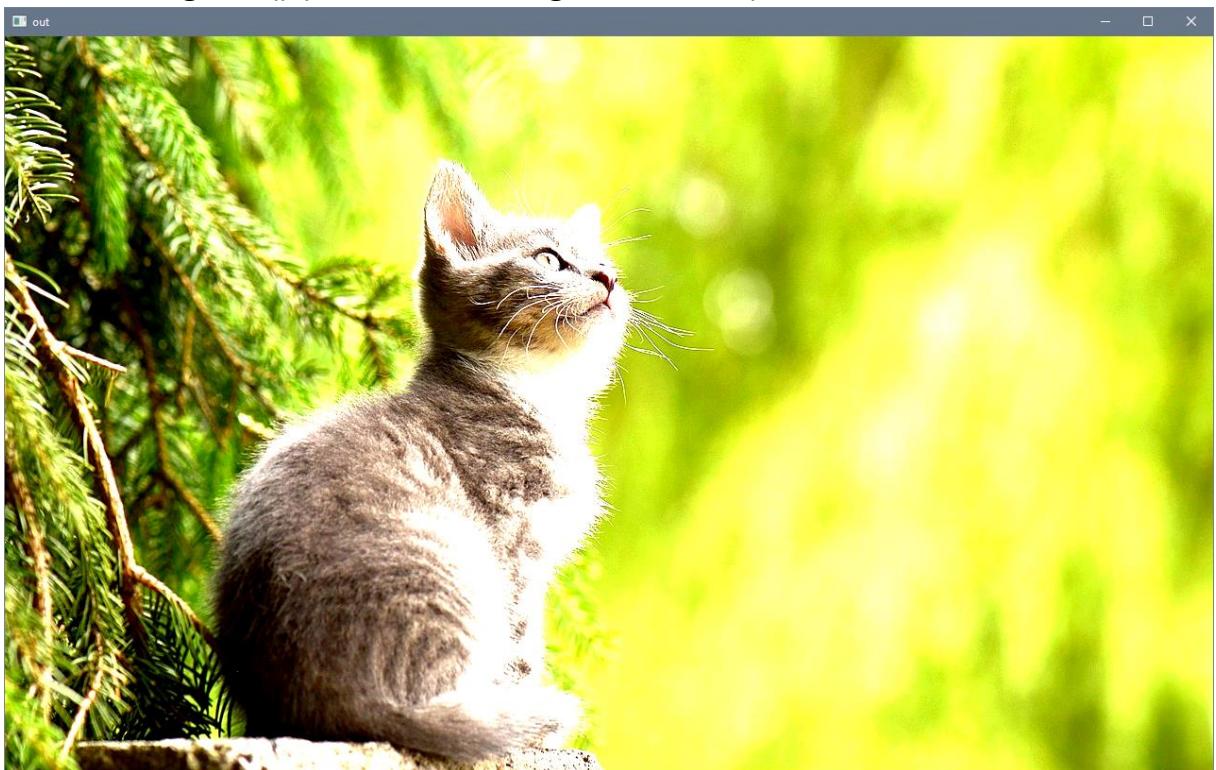
Zdjęcie po zmniejszeniu wartości kolorów o 50:



Zdjęcie po zmianie kontrastu i jasności za pomocą funkcji  
cv2.convertScaleAbs() (alpha = 0.75, beta = 10):



Zdjęcie po zmianie kontrastu i jasności za pomocą funkcji  
cv2.addWeighted() (contrast = 2, brightness = -20):



## 2. Generowanie logo OpenCV

- Kod w folderze **p1**
- Plik **main\_p1.py**

Biblioteka opencv umożliwia tworzenie własnych obrazów, rysowanie na nich podstawowych figur geometrycznych oraz wstawianie tekstu. Poniżej przedstawiam kod, pozwalający na wygenerowanie (narysowanie) logo OpenCV oraz wynikowy obrazek.



```
obrazek = np.ones((900, 900, 3), np.uint8) * 255
thickness = 75
cv2.ellipse(
    img=obrazek,
    center=(int((250 + 600) / 2), 500 - int((600 - 250) * np.sqrt(3) / 2)),
    axes=(125, 125),
    angle=0,
    startAngle=120 - 360,
    endAngle=60,
    color=(0, 0, 255),
    thickness=thickness,
    lineType=cv2.LINE_AA,
)
cv2.ellipse(
    img=obrazek,
    center=(250, 500),
    axes=(125, 125),
    angle=0,
    startAngle=0,
    endAngle=300,
    color=(0, 255, 0),
    thickness=thickness,
    lineType=cv2.LINE_AA,
)
cv2.ellipse(
    img=obrazek,
    center=(600, 500),
    axes=(125, 125),
    angle=0,
    startAngle=-60,
    endAngle=240,
    color=(255, 0, 0),
    thickness=thickness,
    lineType=cv2.LINE_AA,
)
cv2.putText(
    img=obrazek,
    text="OpenCV",
    org=(100, 820),
    fontFace=cv2.FONT_HERSHEY_SIMPLEX,
    fontScale=5.5,
    color=(0, 0, 0),
    thickness=12,
)
cv2.imshow(winname="obrazek", mat=obrazek)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

### 3. Przekształcenia geometryczne obrazów

- Kod w folderze **p2** i **p3**
- Pliki **main\_p2.py** i **main\_p3.py**

Biblioteka opencv udostępnia różne funkcje pozwalające na transformacje geometryczne obrazów. Poniżej przykłady.

Dystorsja radialna:



```

114 # ===== zniekształcenie dystorsja radialna =====
115 zdjecie = cv2.imread(img_filepath, 1)
116 zniekształcenie = 3
117 # zapis do zmiennej wymiarów obrazu
118 (wysokosc, szerokosc, _) = zdjecie.shape
119 # Zdefiniowanie map przekształceń współrzędnych w formacie float32
120 mapaPrzekształcenX = np.zeros((wysokosc, szerokosc), np.float32)
121 mapaPrzekształcenY = np.zeros((wysokosc, szerokosc), np.float32)
122 wspolrzednaXsrodka = szerokosc / 2
123 wspolrzednaYsrodka = wysokosc / 2
124 promienNormalizujacy = szerokosc / 2
125
126 for y in range(wysokosc):
127     deltaY = y - wspolrzednaYsrodka
128     for x in range(szerokosc):
129         deltaX = x - wspolrzednaXsrodka
130         odlegloscOdSrodka = np.power(deltaX, 2) + np.power(deltaY, 2)
131         if odlegloscOdSrodka >= np.power(promienNormalizujacy, 2):
132             mapaPrzekształcenX[y, x] = x
133             mapaPrzekształcenY[y, x] = y
134         else:
135             wspolczynnikZniekształcenia = 1.0 # zniekształcenie = 1 ???
136             if odlegloscOdSrodka > 0.0:
137                 wspolczynnikZniekształcenia = math.pow(
138                     math.sin(math.pi * math.sqrt(odlegloscOdSrodka) / promienNormalizujacy / 2),
139                     zniekształcenie,
140                 )
141             mapaPrzekształcenX[y, x] = wspolczynnikZniekształcenia * deltaX + wspolrzednaXsrodka
142             mapaPrzekształcenY[y, x] = wspolczynnikZniekształcenia * deltaY + wspolrzednaYsrodka
143
144 dst = cv2.remap(zdjecie, mapaPrzekształcenX, mapaPrzekształcenY, cv2.INTER_LINEAR)
145
146 cv2.imshow("Zniekształcone", dst)
147 cv2.waitKey(0)

```

Translacja:

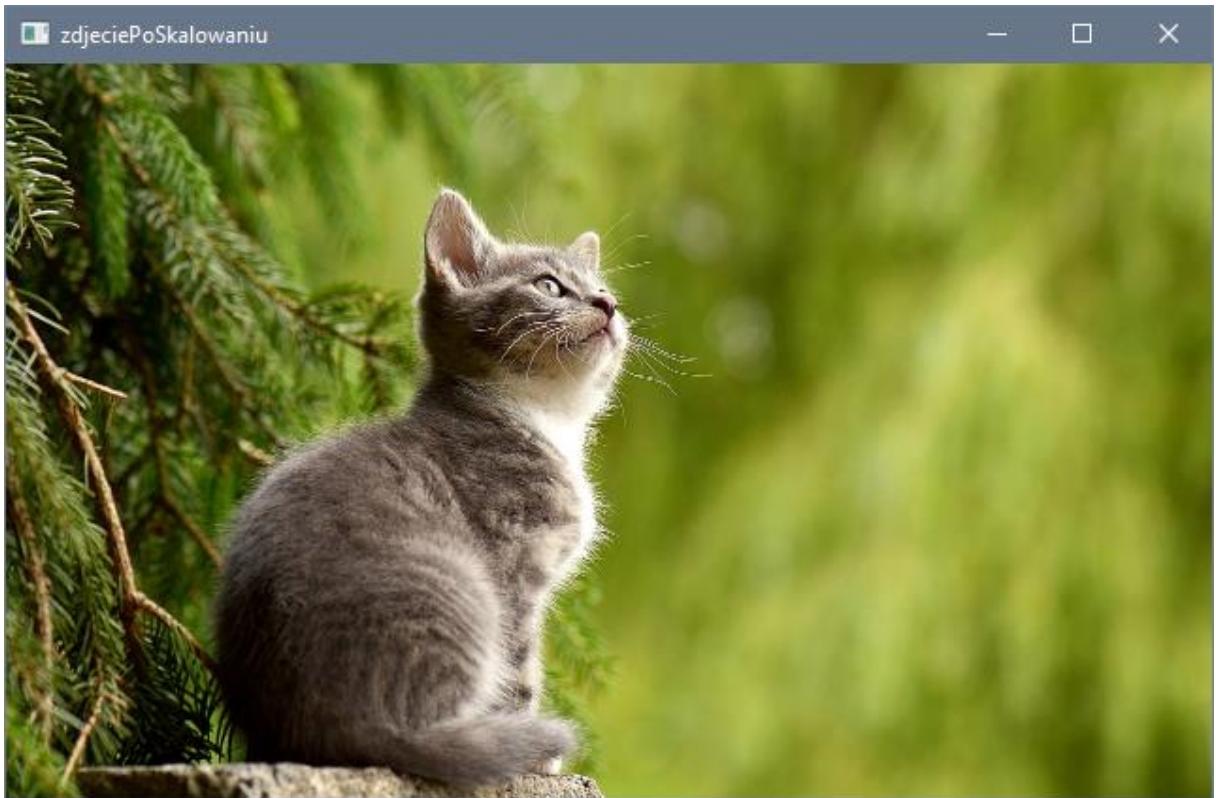


```

150 # ===== translacja z użyciem warpAffine() =====
151 zdjecie = cv2.imread(img_filepath, 1)
152 wysokosc, szerokosc, _ = zdjecie.shape
153 macierzTranslacji = np.float32([[1, 0, 50], [0, 1, 150]])
154 print(macierzTranslacji)
155 zdjeciePoTranslacji = cv2.warpAffine(zdjecie, macierzTranslacji, (wysokosc, szerokosc))
156 cv2.imshow("Oryginalne", zdjecie)
157 cv2.imshow("Nowy obraz", zdjeciePoTranslacji)
158 cv2.waitKey(0)
159 (wysokosc, szerokosc, _) = zdjecie.shape
160 macierzTranslacji = np.float32([[1, 0, 50], [0, 1, 150], [0, 0, 1]])
161 print(macierzTranslacji)
162 zdjeciePoTranslacji = cv2.warpPerspective(zdjecie, macierzTranslacji, (wysokosc, szerokosc))
163 cv2.imshow("Oryginalne", zdjecie)
164 cv2.imshow("Nowy obraz", zdjeciePoTranslacji)
165 cv2.waitKey(0)

```

## Skalowanie:

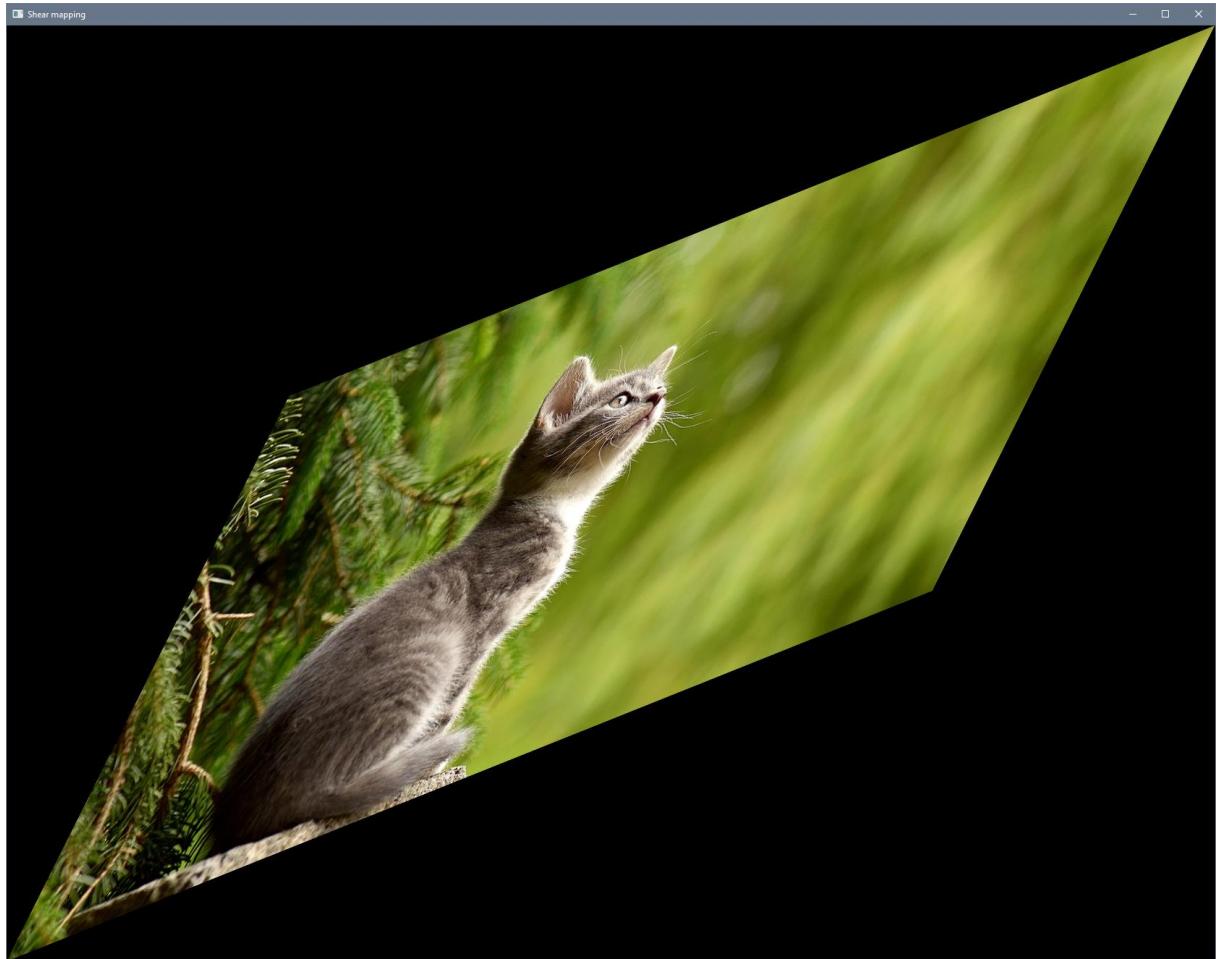


```

167 # ===== skalowanie obrazu =====
168 zdjecie = cv2.imread(img_filepath, 1)
169 wysokosc, szerokosc, _ = zdjecie.shape
170 zdjeciePoSkalowaniu = cv2.resize(zdjecie, (int(szerokosc / 2), int(wysokosc / 2)))
171 cv2.imshow("zdjeciePoSkalowaniu", zdjeciePoSkalowaniu)
172 cv2.waitKey(0)
173
174 zdjeciePoSkalowaniu = cv2.resize(zdjecie, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_CUBIC)
175 cv2.imshow("zdjeciePoSkalowaniu", zdjeciePoSkalowaniu)
176 cv2.waitKey(0)

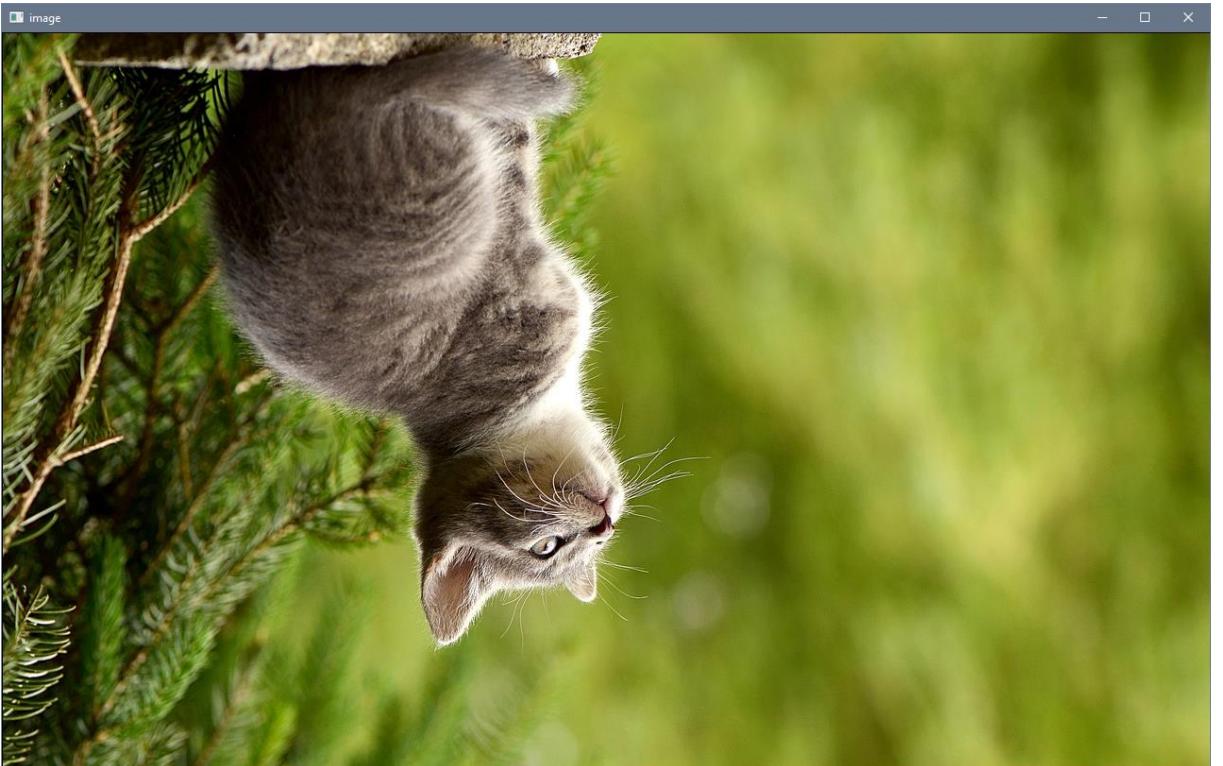
```

## Shear mapping:



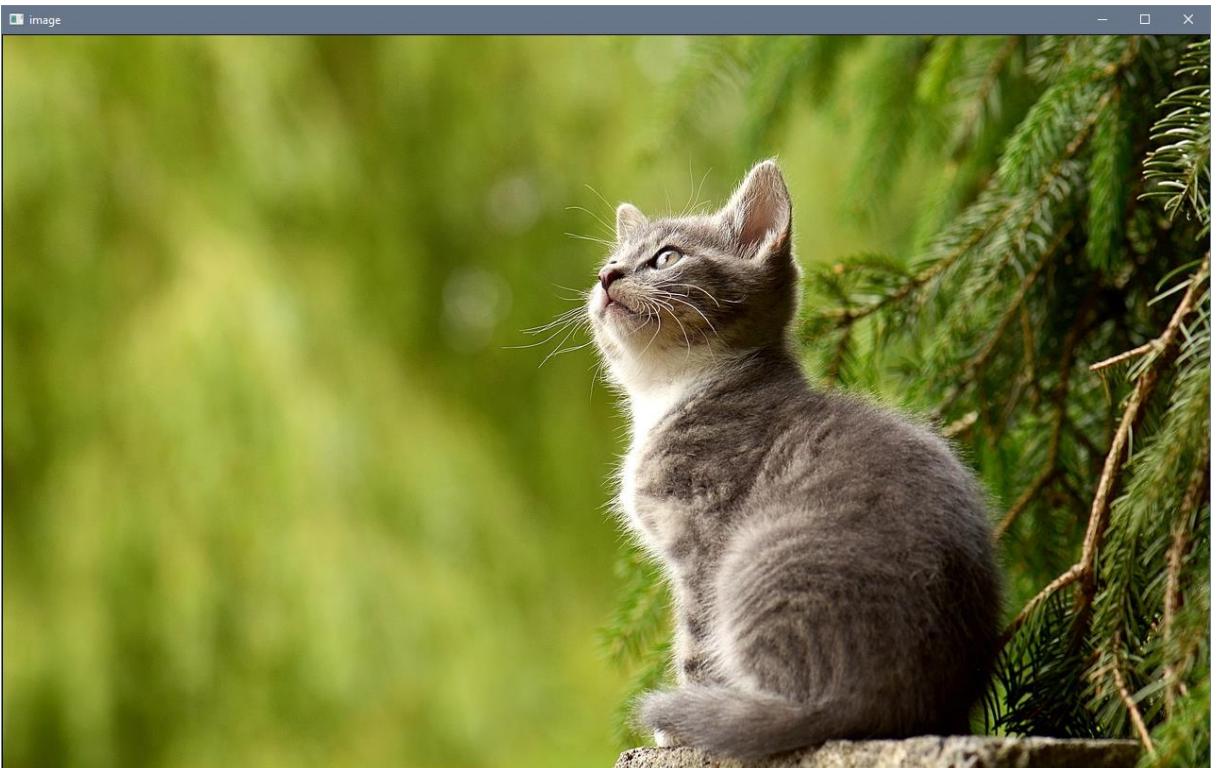
```
179 # ===== transformacja shear =====
180 zdjecie = cv2.imread(img_filepath, 1)
181 wysokosc, szerokosc, _ = zdjecie.shape
182 print(wysokosc, szerokosc)
183 macierzTransformacji = np.float32([[1, -0.5, 0], [-0.4, 1, 0], [0, 0, 1]])
184 LD = macierzTransformacji @ np.float32([0, 0, 1])
185 RU = macierzTransformacji @ np.float32([szerokosc, wysokosc, 1])
186 LU = macierzTransformacji @ np.float32([0, wysokosc, 1])
187 RD = macierzTransformacji @ np.float32([szerokosc, 0, 1])
188 wysokosc = abs(min([c[0] for c in [LD, RU, LU, RD]])) + abs(max([c[0] for c in [LD, RU, LU, RD]]))
189 szerokosc = abs(min([c[1] for c in [LD, RU, LU, RD]])) + abs(max([c[1] for c in [LD, RU, LU, RD]]))
190 dx = int(-1 * min([c[0] for c in [LD, RU, LU, RD]]))
191 dy = int(-1 * min([c[1] for c in [LD, RU, LU, RD]]))
192 macierzTransformacji[0][2] = dx
193 macierzTransformacji[1][2] = dy
194 zdjeciePoPochyleniu = cv2.warpPerspective(zdjecie, macierzTransformacji, (int(wysokosc + 0.5), int(szerokosc + 0.5)))
195 cv2.imshow("Shear mapping", zdjeciePoPochyleniu)
196 cv2.waitKey(0)
```

Odbicie względem osi X:



```
# Odbicie względem osi X: CTRL + LPM
if flags == cv2.EVENT_FLAG_CTRLKEY + cv2.EVENT_FLAG_LBUTTON:
    macierzTransformacjiX = np.float32([[1, 0, 0], [0, -1, wysokosc], [0, 0, 1]])
    img = cv2.warpPerspective(img, macierzTransformacjiX, (szerokosc, wysokosc))
```

Odbicie względem osi Y:



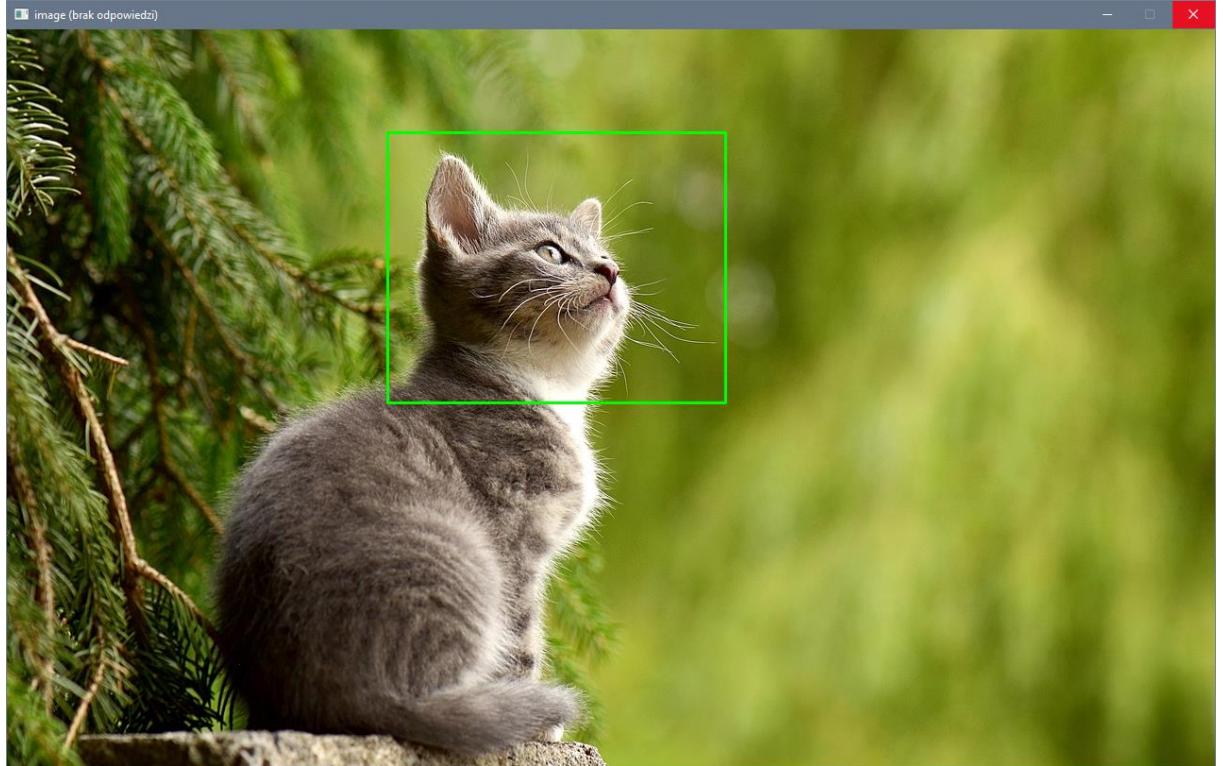
Skalowanie do wybranej przekątnej:



```
# Skalowanie do wybranej przekątnej: ALT + LPM, ALT + PPM
elif flags == cv2.EVENT_FLAG_ALTKEY + cv2.EVENT_FLAG_LBUTTON:
    points = [(x, y)]
elif flags == cv2.EVENT_FLAG_ALTKEY + cv2.EVENT_FLAG_RBUTTON:
    points.append((x, y))
width = max([k[0] for k in points]) - min([k[0] for k in points])
height = max([k[1] for k in points]) - min([k[1] for k in points])
if np.size(points, 0) == 2:
    img = cv2.resize(img, (width, height), interpolation=cv2.INTER_CUBIC)
```

## Skalowanie względem odcinka:

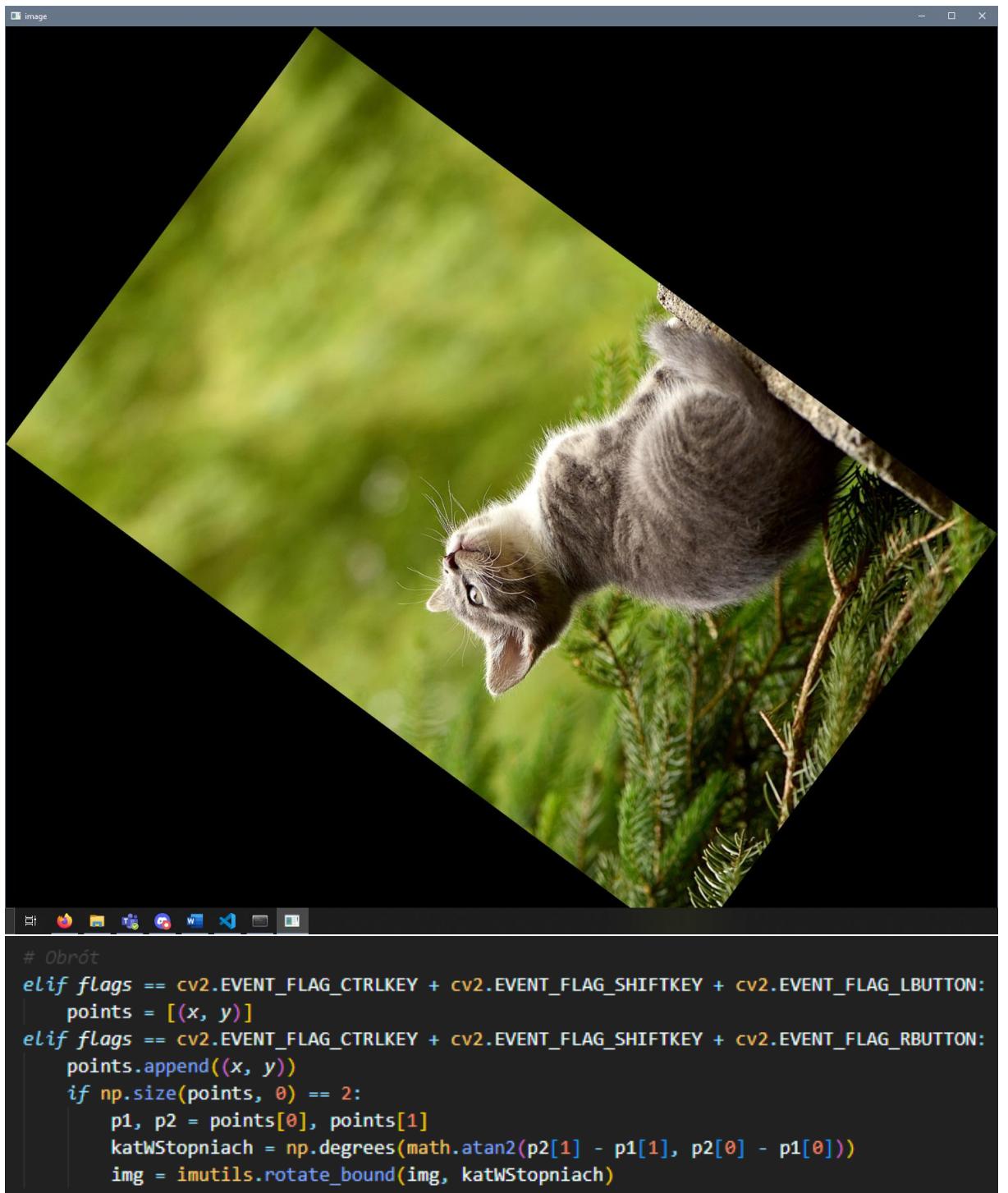
```
# Skalowanie względem odcinka
elif flags == cv2.EVENT_FLAG_SHIFTKEY + cv2.EVENT_FLAG_LBUTTON:
    points = [(x, y)]
    drawing = True
    s_x, s_y = x, y
elif event == cv2.EVENT_MOUSEMOVE:
    if drawing == True:
        img = img2.copy()
        cv2.rectangle(img, (s_x, s_y), (x, y), (0, 255, 0), 2)
elif flags == cv2.EVENT_FLAG_SHIFTKEY + cv2.EVENT_FLAG_RBUTTON:
    drawing = False
    cv2.rectangle(img, (s_x, s_y), (x, y), (0, 255, 0), 2)
    w = max(x, s_x) - min(x, s_x)
    h = max(y, s_y) - min(y, s_y)
    print(f"w: {w}, h: {h}")
    print("Podaj długość odcinka x (|| os X) [px] (bez zmian -> wpisz -1): ")
    odc_x = int(input())
    if odc_x >= 0:
        skala_fx = odc_x / w
    else:
        skala_fx = 1
    print(f"skala_fx: {skala_fx}")
    print("Podaj długość odcinka y (|| os Y) [px] (bez zmian -> wpisz -1): ")
    odc_y = int(input())
    if odc_y >= 0:
        skala_fy = odc_y / h
    else:
        skala_fy = 1
    print(f"skala_fy: {skala_fy}")
    img = img3.copy()
    img = cv2.resize(img, dsize=None, fx=skala_fx, fy=skala_fy, interpolation=cv2.INTER_CUBIC)
```



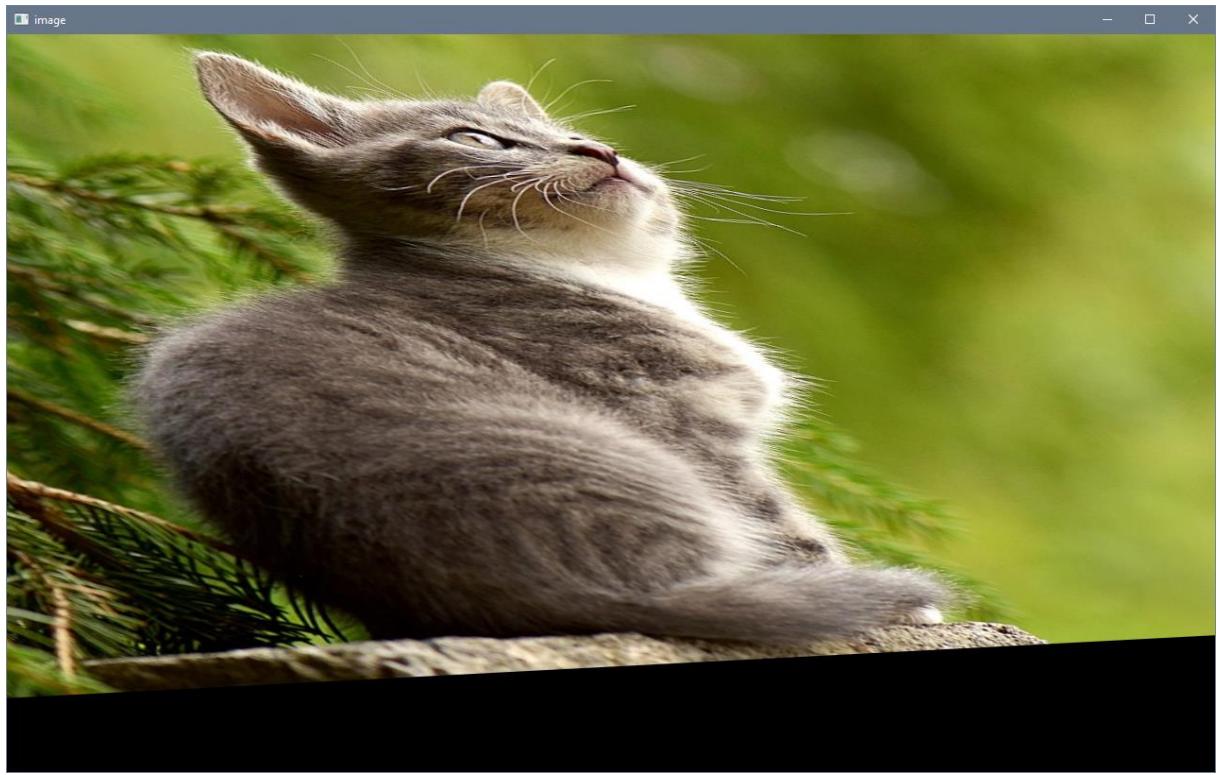
```
w: 358, h: 286
Podaj długość odcinka x (|| oś X) [px] (bez zmian -> wpisz -1):
700
skala_fx: 1.9553072625698324
Podaj długość odcinka y (|| oś Y) [px] (bez zmian -> wpisz -1):
-1
skala_fy: 1
```



Obrót:



Transformacja afiniczna:



```
# Transformacja afiniczna (najpierw wybieramy punkt LPM, potem dodajemy kolejne PPM)
elif flags == cv2.EVENT_FLAG_SHIFTKEY + cv2.EVENT_FLAG_ALTKEY + cv2.EVENT_FLAG_LBUTTON:
    points = [(x, y)]
    print(points)
elif flags == cv2.EVENT_FLAG_SHIFTKEY + cv2.EVENT_FLAG_ALTKEY + cv2.EVENT_FLAG_RBUTTON:
    points.append((x, y))
    print(points)
if np.size(points, 0) == 3:
    print("podaj 3 punkty referencyjne(?) (x1,y1,x2,y2,x3,y3):")
    p1x = int(input())
    p1y = int(input())
    p2x = int(input())
    p2y = int(input())
    p3x = int(input())
    p3y = int(input())
    print(p1x, p1y, p2x, p2y, p3x, p3y)
    punkty1 = np.float32([points[0], points[1], points[2]])
    punkty2 = np.float32([[p1x, p1y], [p2x, p2y], [p3x, p3y]])
    macierzTransformacji = cv2.getAffineTransform(punkty1, punkty2)
    img = cv2.warpAffine(img, macierzTransformacji, (szerokosc, wysokosc))
```

```
[(365, 105)]
[(365, 105), (900, 155)]
[(365, 105), (900, 155), (651, 636)]
podaj 3 punkty referencyjne(?) (x1,y1,x2,y2,x3,y3):
0
0
1000
0
1000
500
0 0 1000 0 1000 500
```

## Transformacja perspektywiczna:



```
# Transformacja perspektywiczna (najpierw wybieramy punkt LPM, potem dodajemy kolejne PPM)
elif flags == cv2.EVENT_FLAG_CTRLKEY + cv2.EVENT_FLAG_ALTKEY + cv2.EVENT_FLAG_LBUTTON:
    points = [(x, y)]
    print(points)
elif flags == cv2.EVENT_FLAG_CTRLKEY + cv2.EVENT_FLAG_ALTKEY + cv2.EVENT_FLAG_RBUTTON:
    points.append((x, y))
    print(points)
if np.size(points, 0) == 4:
    print("podaj 4 punkty referencyjne(?) (x1,y1,x2,y2,x3,y3,x4,y4):")
    p1x = int(input())
    p1y = int(input())
    p2x = int(input())
    p2y = int(input())
    p3x = int(input())
    p3y = int(input())
    p4x = int(input())
    p4y = int(input())
    print(p1x, p1y, p2x, p2y, p3x, p3y, p4x, p4y)
    punkty1 = np.float32([points[0], points[1], points[2], points[3]])
    punkty2 = np.float32([[p1x, p1y], [p2x, p2y], [p3x, p3y], [p4x, p4y]])
    macierzTransformacji = cv2.getPerspectiveTransform(punkty1, punkty2)
    img = cv2.warpPerspective(img, macierzTransformacji, (szerokosc, wysokosc))
    # ^ szerokosc, wysokosc trzeba obliczyc by pokazalo caly obraz...
cv2.imshow("image", img)
```

```
[(433, 111)]
[(433, 111), (727, 144)]
[(433, 111), (727, 144), (682, 761)]
[(433, 111), (727, 144), (682, 761), (70, 720)]
podaj 4 punkty referencyjne(?) (x1,y1,x2,y2,x3,y3,x4,y4):
0
0
1000
0
1000
1000
0
1000
0 0 1000 0 1000 1000 0 1000
```

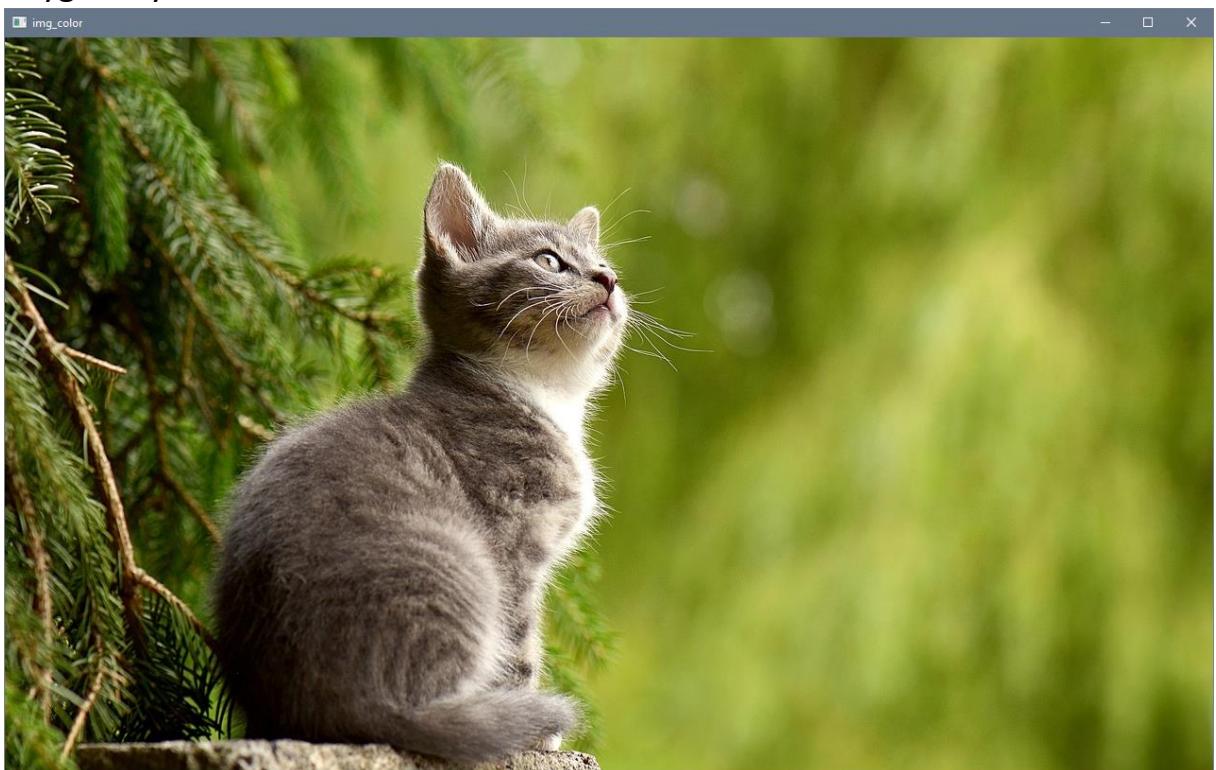
#### 4. Analiza poprawy histogramów metoda globalną oraz CLAHE

- Kod w folderze p4
- Plik **main\_p4\_histogarmy\_binaryzacja.py**

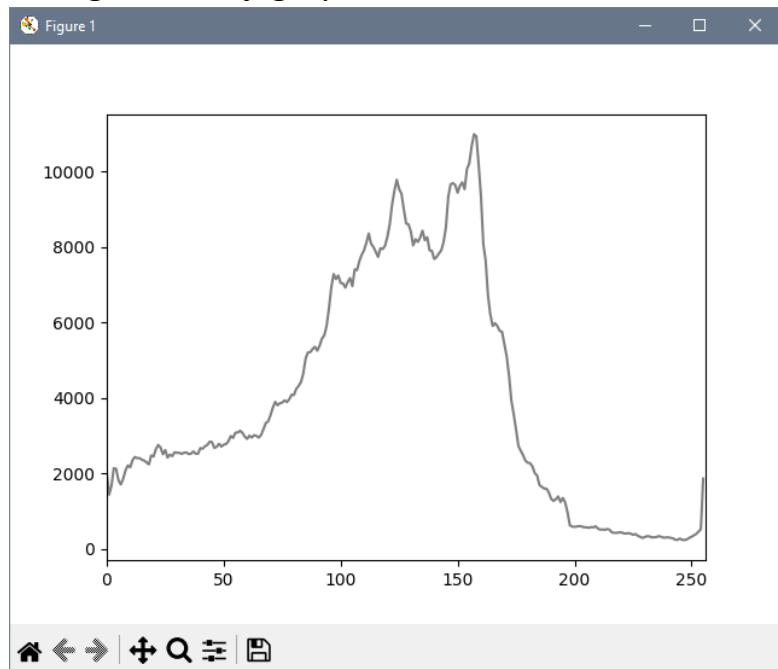
Jednym z podstawowych zadań przetwarzania obrazów jest ich poprawa poprzez analizę i przetwarzanie histogramów. Przetwarzanie (modelowanie) histogramu polega na statystycznej analizie obrazu. W celu wyświetlenia histogramu przy wykorzystaniu biblioteki opencv wykorzystywana jest następująca funkcja: cv2.calcHist(). Zastosowanie korekcji histogramu umożliwia poprawę jakości obrazu np. gdy jego fragmenty są niedoświetlone lub prześwietlone. Złe warunki oświetleniowe wpływają na niepełne wykorzystanie wszystkich poziomów stopnia szarości z przedziału 0 - 255. Korekcja histogramu polega na takim przekształceniu jego elementów, aby wynikowy histogram był płaski i równomiernie wypełniał cały zakres jasności. Wyrównanie histogramu obrazu ma na celu poprawę globalnego kontrastu , szczególnie w przypadku gdy wartości stopni szarości są reprezentowane przez wartości z niewielkiego obszaru. Dzięki wyrównaniu histogramu czyli jego „rozciągnięciu” na większy zakres możliwe jest zwiększenie kontrastu na niskich poziomach. Wyrównanie histogramu jest dobre, gdy histogram obrazu jest ograniczony do określonego regionu. Nie będzie działać dobrze w miejscach, gdzie występują duże różnice w intensywności, gdzie histogram obejmuje duży obszar, tzn. występują zarówno jasne, jak i ciemne piksele. Chociaż globalne wyrównywanie histogramu może być użyteczne, w przypadku niektórych obrazów korzystniejsze może być zastosować różne rodzaje wyrównania w różnych regionach. Metodą wartą uwagi jest adaptacyjne wyrównywanie histogramów (Contrast Limited Adaptive Histogram Equalization – CLAHE). Obraz jest dzielony na małe bloki zwane "kafelkami" (domyślny rozmiar 8x8). Następnie, w każdym z bloków wyznaczana jest lokalna funkcja poprawy kontrastu, która definiuje nową wartość barwy dla centralnego piksela wewnątrz bloku. Tak więc na małym obszarze, histogram będzie ograniczony do małego obszaru (chyba, że jest tam szum). Jeśli jest tam szum, zostanie on wzmacniony. Aby tego uniknąć, stosowane jest ograniczenie kontrastu. Jeśli któryś z przedziałów histogramu jest powyżej określonej granicy kontrastu

(domyślnie 40 w OpenCV), piksele te są przycinane i rozdzielane równomiernie do innych przedziałów przed zastosowaniem wyrównania histogramu. W CLAHE wzmacnienie kontrastu w sąsiedztwie danej wartości piksela jest dane przez nachylenie funkcji transformacji. Jest ono proporcjonalne do nachylenia funkcji skumulowanego rozkładu gęstości funkcji prawdopodobieństwa (CDF), a zatem do wartości histogramu przy tej wartości piksela. CLAHE ogranicza wzmacnienie poprzez obcięcie histogramu przy predefiniowanej wartości przed obliczeniem CDF. Ogranicza to nachylenie CDF, a tym samym funkcji transformacji. Zazwyczaj przyjmuje się te wartości z przedziału 3 a 4. Po wyrównaniu, aby usunąć artefakty na granicach kafelków, stosowana jest interpolacja bilinearna.

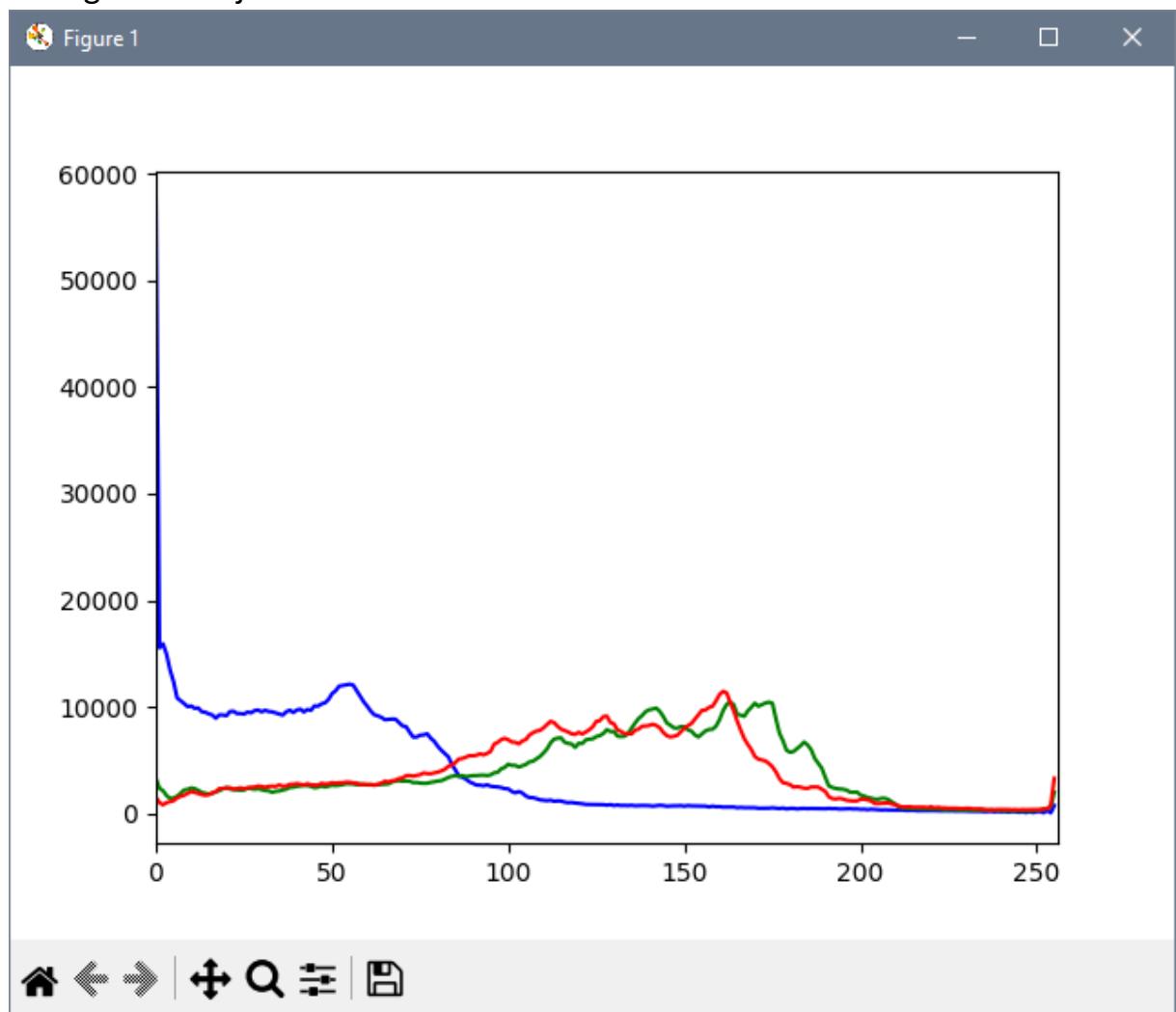
Oryginalny obraz:



Histogram wersji greyscale:



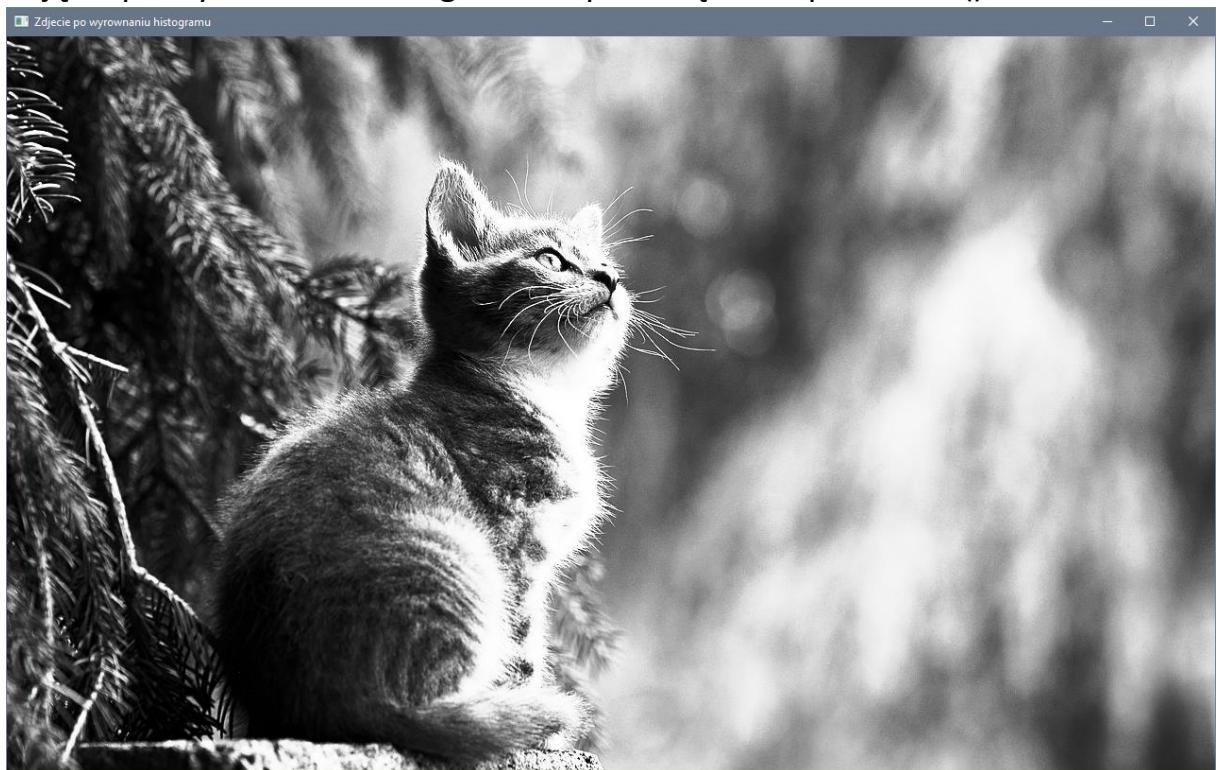
Histogram wersji RGB:



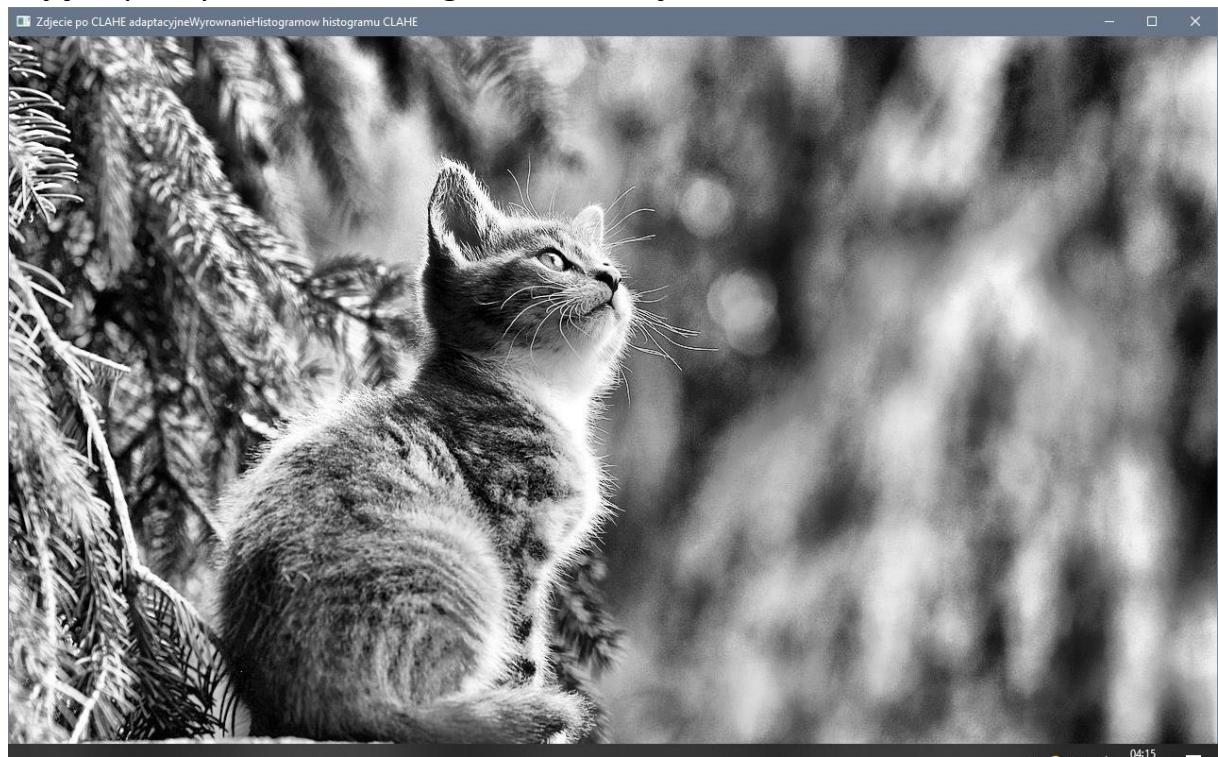
Oryginalne zdjęcie w skali szarości:



Zdjęcie po wyrownaniu histogramu za pomocą cv2.equalizeHist():



## Zdjęcie po wyrównaniu histogramu metodą CLAHE:



```
## Histogram adaptacyjny CLAHE
claheAlgorytm = cv2.createCLAHE(clipLimit=4.0, tileSize=(8, 8))
adaptacyjneWyrownanieHistogramow = claheAlgorytm.apply(img_grey)
cv2.imshow("Oryginalne zdjecie", img_grey) # or show image in two separate windows
cv2.imshow(
    "Zdjecie po CLAHE adaptacyjneWyrownanieHistogramow histogramu CLAHE",
    adaptacyjneWyrownanieHistogramow,
)
cv2.imshow("Zdjecie po wyrownaniu histogramu", equ)
cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
    cv2.waitKey(1)
```

## 5. Analiza doboru parametrów binaryzacji

- Kod w folderze **p4**
- Plik **main\_p4\_histogarmy\_binaryzacja.py**

Binaryzacja polega na zmianie obrazów RGB lub w stopniach szarości na obrazy binarne (zero-jedynkowe). To przekształcenie jest wykorzystywane w wielu zagadnieniach związanych m.in.:

- określeniem liczebności elementów na obrazie,
- pomiarem pola powierzchni, obwodu czy długości linii,
- automatyczną wektoryzacją,
- analizą i modyfikacją kształtu obiektów,
- klasyfikacją lub w detekcji punktów charakterystycznych, wykorzystywanych w procesie orientacji zdjęć.

Biblioteka OpenCV umożliwia wykonanie binaryzacji na dwa sposoby tj. prostej binaryzacji uwzględniającej globalny próg lub metody adaptacyjnej bazującej na obszarach otaczających przetwarzany piksel. Prosta (globalna) binaryzacja polega na wykorzystaniu przetworzeniu każdego z pikseli przy wykorzystaniu tej samej wartości progowej. Jeśli wartość piksela jest mniejsza od progu, to nowa wartość piksela ustawiana jest na 0, w przeciwnym razie ustawiana jest na wartość maksymalną. Do prostej binaryzacji służy funkcja cv2.threshold().

Podstawowe progowanie odbywa się przy użyciu typu cv2.THRESH\_BINARY. Wszystkie proste typy progowania to:

THRESH\_BINARY

Python:

cv2.THRESH\_BINARY

$$\text{wynik}(x, y) = \begin{cases} \text{maksymala}_{\text{wartość}} & \text{if } \text{zdjęcie}(x, y) > \text{próg} \\ 0 & \text{else} \end{cases}$$

THRESH\_BINARY\_INV

Python:

cv2.THRESH\_BINARY\_INV

$$\text{wynik}(x, y) = \begin{cases} 0 & \text{if } \text{zdjęcie}(x, y) > \text{próg} \\ \text{maksymala}_{\text{wartość}} & \text{else} \end{cases}$$

THRESH\_TRUNC

Python:

cv2.THRESH\_TRUNC

$$\text{wynik}(x, y) = \begin{cases} \text{próg} & \text{if } \text{zdjęcie}(x, y) > \text{próg} \\ \text{zdjęcie}(x, y,) & \text{else} \end{cases}$$

THRESH\_TOZERO

Python:

cv2.THRESH\_TOZERO

$$\text{wynik}(x, y) = \begin{cases} \text{zdjęcie}(x, y,) & \text{if } \text{zdjęcie}(x, y,) > \text{próg} \\ 0 & \text{else} \end{cases}$$

THRESH\_TOZERO\_INV

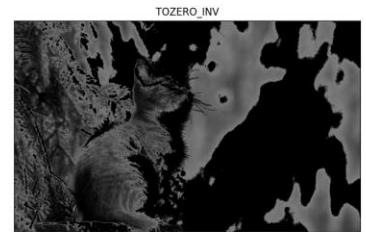
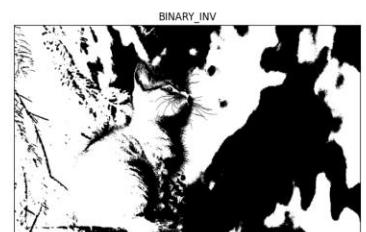
Python:

cv2.THRESH\_TOZERO\_INV

$$\text{wynik}(x, y) = \begin{cases} 0 & \text{if } \text{zdjęcie}(x, y,) > \text{próg} \\ \text{zdjęcie}(x, y,) & \text{else} \end{cases}$$

Podobnie jak w przypadku wyrównania histogramów, zastosowanie globalnego progowania nie przyniesie oczekiwanych efektów w przypadku występowania np. znaczących zmian oświetlenia w różnych fragmentach obrazów. Z tego też powodu niezbędne jest wykorzystanie metod lokalnej binaryzacji nazywanej również adaptive thresholding, w której wartość progu jest automatycznie wyznaczana w oparciu zdefiniowany obszar wokół przetwarzanego punktu. Otrzymujemy więc różne progi dla różnych regionów tego samego obrazu, co daje lepsze rezultaty dla obrazów o zmiennym oświetleniu. Do adaptive thresholding służy funkcja cv2.adaptiveThreshold(). W zależności od sposobu zdefiniowania progu doboru pikseli wykorzystywanych w procesie zamiany na postać binarną obrazów cyfrowych rozróżniamy m.in.:  
- binaryzację z dolnym progiem, dla której wartości intensywności mniejsze od założonego progu zostają zmienione na czarny (0), a reszta na kolor biały (1);  
- binaryzacja z górnym progiem, dla której wartości intensywności większe od założonego progu zostają zmienione na czarny (0), a reszta na kolor biały (1);  
- binaryzacja z podwójny progiem, która bazuje na dwóch progach - dolnym i górnym. Przyjmuje się, że piksele, które mieszczą się w zdefiniowanym zakresie, przyjmują kolor biały (1), a pozostałe czarny (0).

Różne typy prostej binaryzacji w opencv:



## Adaptive Mean Thresholding:



Różne progi prostej binaryzacji (25%, 50% i 75% maksymalnej wartości):



```
imgArray = np.asarray(img_grey)
ret1, imgThresholded1 = cv2.threshold(imgArray, 64, 255, cv2.THRESH_BINARY)
ret2, imgThresholded2 = cv2.threshold(imgArray, 127, 255, cv2.THRESH_BINARY)
ret3, imgThresholded3 = cv2.threshold(imgArray, 191, 255, cv2.THRESH_BINARY)
plt.figure(1, figsize=(15, 10))
plt.subplot(131)
plt.title("Próg 64")
plt.axis("off")
plt.imshow(imgThresholded1, cmap="gray")
plt.subplot(132)
plt.title("Próg 127")
plt.axis("off")
plt.imshow(imgThresholded2, cmap="gray")
plt.subplot(133)
plt.title("Próg 191")
plt.axis("off")
plt.imshow(imgThresholded3, cmap="gray")
plt.show()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
```

## 6. Analiza działania filtrów dolno- i górnoprzepustowych

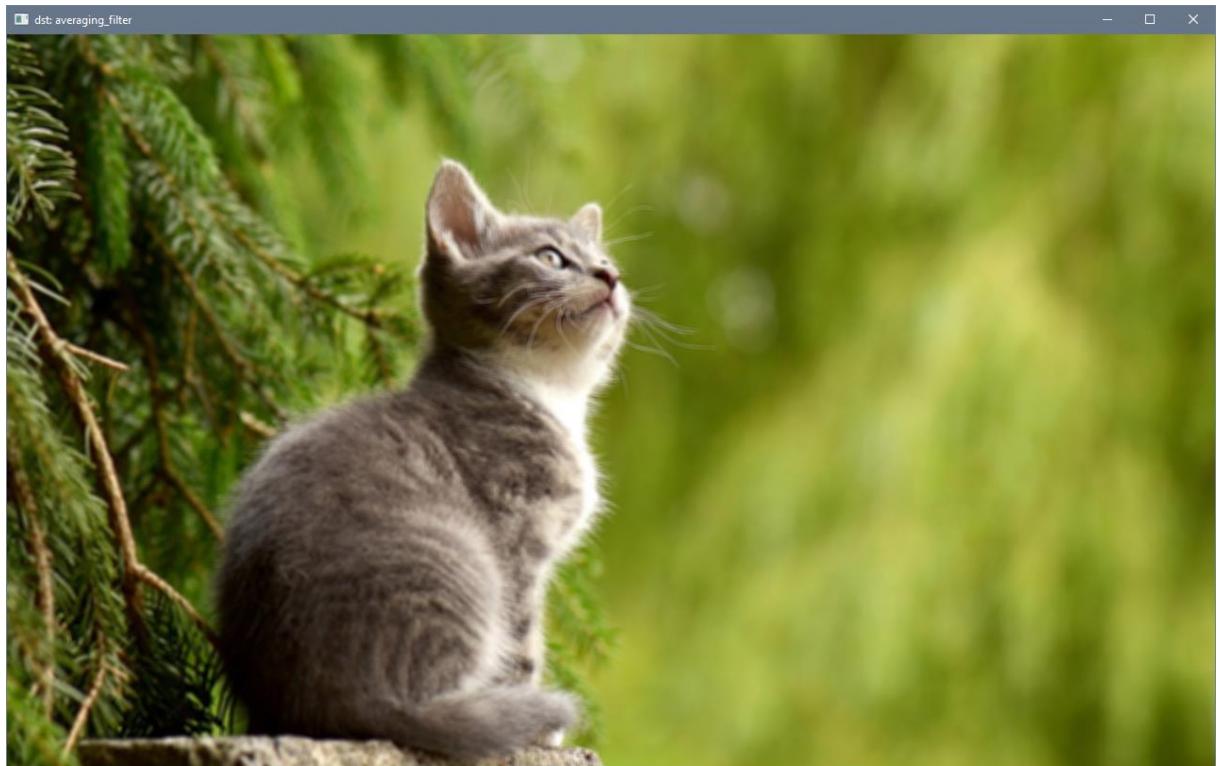
- Kod w folderze **p5**
- Plik **main\_p5\_filtry.py**

Operacje kontekstualne różnią się od operacji bezkontekstowych (inaczej: punktowych) tym, że wynikowa wartość piksela zależy nie tylko od źródłowej wartości poszczególnych pikseli, ale również od wartości otaczających je pikseli. Tego typu operacje znajdują zastosowanie wszędzie tam, gdzie istotna jest analiza nie tylko spektralnych, ale także (a może przede wszystkim) przestrzennych cech obrazu, takich jak sąsiedztwo, kształt, tekstura, rozmiar obiektu itp. Do najczęstszych zastosowań operacji kontekstualnych zaliczamy usuwanie (tłumienie) szumu na obrazie oraz wykrywanie wybranych cech obrazu – jego charakterystycznych elementów, np. krawędzi lub narożników obiektów. Istnieje przynajmniej kilka kryteriów klasyfikacji operacji kontekstualnych. Z punktu widzenia potencjalnych zastosowań, najważniejszy jest podział na operacje dolnoprzepustowe (ang. low-pass) i górnoprzepustowe (ang. high-pass). Dolno- i górnoprzepustowość odnosi się do częstotliwości przestrzennej (ang. spatial frequency) obrazu. Według definicji PWN, częstotliwość przestrzenna to liczba cykli jasność-ciemność we wzorcu na danym wymiarze odległości w przestrzeni wzrokowej. Z punktu widzenia cyfrowych przetworzeń obrazów cyfrowych, możemy ją wyjaśnić jako zróżnicowanie wartości pikseli. Zatem dużą częstotliwość przestrzenną zaobserwujemy na obrazie o dużym zróżnicowaniu wartości pikseli, natomiast małą – na obrazie o małym zróżnicowaniu wartości pikseli. Operacje dolnoprzepustowe obniżają więc częstotliwość przestrzenną obrazu, zmniejszając różnice między wartościami pikseli położonych blisko siebie, tym bardziej, im większe są te różnice. Natomiast operacje górnoprzepustowe zwiększają częstotliwość przestrzenną lub wykrywają te elementy obrazu, które cechują się dużą częstotliwością przestrzenną (np. krawędzie obiektów). Z tego względu – usuwania, czy też odsiewania wybranych elementów obrazu – operacje kontekstualne nazywamy często filtrami. Choć warto pamiętać, że to określenie nie zawsze jest adekwatne.

Filtry rozmywające to filtry dolnoprzepustowe uśredniające wartości otoczenia znajdującego się w zasięgu maski filtru. Ideą takiego filtru jest zastąpienie wartości każdego punktu w obrazie przez średnią intensywność jego sąsiedztwa zdefiniowanego przez maskę filtru. Filtr taki tłumia sygnały składowe widma o wysokiej częstotliwości a pozostawia bez zmian niskie częstotliwości. Najbardziej oczywistym zastosowaniem takiego filtru jest redukcja przypadkowego szumu i wygładzenie obrazu. Jednakże, krawędzie, które są prawie zawsze pożądaną informacją w obrazie również mają ostre przejścia jasności, zatem jako obszar o wysokiej częstotliwości ulegną tłumieniu przez filtr, który spowoduje ich rozmycie. Jest to niewątpliwie niepożądany efekt działania filtru i zarazem jego największa wada.

Filtr uśredniający to najprostszy filtr rozmywający o rozmiarze  $3 \times 3$  i współczynnikach równych 1 i wagę filtru równej 9 w wyniku działania zastępuje jasność punktu aktualnie przetwarzanego średnią arytmetyczną ze swojego 9-elementowego otoczenia. Tego typu filtr jedynkowy nazywa się filtrem uśredniającym lub pudełkowym (ang. box filter). Jeszcze lepszy efekt można osiągnąć aproksymując wartości maski filtru funkcją rozkładu Gaussa w dwuwymiarowej formie. Jednym z najbardziej popularnych filtrów statystycznych jest filtr medianowy. Obliczenie wartości wynikowej przetworzenia odbywa się poprzez uszeregowanie wartości pikseli objętych maską filtru w porządku rosnącym, a następnie wybranie wartości środkowej (median). Kolejnymi przykładami statystycznych filtrów dolnoprzepustowych są filtr minimalny i maksymalny. Są one dualne względem siebie. Jak łatwo się domyślić, wynik filtru minimalnego stanowi najmniejsza wartość spośród pikseli objętych maską, natomiast wynik filtru maksymalnego – największa wartość. Filtr minimalny jest często nazywany filtrem kompresującym albo erozjnym ponieważ efektem jego działania jest zmniejszenie globalnej jasności obrazu, jasne obiekty ulegną zmniejszeniu a powiększą się obiekty ciemne. Filtr maksymalny bywa nazywany filtrem dekompresującym albo ekspansywnym gdyż w wyniku filtracji zwiększa globalnie jasność obrazu i analogicznie do filtru minimalnego tu powiększone zostaną obiekty jasne a zmniejszone ciemne.

Obraz po zastosowaniu filtru uśredniającego (kernel 5x5):

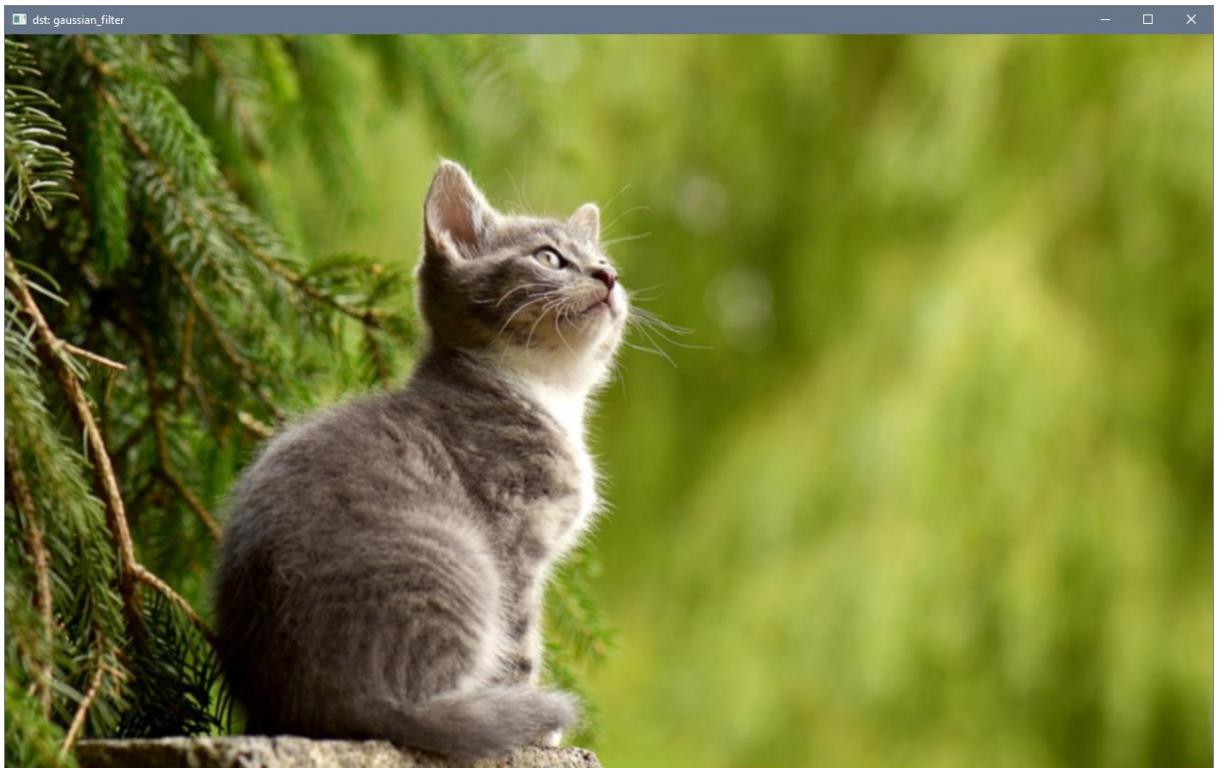


```
def display(func): # eksperyment, don't ask...
    def wrapper(*args, **kwargs):
        keys = func.__code__.co_varnames[: func.__code__.co_argcount][::-1]
        sorter = {j: i for i, j in enumerate(keys[::-1])}
        values = func.__defaults_[::-1]
        kwa = {i: j for i, j in zip(keys, values)}
        sorted_default_kwargs = {i: kwa[i] for i in sorted(kwa.keys(), key=sorter.get)}
        img = args[0]
        retv = func(*args, **kwargs)
        domyslny = sorted_default_kwargs["display"]
        podany = kwargs["display"] if "display" in kwargs.keys() else "cokur2"
        if podany != "cokur2" and domyslny != podany:
            if not podany:
                return retv
            elif not domyslny:
                return retv
            cv2.imshow("original image", img)
            cv2.imshow(f"dst: {func.__name__}", retv)
            print(cv2.waitKey(0))
            cv2.destroyAllWindows()
        return retv

    return wrapper

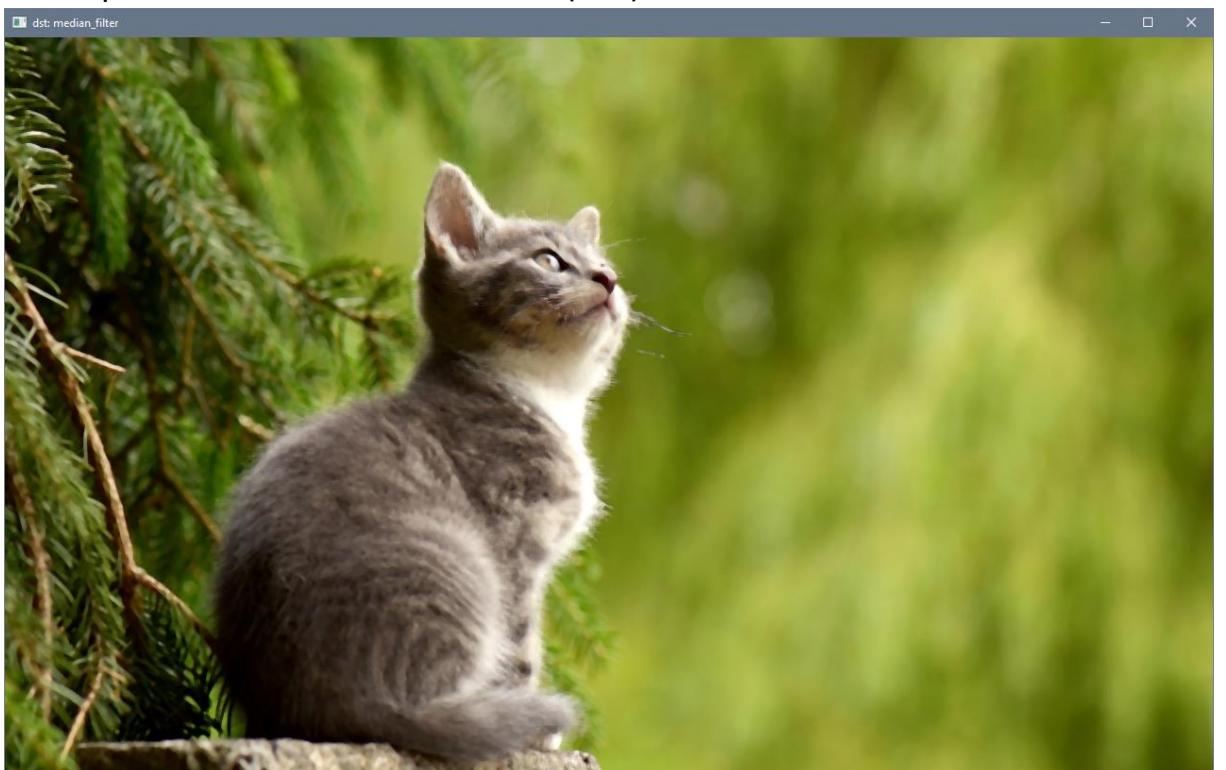
@display
def averaging_filter(img, shape=(5, 5), display=True):
    """Averaging (box) filter"""
    return cv2.filter2D(img, ddepth=-1, kernel=np.ones(shape, np.float32) / (shape[0] * shape[1]))
```

Obraz po zastosowaniu filtra Gaussa (kernel 5x5):

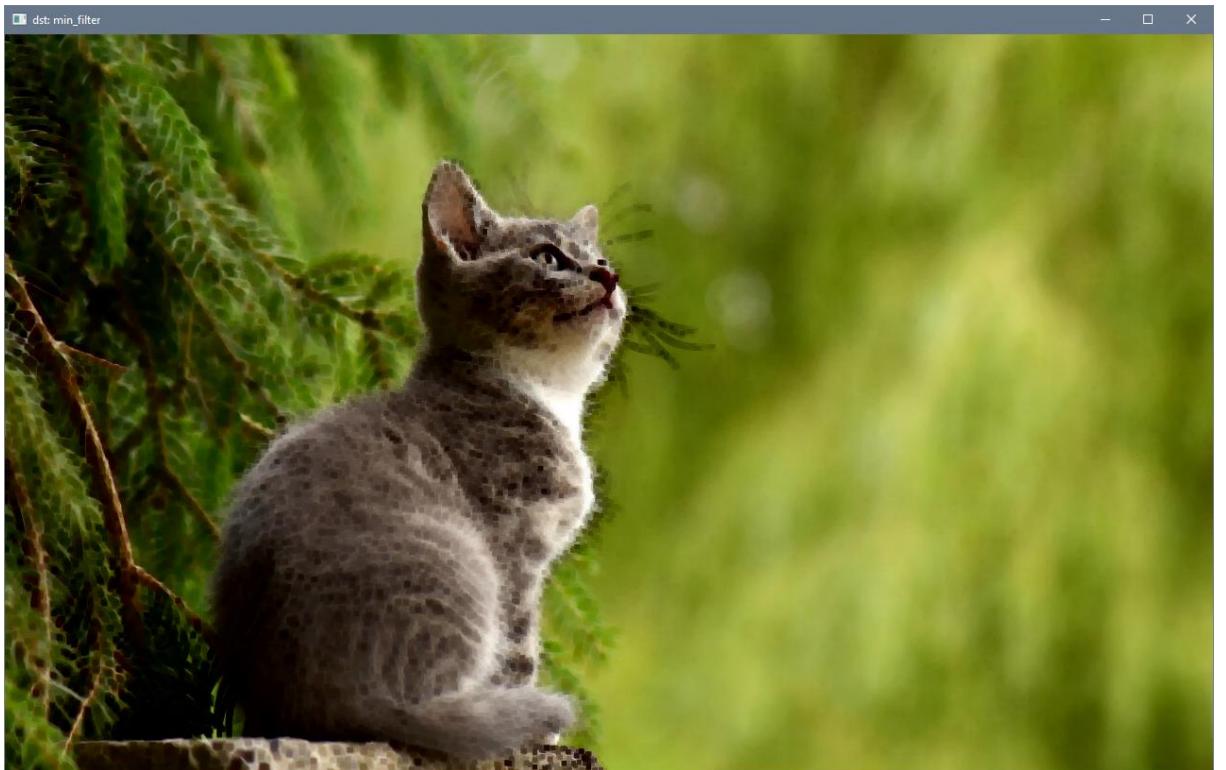


```
@display
def gaussian_filter(img, shape=(5, 5), display=True):
    """Gaussian filter"""
    return cv2.GaussianBlur(img, shape, sigmaX=0)
```

Obraz po zastosowaniu median filter (5x5):

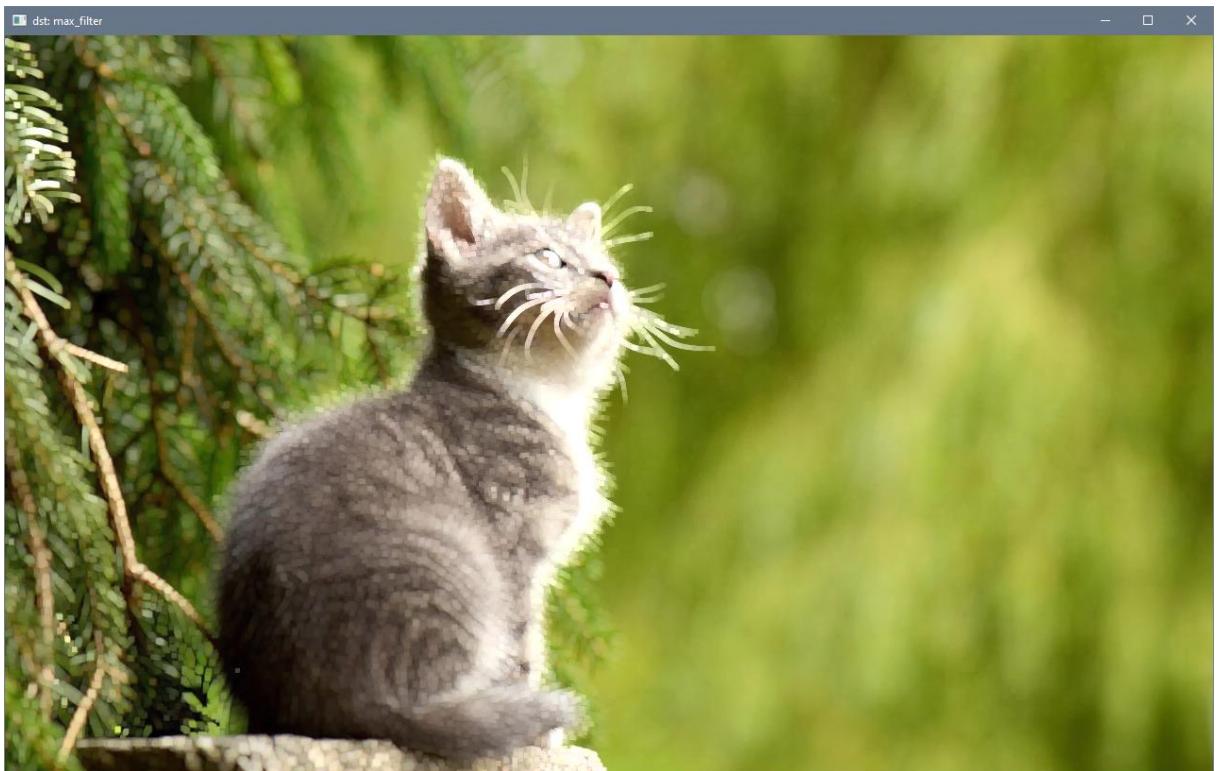


Obraz po zastosowaniu min filter (5x5):



```
@display
def min_filter(img, size=(5, 5), iterations=1, display=True):
    """Min filter - morphological erosion"""
    shape = cv2.MORPH_RECT
    return cv2.erode(img, kernel=cv2.getStructuringElement(shape=shape, ksize=size), iterations=iterations)
```

Obraz po zastosowaniu max filter (5x5):



Filtры wyostrzające znacznie różnią się od poprzednich trzech rodzajów operacji. Celem filtrów wyostrzających jest bowiem wyostrzenie – pozorne – obrazu, podczas gdy pozostałe operacje mają za zadanie wskazanie miejsc o dużych częstotliwościach przestrzennych – krawędzi i innych istotnych szczegółów obrazu. Istnieje kilka rodzajów operacji górnoprzepustowych, w zależności od sposobu realizacji operacji lub od jej celu. Możemy wyróżnić m.in.:

- operatory kierunkowe;
- filtry gradientowe – wykrywające krawędzie;
- laplasjany;
- filtry wyostrzające.

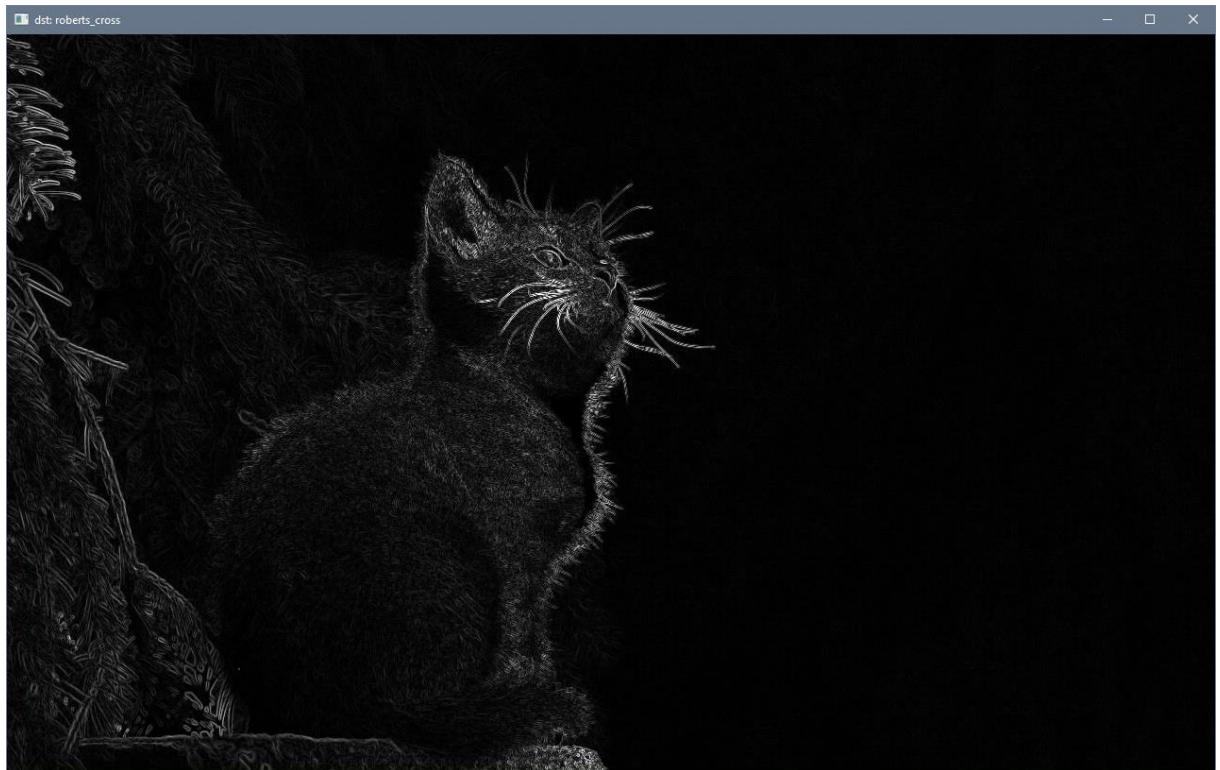
Operatory kierunkowe nazywamy tak dlatego, że wykrywają krawędzie obiektów z różnym efektem, w zależności od orientacji krawędzi. Jednym z najprostszych operatorów kierunkowych jest operator Robertsa. Innym przykładem jest operator Prewitta. Natomiast najpopularniejszym filtrem gradientowym jest filtr Sobela. Filtr Sobela jest operacją złożoną. Polega na wykonaniu – niezależnie – przetworzenia obrazu źródłowego za pomocą dwóch operatorów Sobela: SobelS i SobelE. Następnie dwa obrazy wynikowe (nazwijmy je, odpowiednio SX oraz SY ) są przetwarzane według formuły: pierwiastek sumy kwadratów SX i SY . Zasada działania filtru opiera się na połączeniu efektów działania obydwu operatorów, z których każdy wykrywa krawędzie o innych orientacjach – z innym natężeniem. Przeznaczeniem filtrów Laplace'a, czyli laplasjanów jest wykrywanie krawędzi – niezależnie od kierunku. Laplasjany z maską o rozmiarze 1 można przedstawić w jeden z dwóch sposobów:

0	-1	0
-1	4	-1
-1	-1	0

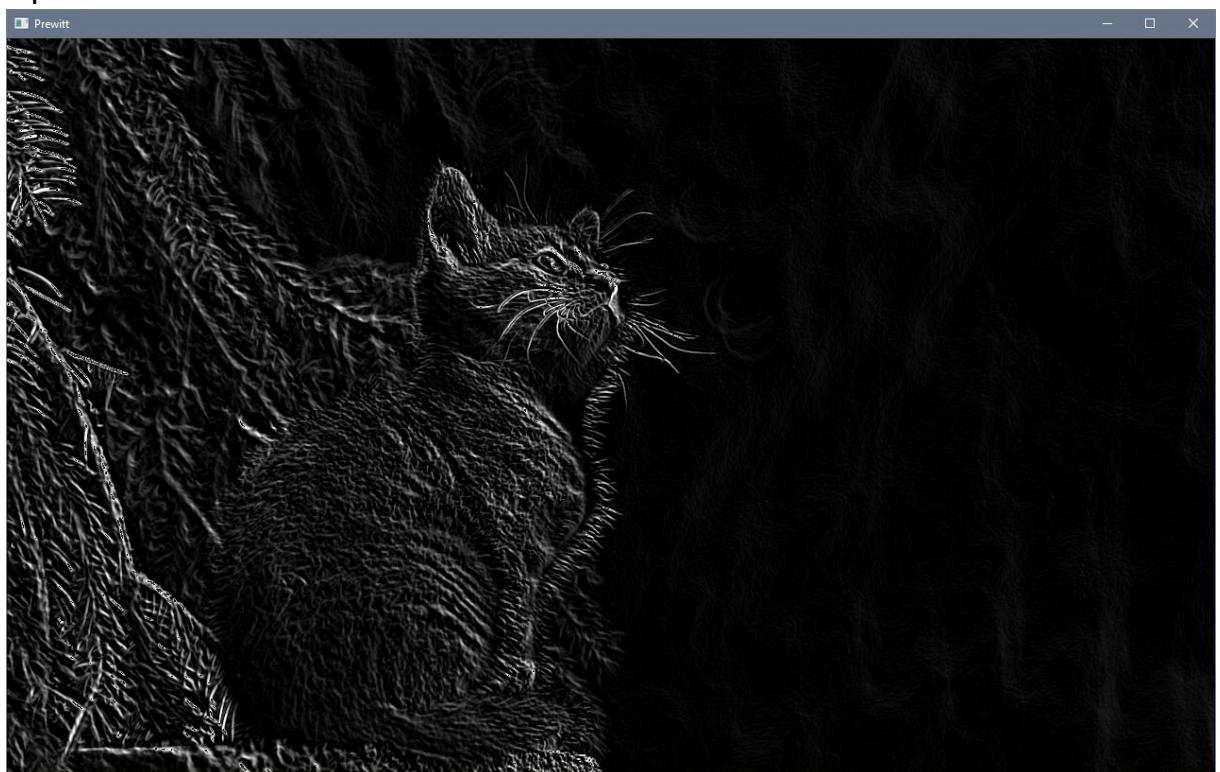
-1	-1	-1
-1	8	-1
-1	-1	-1

Biblioteka opencv udostępnia wiele funkcji do filtrowania obrazów, np.: cv2.Laplacian(), cv2.sobel(), cv2.GaussianBlur(), cv2.filter2D().

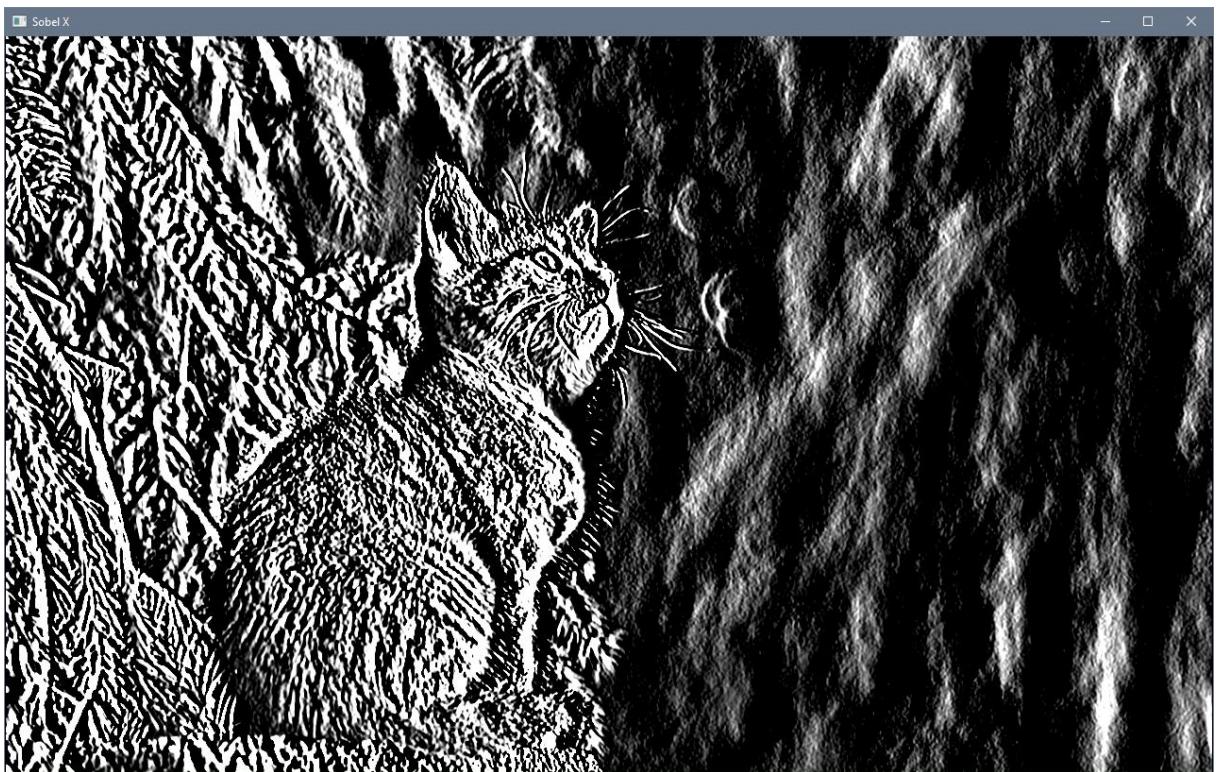
Robert's cross:



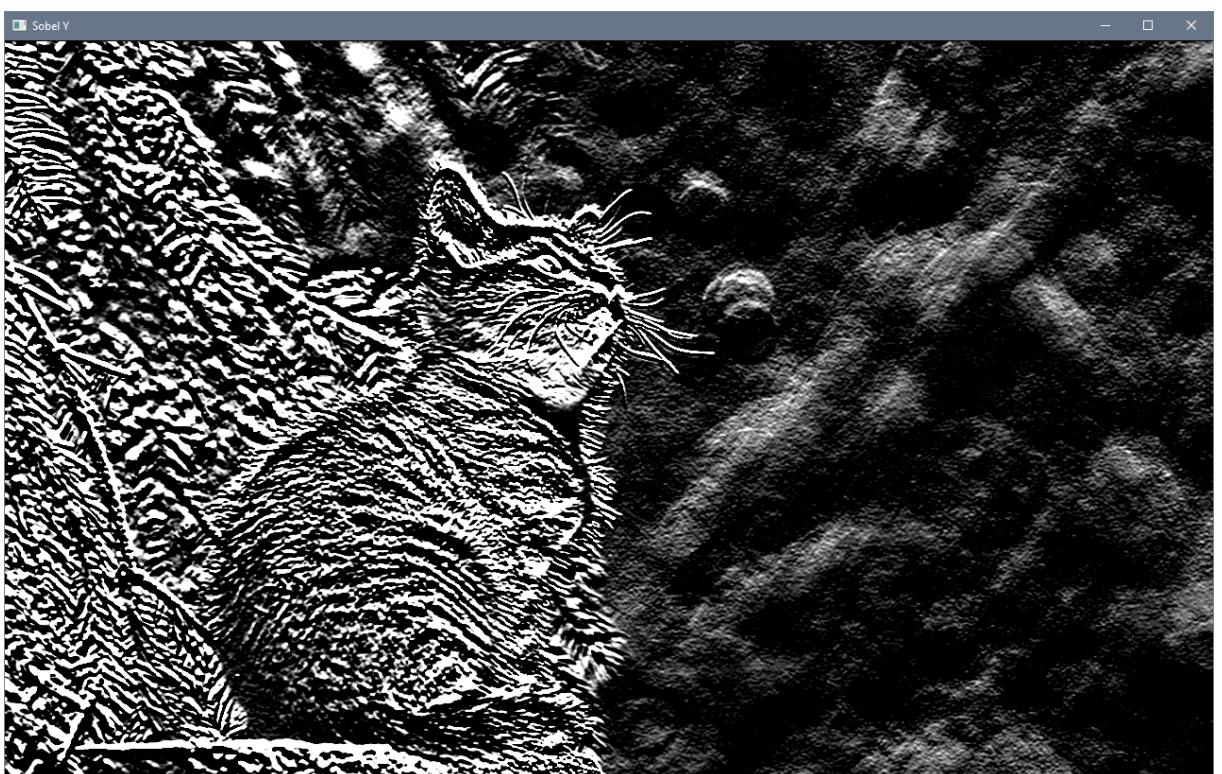
Operator Prewitta:



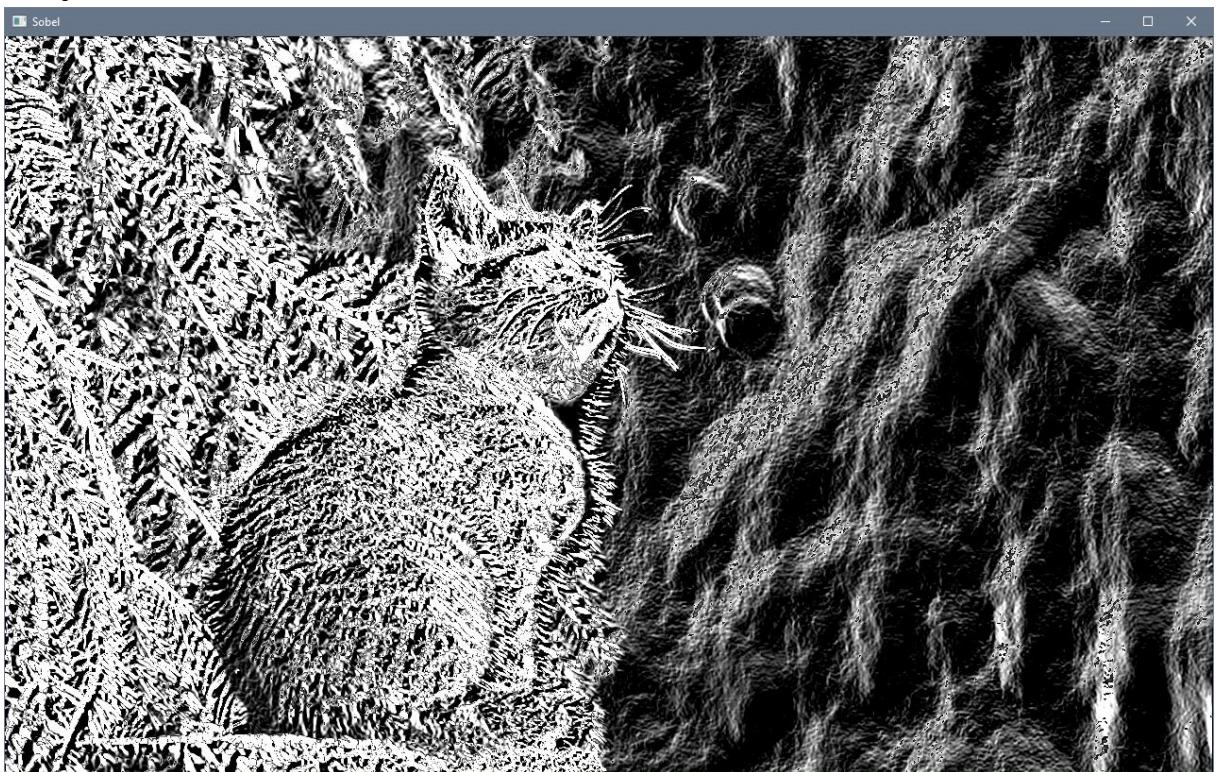
Sobel SX:



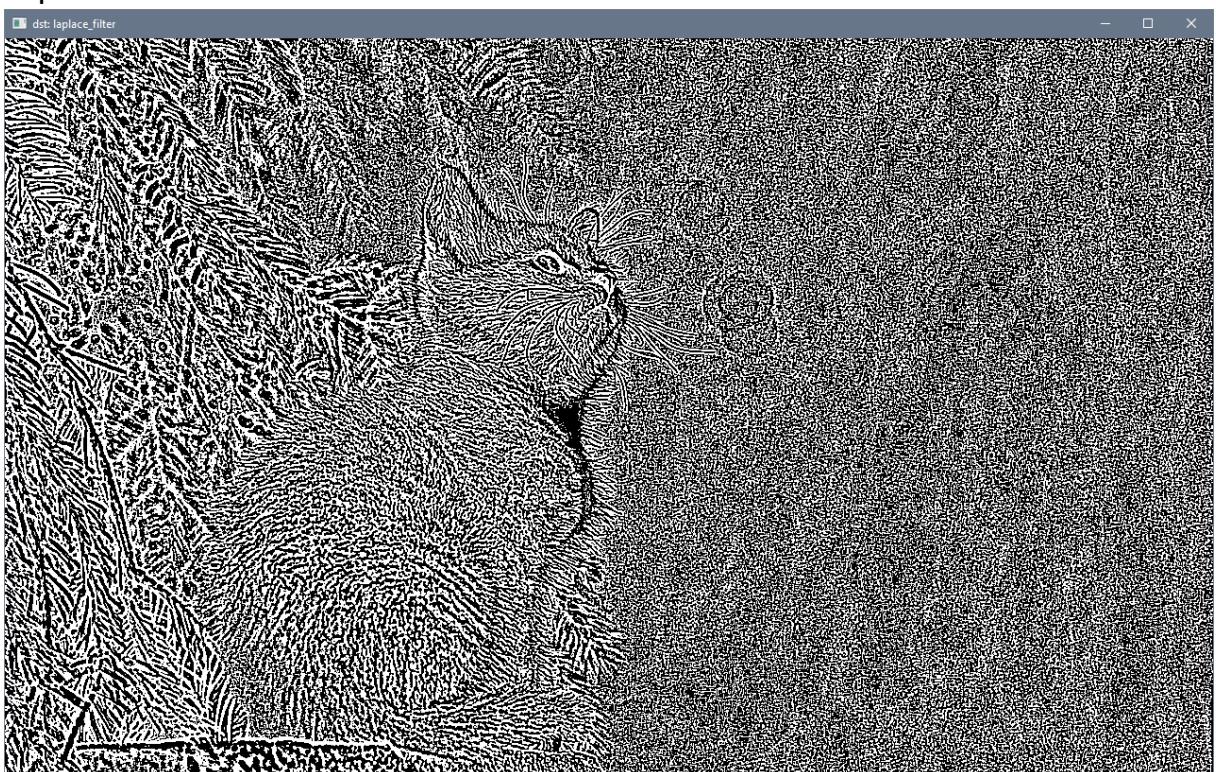
Sobel SY:



Połączenie – Sobel:



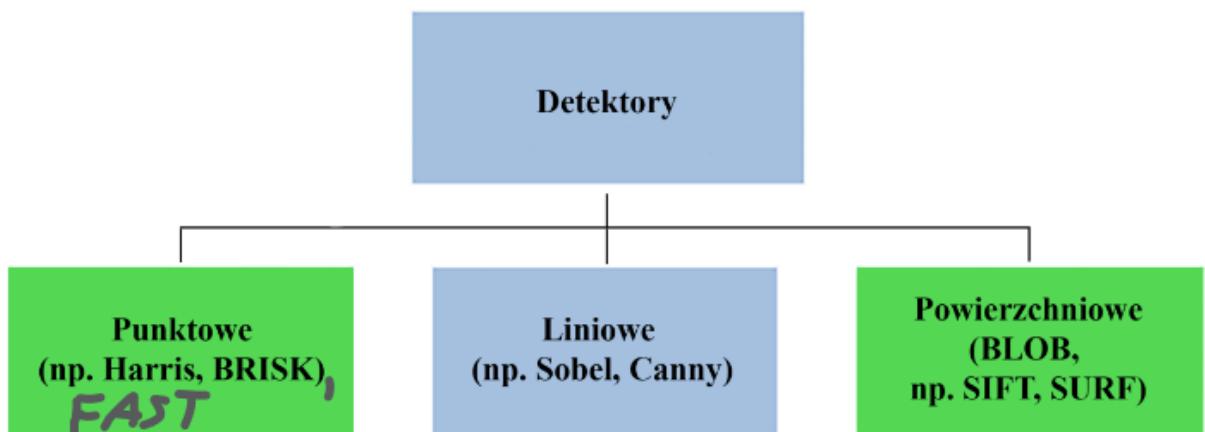
Laplace filter:



## 7. Analiza doboru parametrów w detekcji cech przy wykorzystaniu algorytmów FAST, SIFT oraz MSER

- Kod w folderze **p6** i **p7**
- Pliki **p6\_harris\_detector\_&\_FAST.ipynb** i **p7\_SIFT\_detector\_&\_MSER.ipynb**

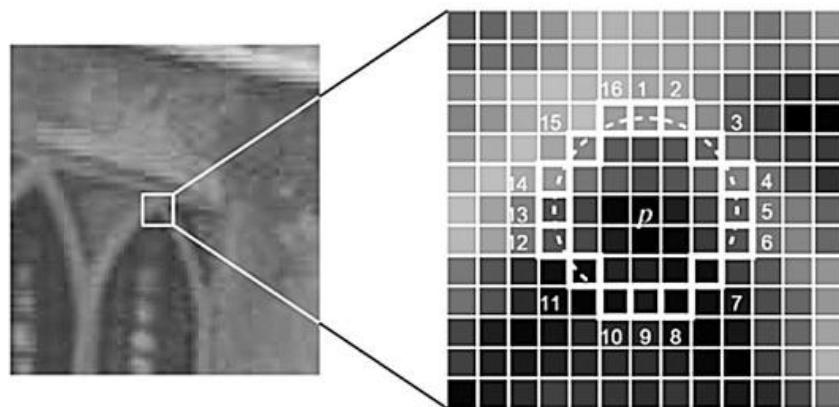
Standardowo do wykrywania punktów charakterystycznych na zdjęciach wykorzystywane są trzy rodzaje detektorów tj. punktowe, liniowe oraz powierzchniowe.



Koncepcja działania detektora FAST (ang. Features from Accelerated Segment Test), przedstawiona przez Rostena oraz Drummonda w 2006 roku, opiera się na założeniu, że punkty charakterystyczne posiadają jasno zdefiniowaną pozycję oraz są nośnikiem dobrze rozpoznawalnej informacji, pozwalającej na ich jednoznaczne wykrywanie na sąsiednich zdjęciach. Zaletą detektora FAST jest szybkość przetwarzania zdjęć, gdyż został zaprojektowany w celu wykrywania punktów wiążących w czasie rzeczywistym, np. w przypadku orientacji zdjęć z wykorzystaniem algorytmu SLAM. Działanie algorytmu FAST składa się z pięciu głównych kroków.

1. Wybranie na zdjęciu odpowiedniego piksela, który zostanie poddany analizie pod kątem przynależności do grupy punktów traktowanych jako narożnik. Wartość intensywności wyznaczonego punktu jest oznaczana symbolem  $I_p$ .
2. Określenie akceptowalnej wartości progu, poniżej której punkty są odrzucane.
3. Zdefiniowanie okręgu o promieniu 16 pikseli, dla którego będą analizowane wartości pikseli.

4. Zdefiniowanie początkowo wybranego punktu jako narożnika – jeżeli w jego sąsiedztwie (czyli okręgu o obwodzie 16 pikseli) znajduje się n pikseli o intensywności większej, bądź równej od  $I_p$ , to wybierane jest 12 z nich.  
5. Wyselekcyjowanie i porównanie pikseli oznaczonych numerami: 1, 3, 5 oraz 13, co ma na celu przyspieszenie działania pierwszego etapu algorytmu. W przypadku gdy dwa pierwsze piksele są ciemniejsze lub jaśniejsze od porównywanego piksela, to algorytm przechodzi do następnego kroku, czyli porównania kolejnych dwóch pikseli. Dla piksela stanowiącego potencjalny narożnik, niezbędne jest sprawdzenie, czy trzy piksele otaczające są jaśniejsze lub ciemniejsze. Jeżeli oba przedstawione warunki zostają spełnione, to porównywane są kolejne piksele w celu sprawdzenia poprawności przyjętej hipotezy.



Rysunek 3 Przykład detekcji narożnika wraz z okręgiem 16 pikseli poddanych dalszej analizie dla detektora FAST<sup>4</sup>

Duża szybkość działania algorytmu wiąże się z istotnym ograniczeniem – algorytm bywa niestabilny i nie zawsze działa poprawnie. Dla przykładu, jeżeli w oknie poszukiwawczym wykrytych zostało mniej niż 12 punktów, zbyt wiele punktów uznawanych jest za narożniki. Co więcej, czas trwania procesu wyboru punktów inicjalnych determinuje całkowity czas wyszukiwania punktów – narożników. Jednym z problemów wykorzystania algorytmu FAST jest duża liczba wykrytych punktów charakterystycznych traktowanych jako narożniki.. W celu wyboru najlepszych punktów charakterystycznych (stabilnych według teorii Mikołajczyka i Lowe), wykorzystuje się podejście Non-maximum Suppression, które zakłada następujące kroki:

- Obliczana jest funkcja kosztu, V dla wszystkich wykrytych punktów charakterystycznych. V jest sumą bezwzględnych różnic pomiędzy wartościami p i 16 otaczających go pikseli.
- Analizowane są dwa sąsiadujące punkty kluczowe , dla których wyznaczane są wartości V.
- Odrzucany jest punkt dla którego uzyskano niższą wartość funkcji kosztu V.

Zdefiniowana funkcja FAST, przyjmująca m.in. parametr *threshold*:

```
def fast(filepath, threshold=10, nonmaxSuppression=True, type=cv2.FAST_FEATURE_DETECTOR_TYPE_9_16):
    """
    FAST: Features from Accelerated Segment Test
    cv2.FAST_FEATURE_DETECTOR_TYPE_9_16, cv2.FAST_FEATURE_DETECTOR_TYPE_5_8, cv2.FAST_FEATURE_DETECTOR_TYPE_7_12
    """
    img = cv2.imread(filepath)
    img_grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    fast = cv2.FastFeatureDetector.create(threshold, nonmaxSuppression, type)
    keypoints = fast.detect(img_grey, None)
    img2 = cv2.drawKeypoints(img_grey, keypoints, None, color=(255, 0, 0))
    cv2.imshow('Punkty wykryte algorytmem Fast', img2)
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
    return img2, keypoints # . . .
```

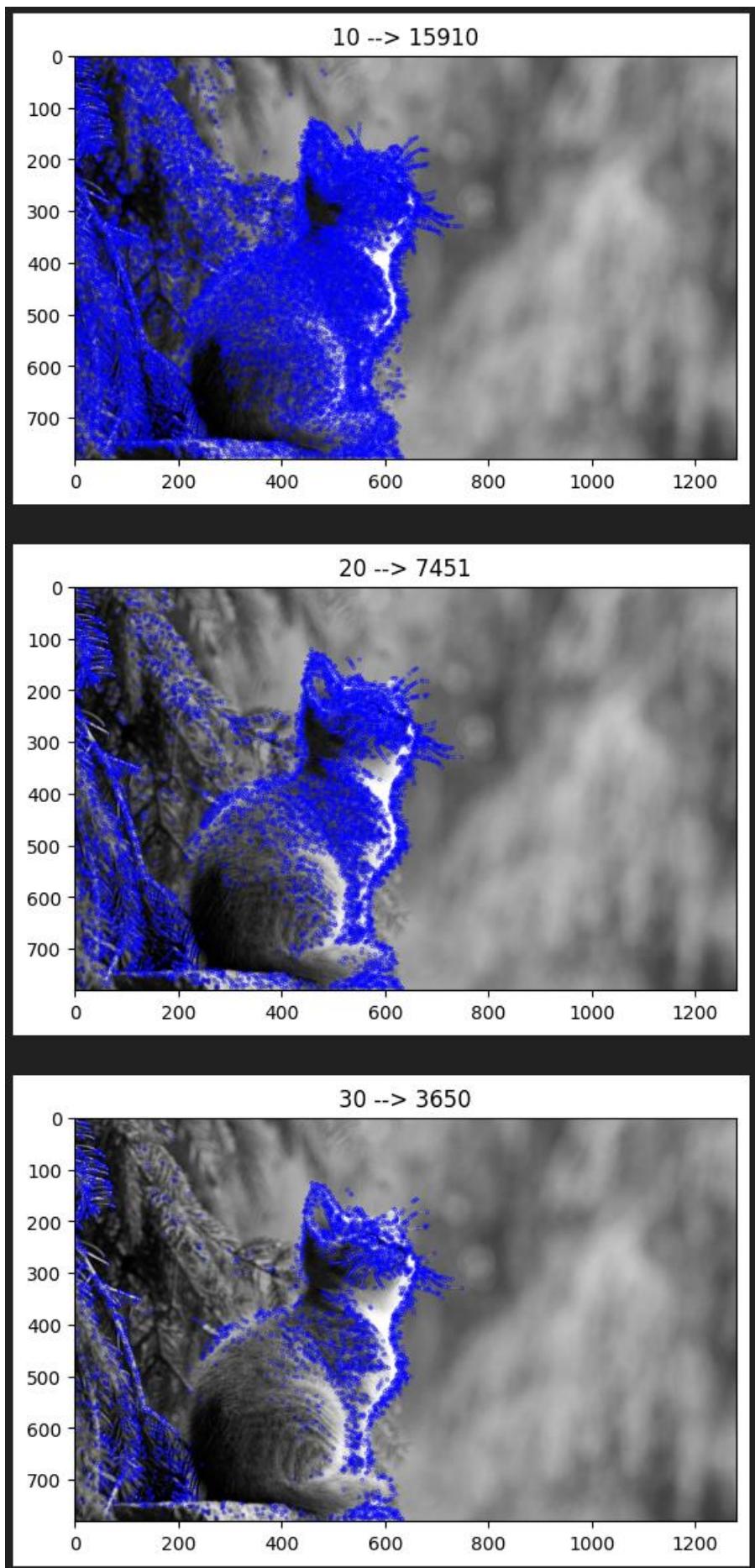
Python

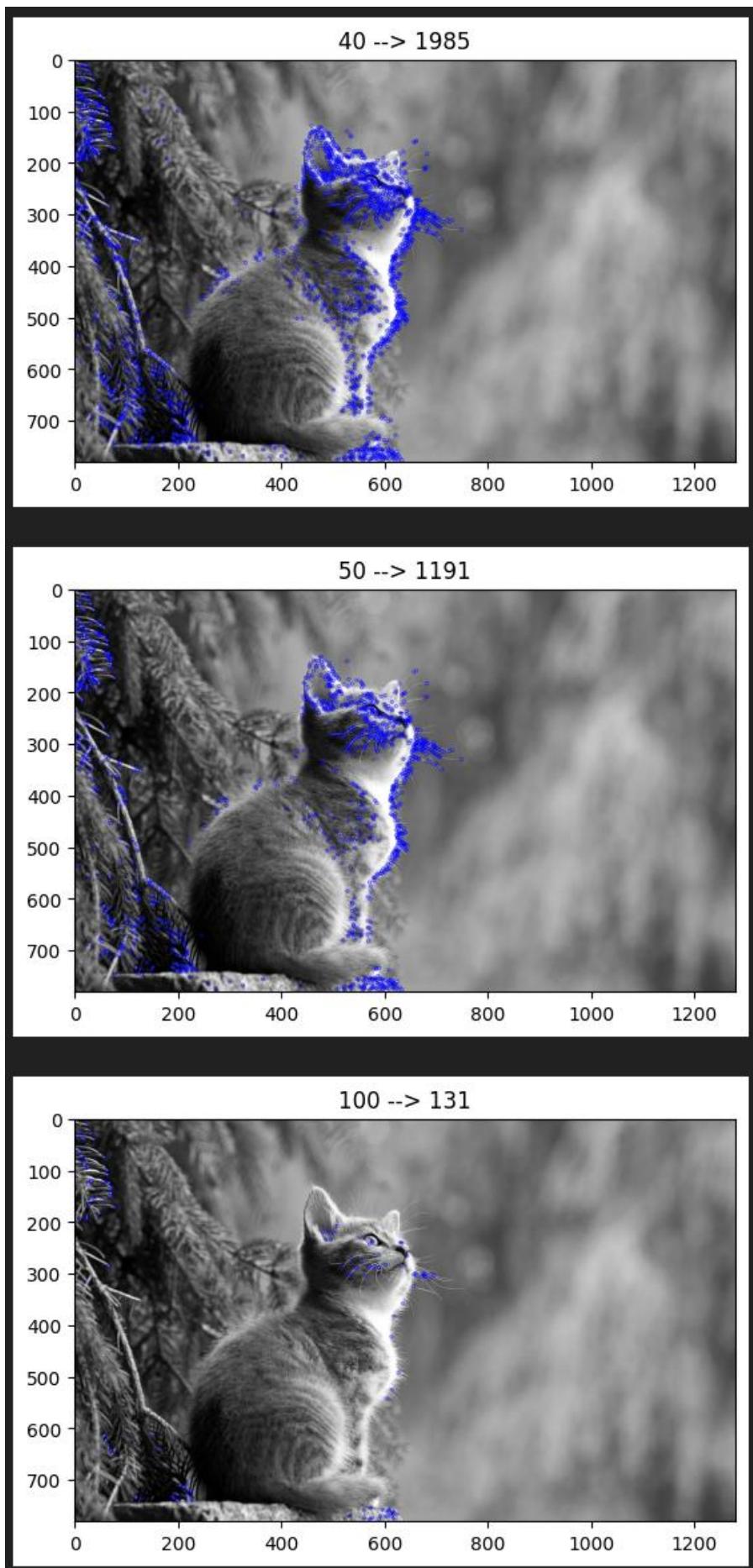
## FAST różne parametry threshold

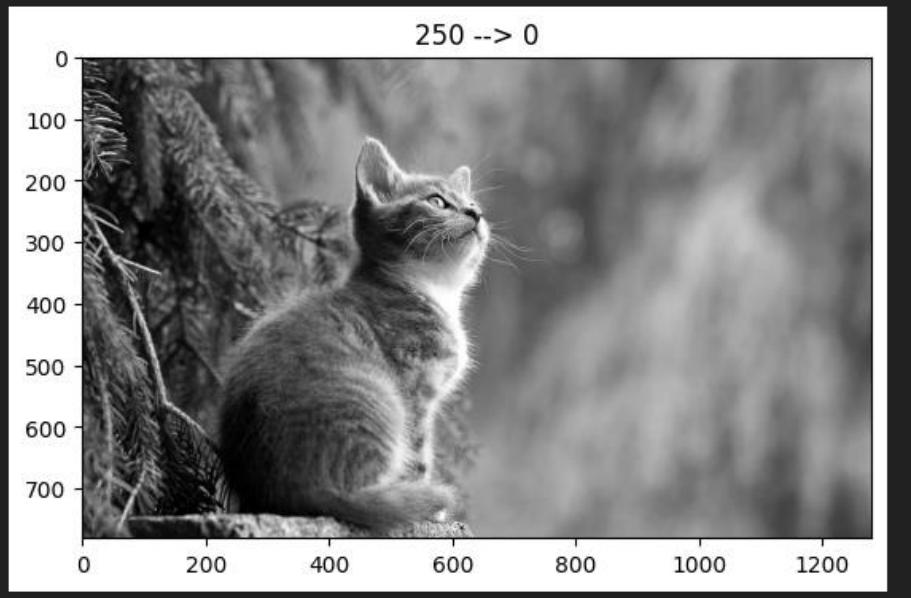
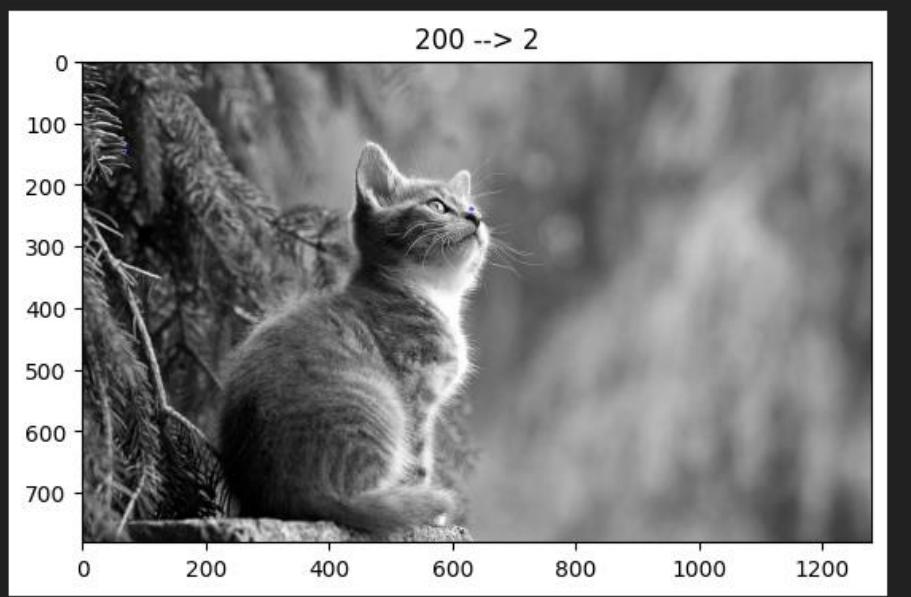
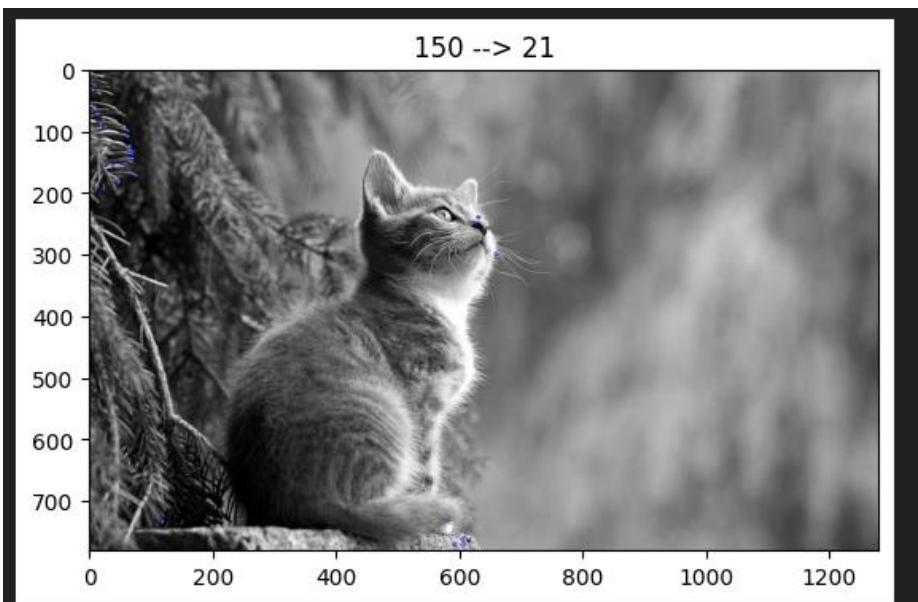
```
threshold_values = [10, 20, 30, 40, 50, 100, 150, 200, 250]
detected_points = []
for t in threshold_values:
    img, pts = fast(fp1, threshold=t, nonmaxSuppression=True)
    no_pts = len(pts)
    detected_points.append(no_pts)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"{t} --> {no_pts}")
    plt.show()
```

[15]

Python

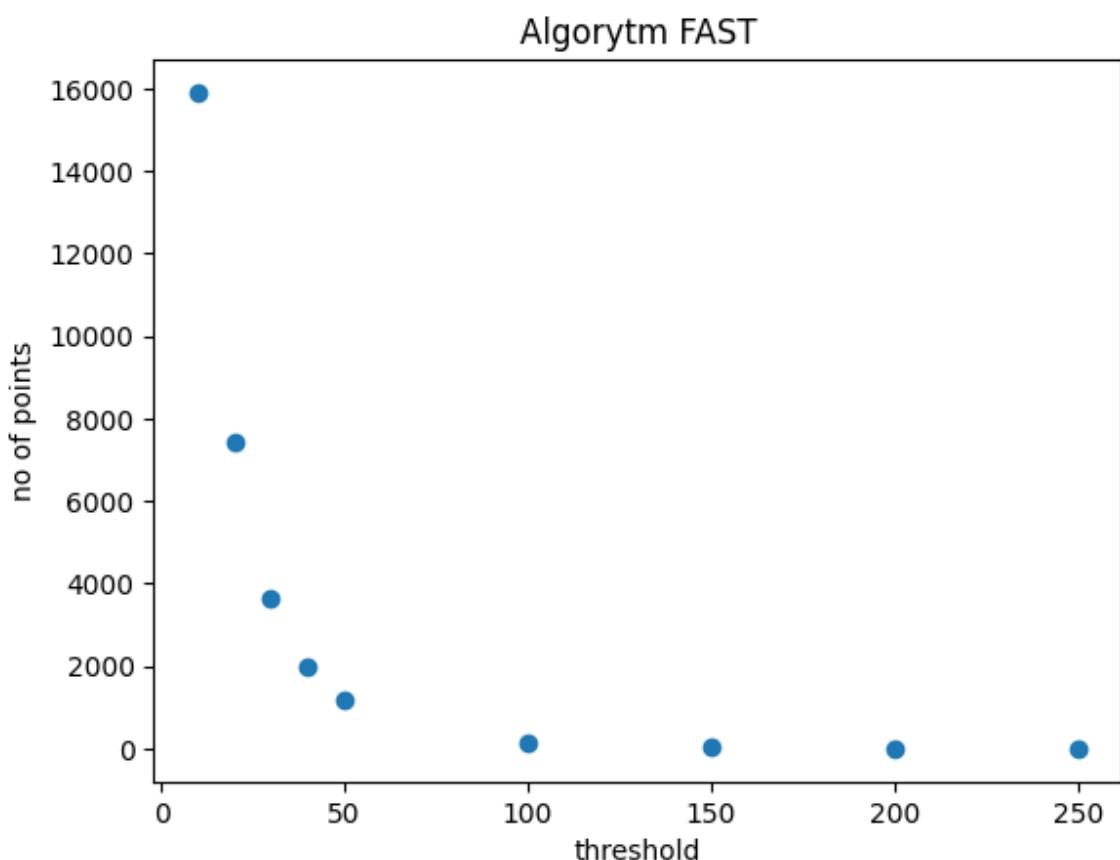






Liczba wykrytych punktów w zależności od parametru threshold:

```
plt.scatter(threshold_values, detected_points)
plt.xlabel("threshold")
plt.ylabel("no of points")
plt.title("Algorytm FAST")
plt.show()
```



Detektor SIFT (Scale Invariant Feature Transform) był jednym z pierwszych algorytmów umożliwiających wykrywanie regionów tzw. BLOB-ów, charakteryzujących się niezmiennymi cechami bez względu na skalę obrazów oraz wpływ oświetlenia. Głównym założeniem algorytmu SIFT jest zastosowanie tzw. detektora Difference – of – Gaussian (DoG; Lindberg, 1993). Jego głównym założeniem jest to, że potencjalne punkty dla cech lokalnych (niezależne od skali i orientacji) odpowiadają lokalnym ekstremom na obrazach różnicowych uzyskanych po filtracji filtrem Gaussa w różnych skalach. Działanie algorytmu SIFT składa się z następujących etapów:

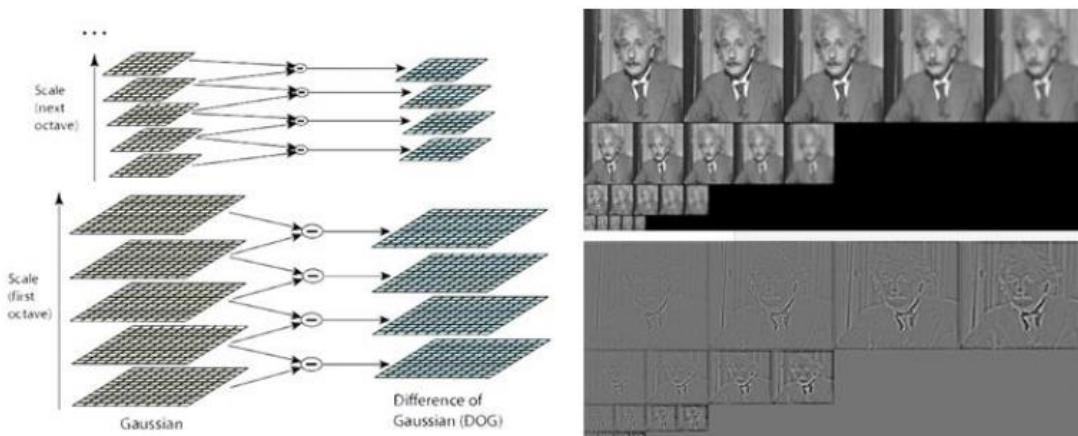
#### *Wykrywanie ekstremów funkcji w różnych skalach*

W pierwszym etapie wykrywania ekstremów funkcji w przestrzeni skalowania odbywa się poprzez porównanie obrazu oryginalnego z sąsiadującymi obrazami utworzonymi poprzez zastosowanie przekształcenia Gaussa. Przyjmuje się, że potencjalne punkty dla cech lokalnych (niezależne od skali i orientacji) odpowiadają lokalnym ekstremom na obrazach różnicowych uzyskanych po filtracji filtrem Gaussa w różnych skalach. Oryginalny obraz

$I(x,y)$  jest poddawany filtracji przy wykorzystaniu funkcji Gaussa przedstawionej za pomocą wzoru:

$$L(x, y, \sigma) = G(x, y\sigma) * I(x, y)$$

Obraz wejściowy filtrowany jest filtrem Gaussa w różnych skalach rozdzielonych stałym współczynnikiem  $k$ , a następnie obliczane są obrazy różnicowe DoG (Difference of Gaussian). Tą procedurę powtarza się przeprowadzając obraz poprzez wyrzucenie co 2 wiersza i kolumny.



Rysunek 4 A) Przykład działania filtracji funkcja DoG (Difference of Gaussian); B) Przykład obrazu przetworzonego filtrem Gaussowskim i DoG na różnych poziomach.

Funkcję DoG (Difference of Gaussian) można przedstawić za pomocą wzoru:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Wykrywanie lokalnych ekstremów funkcji odbywa się poprzez porównanie punktu z jego sąsiadami w otoczeniu  $3 \times 3 \times 3$ . Punkt jest kwalifikowany jako kandydat na punkt kluczowy cechy, gdy ma wartość większą lub mniejszą od punktów sąsiednich.

### Lokalizacja punktów kluczowych (ang. keypoints)

Kolejnym ważnym krokiem związanym z dzianiem detektora jest wykrycie tzw. punktów kluczowych (keypoints). Lokalizacja punktów kluczowych odbywa się poprzez porównanie wartości funkcji dopasowania, obliczonej na podstawie ekstremów:

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x, x = (x, y, \sigma)^T$$

gdzie D i pochodne tej funkcji obliczane są dla danego punktu, a x jest przesunięciem względem tego punktu. Położenie ekstremum wylicza z funkcji:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x}$$

W celu eliminacji punktów o słabym kontraście wykorzystuje się poniższą funkcję. Przyjmuje się, że gdy wartość ekstremum jest mniejsze niż 0.03 (w przypadku wartości intensywności z przedziału {0,1}), to dany punkt posiada słaby kontrast (Lowe, 2004).

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Dzięki wykorzystaniu powyższej zależności możliwe jest usunięcie punktów leżących na linii. Wykorzystując funkcję DoG, możemy się spodziewać, że algorytm będzie mieć silną odpowiedź wzduż krawędzi, nawet jeżeli krawędź jest słabo odwzorowana na obrazie. W celu wyznaczenia głównych krzywizn należy wykorzystać Hessian:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xt} & D_{yy} \end{bmatrix}$$

Pod pojęciem Hessiana rozumiemy macierz kwadratową drugich pochodnych cząstkowych funkcji o wartościach rzeczywistych dwukrotnie różniczkowej w pewnym punkcie dziedziny.

Wartości własne H są proporcjonalne do głównych krzywizn D. Wykorzystując podejście zaproponowane przez Harrisa i Stephensa (1988), możemy wykorzystać stosunek uzyskanych wartości własnych macierzy H. Przyjmując  $\alpha$  jako największą wartość własną macierzy, a  $\beta$  jako najmniejszy możemy wyznaczyć wartości śladu macierzy i jej wyznacznika:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta 2$$

W przypadku, gdy wyznacznik ma wartość ujemną, krzywizny mają różne znaki, a więc punkt jest odrzucany, ponieważ nie jest on maksimum funkcji. Przyjmując r jako stosunek największej do najmniejszej wartości własne macierzy możemy zapisać to w postaci zależności:  $\alpha = r\beta$ , stąd:

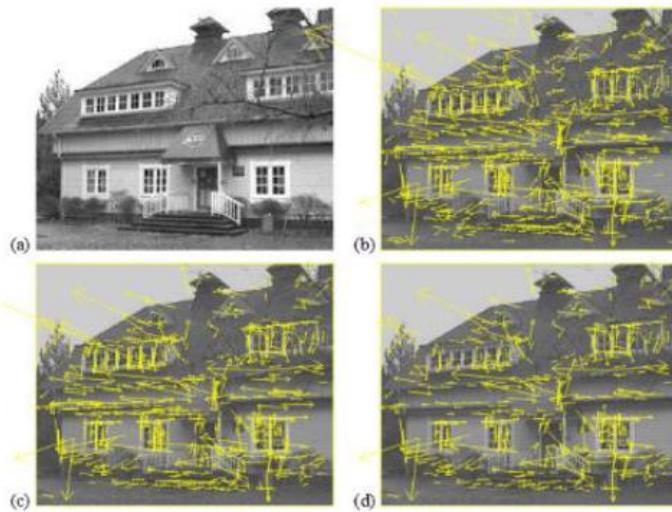
$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Ta zależność przyjmuje wartości minimalne w przypadku, gdy dwie wartości własne macierzy są sobie równe, natomiast wzrasta gdy wzrasta wartość r. W celu sprawdzenia, czy wyszukane punkty nie leżą na linii, należy jedynie sprawdzić, czy zostaje spełniony warunek przy założonej wartości progu r:

$$\frac{Tr(H)^2}{Det(H)} \leq \frac{(r+1)^2}{r}$$

Zastosowanie tego rozwiązania jest bardzo efektywne ze względu na fakt, iż niezbędne jest wykonanie mniej niż 20 operacji w celu sprawdzenia poprawności doboru punktu charakterystycznego. Lowe zaproponował w swojej publikacji zarekomendował używanie wartości  $r = 10$  (Lowe, 2004).

Na rysunku 16 przedstawione zostały wyniki działania algorytmu detekcji punktów przy wykorzystaniu współczynnika filtracji  $D(\hat{x}) = 0.03$  oraz  $\frac{(r+1)^2}{r} = 10$ . W wyniku tego procesu z początkowo wykrytych 832 punktów, odfiltrowano 102 punkty charakteryzujące się niskim kontrastem i prawdopodobnie znajdujące się na liniach.



Rysunek 5 Etapy wyboru punktów charakterystycznych a) Oryginalne zdjęcie w rozdzielcości 233x189 pikseli, b) Wstępnie wykrytych 832 punktów charakterystycznych (keypoints), bazujących na maximach i minimach wartości funkcji Difference of Gaussian. Wykryte punkty wiążące zaznaczono jako wektory ze skalą i orientacją c) Wyniki filtracji punktów charakterystycznych (729) przy zastosowaniu progu filtracji 0.003, d) Wyniki filtracji punktów charakterystycznych (536) leżących na linii ((Lowe, 2004)).

### Zdefiniowana funkcja SIFT wraz z opisanymi parametrami:

```
def sift(img_filepath, nfeatures=0, nOctaveLayers=3, contrastThreshold=0.04, edgeThreshold=10, sigma=1.6):
    """
    Parameters
    img_filepath    : file path to RGB image.
    nfeatures       : The number of best features to retain. The features are ranked by their scores (measured in SIFT algorithm as the local contrast).
    nOctaveLayers   : The number of octave layers to use. 3 is the value used in the paper. The number of octaves is derived automatically from the image resolution.
    contrastThreshold: The contrast threshold used to Filter out weak features in semi-inform (low-contrast) regions. The larger the threshold, the less features are produced by the detector.
    edgeThreshold   : The threshold used to filter out edge-like features. Note that the its meaning is different from the contrastThreshold, i.e., the larger the edgeThreshold, the less features are filtered out (more features are retained).
    sigma           : The sigma of the gaussian applied to the input image at the octave #. If your image is captured with a weak camera with soft lenses, you might want to reduce the number.
    note            : The contrast threshold will be divided by nOctaveLayers when the filtering is applied. When nOctaveLayers is set to default and if you want to use the value used in D. Lowe paper, 0.04, set this argument to 0.09.

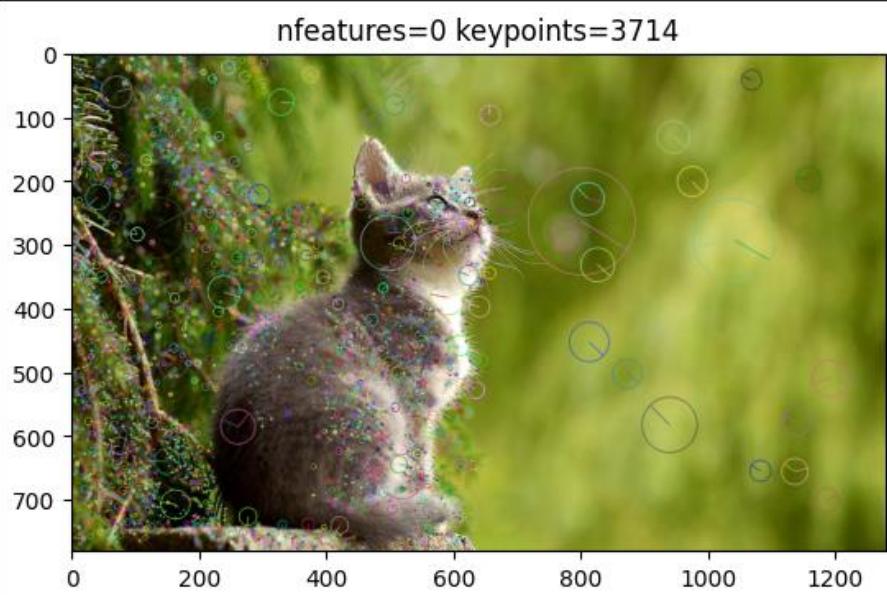
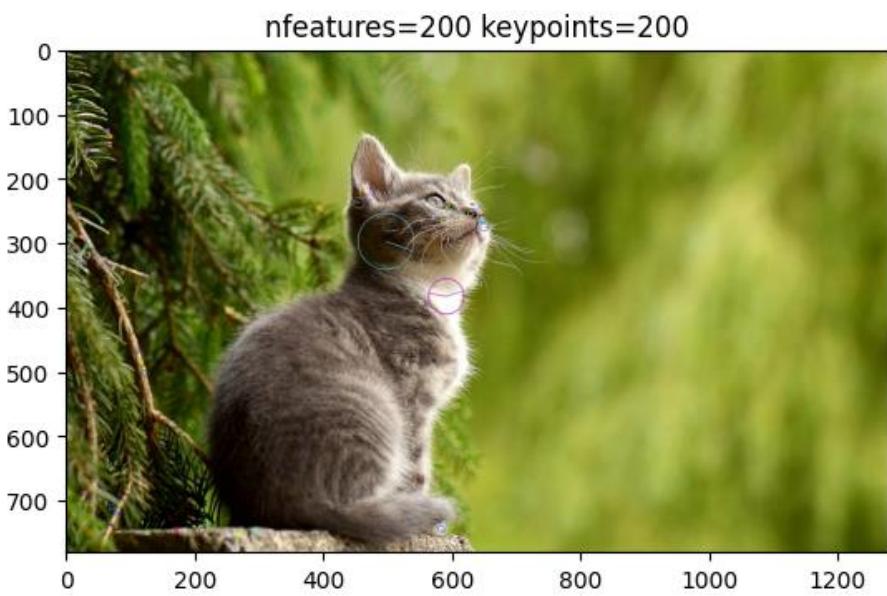
    See https://docs.opencv.org/3.4/d1/d0f/tutorial_py_sift_intro.html
    See https://docs.opencv.org/3.4/d7/d8b/classcv_1_1sift.html
    """

    img = cv2.imread(img_filepath)
    img_grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create(nfeatures, nOctaveLayers, contrastThreshold, edgeThreshold, sigma)
    keypoints = sift.detect(img_grey, None)
    img1 = cv2.drawKeypoints(img, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # If cv2.waitKey(0) & 0xFF == 27:
    #     cv2.destroyAllWindows()
    return img1, keypoints
```

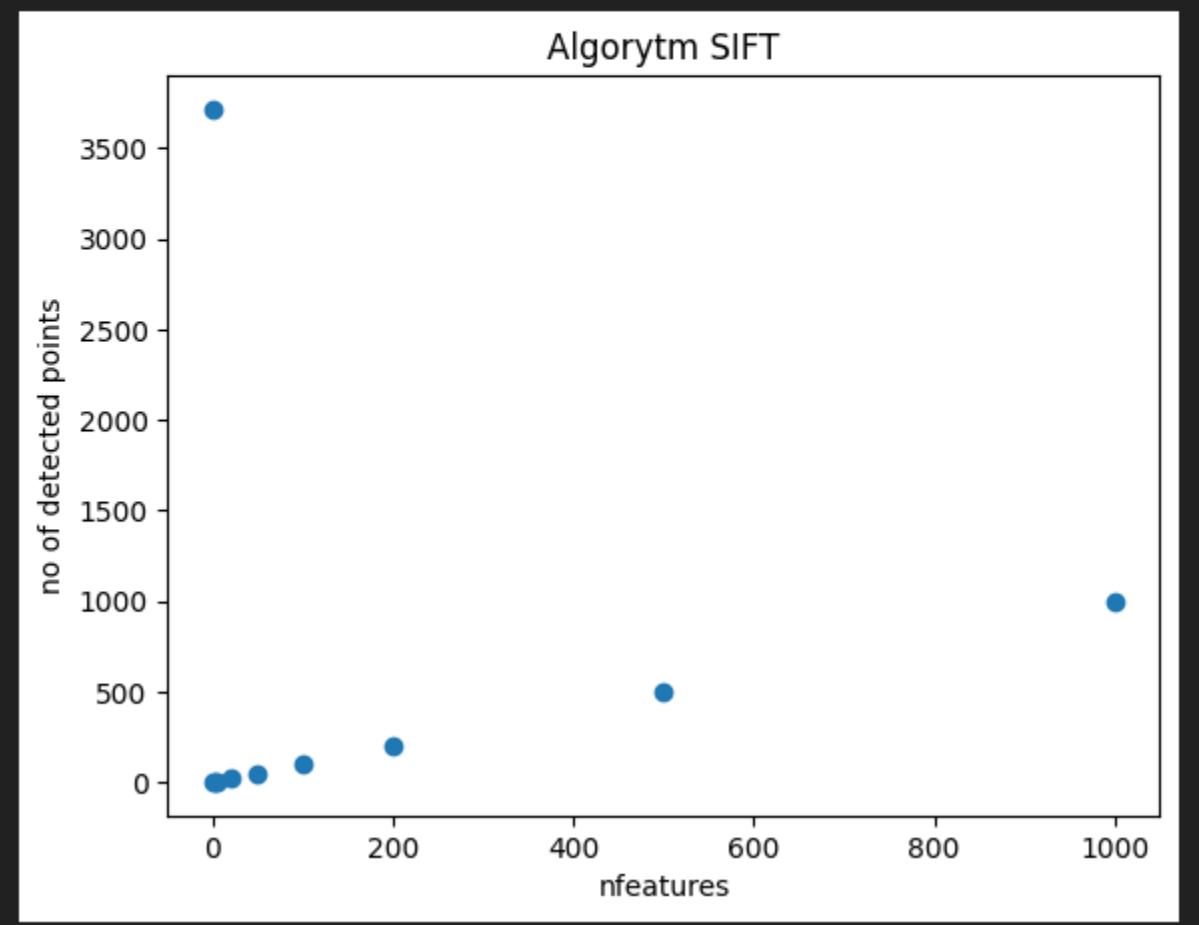
# Różne wartości parametru nfeatures

```
nfeatures = [0, 1, 3, 5, 20, 50, 100, 200, 500, 1000]
detected_points = []
for n in nfeatures:
    img, kp = sift(img_filepath=fp1, nfeatures=n)
    detected_points.append(len(kp))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"nfeatures={n} keypoints={len(kp)}")
    plt.show()
```



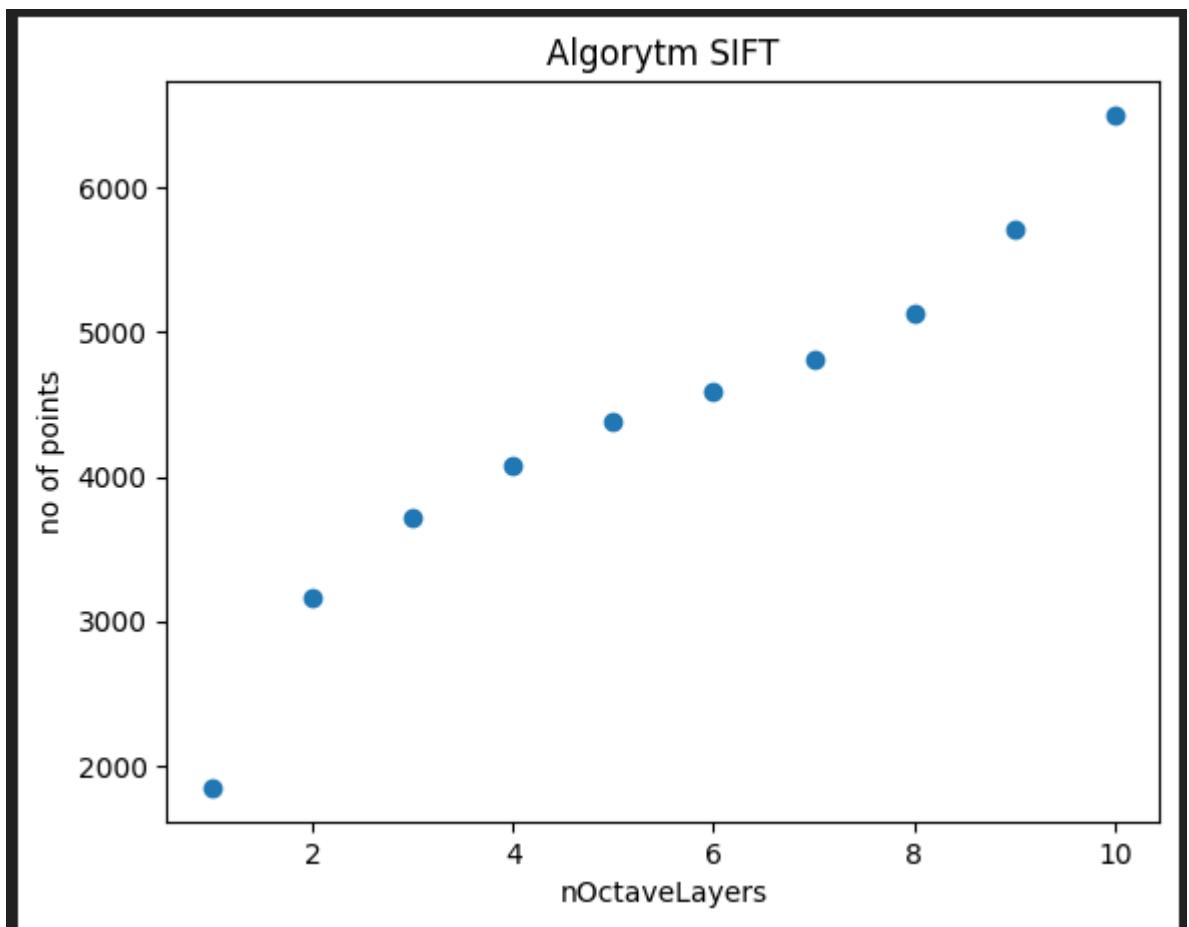
Więcej obrazków w kodzie **p7\_SIFT\_detector\_&\_MSER.ipynb**

```
plt.scatter(nfeatures, detected_points)
plt.xlabel("nfeatures")
plt.ylabel("no of detected points")
plt.title("Algorytm SIFT")
plt.show()
```



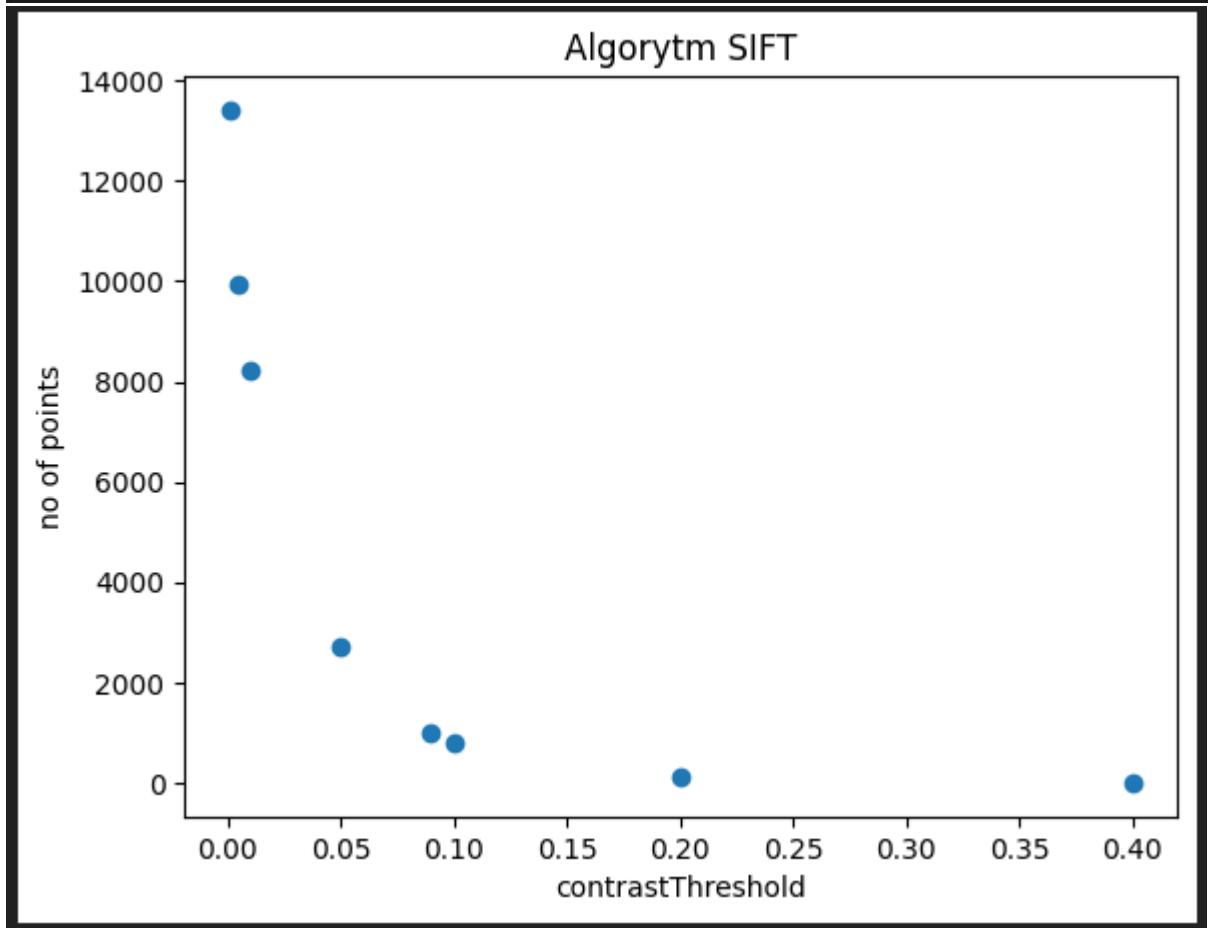
# Różne wartości parametru nOctaveLayers

```
nOctaveLayers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
detected_points = []
for n in nOctaveLayers:
    img, kp = sift(img_filepath=fp1, nOctaveLayers=n)
    detected_points.append(len(kp))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"nOctaveLayers={n} keypoints={len(kp)}")
    plt.show()
```



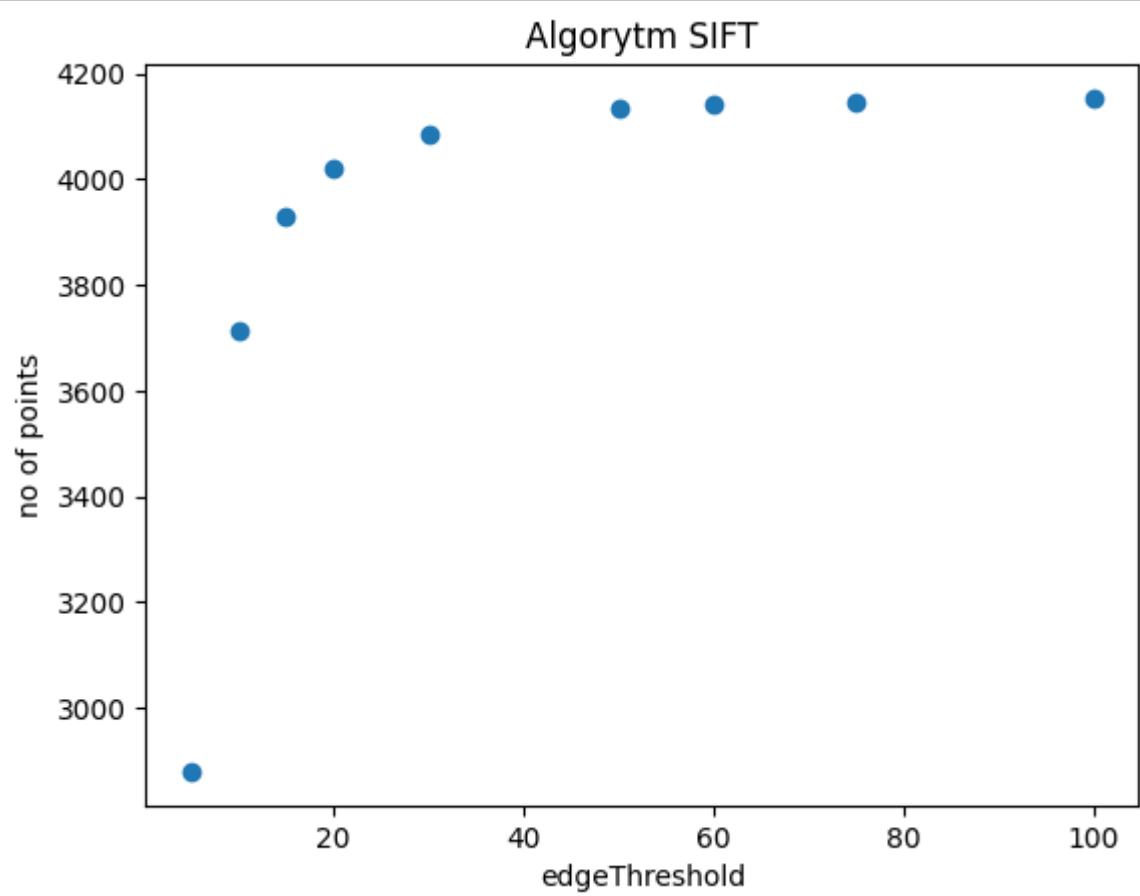
## Różne wartości parametru contrastThreshold

```
contrastThreshold = [0.001, 0.005, 0.01, 0.05, 0.09, 0.1, 0.2, 0.4]
detected_points = []
for n in contrastThreshold:
    img, kp = sift(img_filepath=fp1, contrastThreshold=n)
    detected_points.append(len(kp))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"contrastThreshold={n} keypoints={len(kp)}")
    plt.show()
```



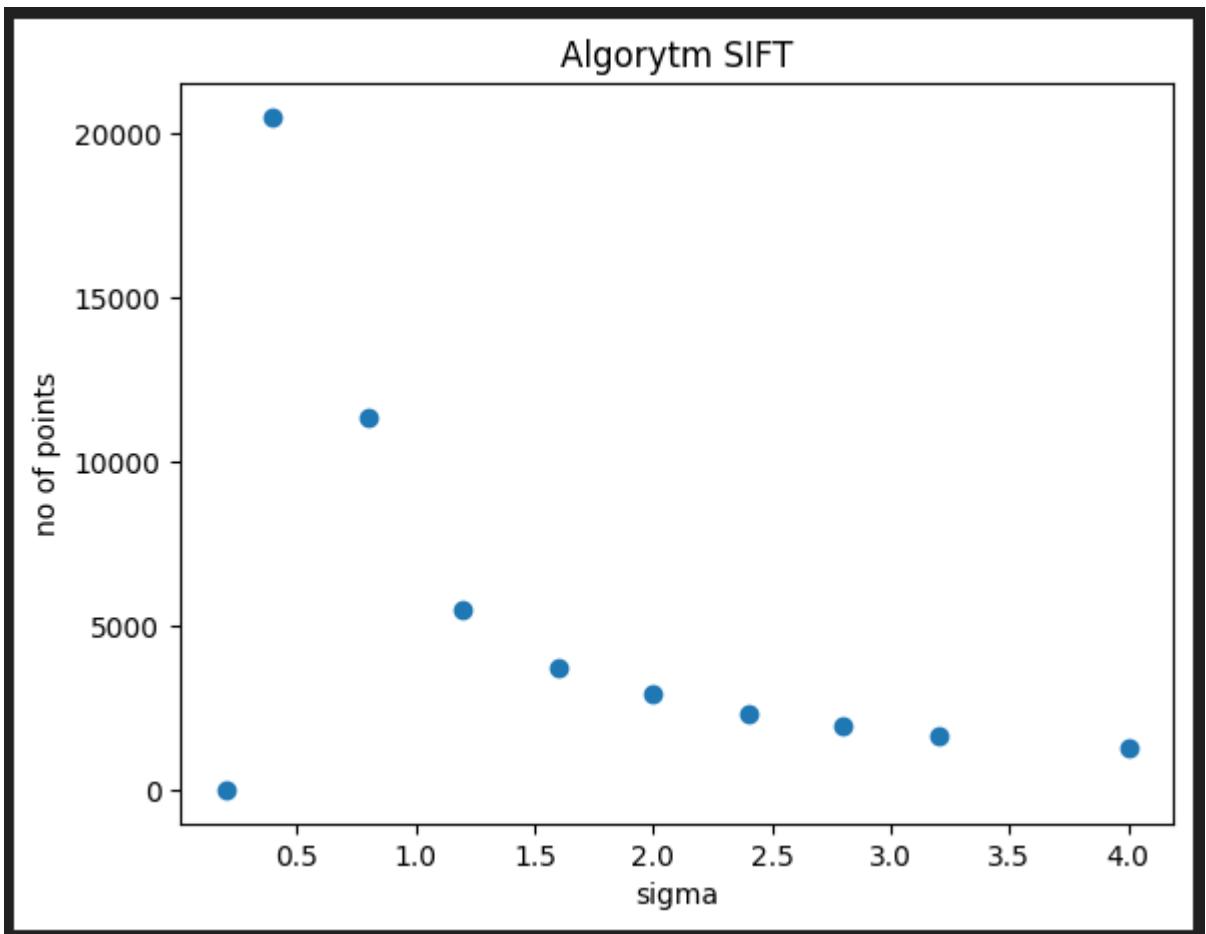
# Różne wartości parametru edgeThreshold

```
edgeThreshold = [5, 10, 15, 20, 30, 50, 60, 75, 100]
detected_points = []
for n in edgeThreshold:
    img, kp = sift(img_filepath=fp1, edgeThreshold=n)
    detected_points.append(len(kp))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"edgeThreshold={n} keypoints={len(kp)}")
    plt.show()
```



# Różne wartości parametru sigma

```
sigma = [0.2, 0.4, 0.8, 1.2, 1.6, 2.0, 2.4, 2.8, 3.2, 4.0]
detected_points = []
for n in sigma:
    img, kp = sift(img_filepath=fp1, sigma=n)
    detected_points.append(len(kp))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title(f"sigma={n} keypoints={len(kp)}")
plt.show()
```



## MSER

Na zajęciach w mojej grupie zajęciowej nie mieliśmy robić analizy parametrów MSER.

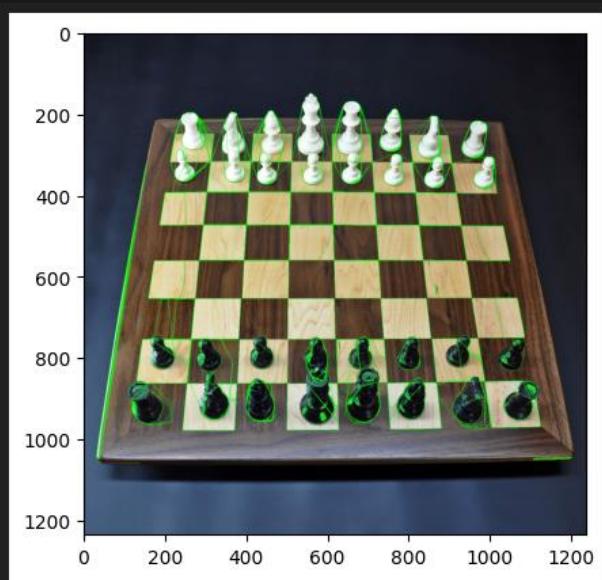
```
def mser(img_filepath):
    """
    # https://docs.opencv.org/4.x/d3/d28/classcv_1_1MSER.html
    """
    img = cv2.imread(img_filepath)
    img_grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    mser = cv2.MSER_create() # domyslne parametry
    regions, boundingBoxes = mser.detectRegions(img_grey)
    hulls = [cv2.convexHull(p.reshape(-1, 1, 2)) for p in regions]
    # cv2.polylines(img, hulls, 1, (0, 255, 0))
    # cv2.imshow('img', img)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
    return hulls, regions, boundingBoxes
```

## Detektor MSER (Maximally Stable Extremal Regions)

```
hulls, regions, bboxes = mser(img_filepath=fp2)

img = cv2.imread(fp2)
cv2.polylines(img, hulls, 1, (0, 255, 0))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()

# cv2.imshow('img', img)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

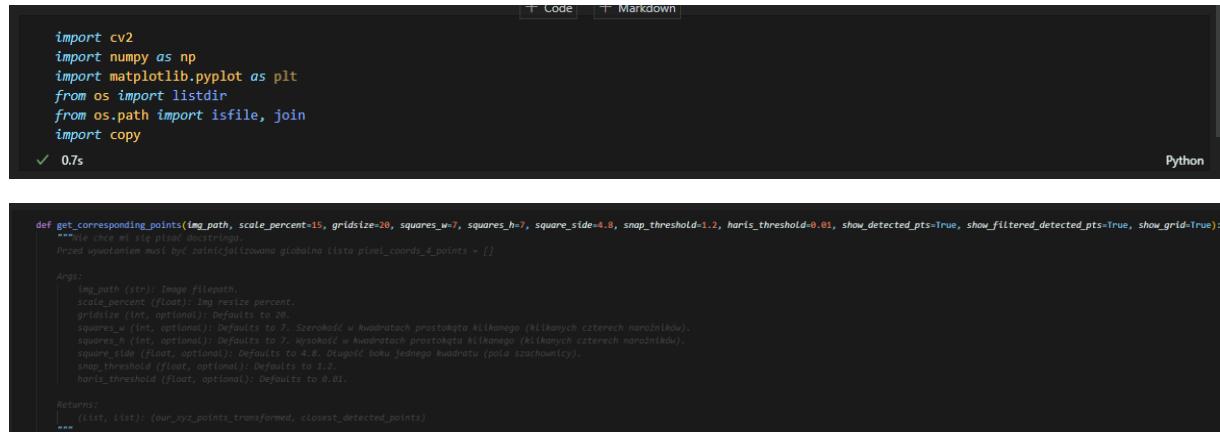


## 8. Program do wykonywania kalibracji kamer

Opis zagadnienia i kod można znaleźć na stronie

[https://docs.opencv.org/4.x/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html).

Na zajęciach podeszliśmy do zagadnienia trochę inaczej i sami klikaliśmy punkty narożne, określaliśmy wymiary siatki punktów, wykrywaliśmy punkty detektorem Harrisa itp. Całość jest opisana w funkcji `get_corresponding_points` w pliku **p10\_kalibracja\_kamery.ipynb**.



The screenshot shows a Jupyter Notebook cell with the following code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from os import listdir
from os.path import isfile, join
import copy

def get_corresponding_points(img_path, scale_percent=15, gridsize=20, squares_w=7, squares_h=7, square_side=4.8, snap_threshold=1.2, haris_threshold=0.01, show_detected_pts=True, show_filtered_detected_pts=True, show_grid=True):
    """Właściwość img_path musi być zapisywana globalnie lista pixel_coords_4_points = []
    Args:
        img_path (str): Image filepath.
        scale_percent (float): Img resize percent.
        gridsize (int, optional): Defaults to 20.
        squares_w (int, optional): Defaults to 7. Szerokość w kwadratach prostokąta kilkatego (kilkaowych czterech narożników).
        squares_h (int, optional): Defaults to 7. Wysokość w kwadratach prostokąta kilkatego (kilkaowych czterech narożników).
        square_side (float, optional): Defaults to 4.8. Długość boku jednego kwadratu (pole szachownicy).
        snap_threshold (float, optional): Defaults to 1.2.
        haris_threshold (float, optional): Defaults to 0.01.

    Returns:
        (list, list): (envr_yx_points_transformed, closest_detected_points)
    """
    ...

    # Code implementation follows...
```

```



```

```

# Transformacja wykrytych narożników do naszego układu współrzędnych
detected_points_in_our_coords = warpPoints(detected_points, macierz_transformacji)

# Dla każdego punktu z naszej siatki wykrycie najbliższego punktu wykrytego detektorem
closest_detected_points_in_our_coords = []
our_xy_points_2 = []
for p in our_xy_points:
    distances = np.linalg.norm(detected_points_in_our_coords - p, axis=1)
    min_index = np.argmin(distances)
    if distances[min_index] <= snap_threshold:
        closest_detected_points_in_our_coords.append(detected_points_in_our_coords[min_index])
        our_xy_points_2.append(p)

# Transformacja punktów z detektora (najbliższych siatce) do ich oryginalnego pikselowego układu współrzędnych zdjęcia
closest_detected_points = warpPoints(closest_detected_points_in_our_coords, macierz_odwrotna)

# Wyświetlenie odfiltrowanych punktów z detektora
if show_filtered_detected_pts:
    image = copy.deepcopy(img_resized)
    for v in closest_detected_points:
        image = cv2.circle(image, (int(v[0]), int(v[1])), radius=8, color=(0, 0, 0), thickness=5)
    cv2.imshow('closest detected points (harris detector)', image)
    if cv2.waitKey(0) & 0xFF == 27:
        cv2.destroyAllWindows()

# Transformacja siatki z naszego układu do układu pikselowego zdjęcia
our_xy_points_transformed = warpPoints(our_xy_points_2, macierz_odwrotna)

# Wyświetlenie przetransformowanych punktów grid
if show_grid:
    image = copy.deepcopy(img_resized)
    for v in our_xy_points_transformed:
        image = cv2.circle(image, (int(v[0]), int(v[1])), radius=8, color=(150, 8, 50), thickness=5)
    cv2.imshow('our xy points transformed', image)
    if cv2.waitKey(0) & 0xFF == 27:
        cv2.destroyAllWindows()

img_grey_resized_size = img_grey_resized.shape[:-1]
our_xyz_points_transformed = np.hstack((np.array(our_xy_points_transformed), np.zeros(shape=(len(our_xy_points_transformed), 1))).astype('float32')) #.tolist()
# return our_xyz_points_transformed, closest_detected_points, img_grey_resized_size

# TODO nwm co się dzieje... CO TRZEBIA ZWRACAĆ?
our_points = []
for pair in our_xy_points_2:
    our_points.append((pair[0], pair[1], 0))
our_points = np.array(our_points, dtype=np.dtype("float32"))
detected_points = np.expand_dims(np.array(closest_detected_points, dtype=np.dtype("float32")), axis=1)
return our_points, detected_points, img_grey_resized_size

```

✓ 0.0s    + Code    + Markdown

Funkcji tej należy użyć dla każdego z dwunastu (bo tyle zrobiliśmy na zajęciach) zdjęć szachownicy na ścianie. W wyniku tego, dostaniemy dla każdego zdjęcia po dwie listy, które potem należy połączyć i użyć jako argumenty *objectPoints* i *imagePoints* w funkcji *cv2.calibrateCamera()*. Przy czym trzeba pamiętać, by argument tej funkcji *cameraMatrix* ustawić na *None*. Dzięki zastosowaniu funkcji *calibrateCamera*, dostaniemy pięć wartości opisujących m.in. błąd RMS reprojekcji oraz inne wartości opisujące kalibrację kamery.

**Output Explanation:**

The *ret variable* is used to print the Error in projection i.e. the overall RMS re-projection error.

The Camera Matrix displays the intrinsic parameters of the camera which is focal length and optical center values.

The Distortion Coefficients Matrix has five values in the form of  $[k_1 \ k_2 \ p_1 \ p_2 \ k_3]$  where  $k_1, k_2, k_3$  indicates radial distortion and  $p_1, p_2$  indicates tangential distortion values.

The Rotation Vector depicts the rotation axis of images and the length of the vector encodes the rotation's angle in radians.

The Translation Vector depicts the values of translation of images i.e. the required shift in pixels values

Gdy mamy te wartości, możemy ich użyć w funkcji *cv2.getOptimalNewCameraMatrix()*, której wyniki będą użyte w „undistortion” zdjęć za pomocą funkcji *cv2.undistort()*. Wszystko jest opisane na stronie [https://docs.opencv.org/4.x/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html).

# TEST

```
import glob
images = glob.glob(r"C:\SEM6\PCPO\p9\images_2\\*.png")
results_folder = r"C:\SEM6\PCPO\p9\images_2\undistorted\\"
# images = glob.glob(r"C:\SEM6\PCPO\p9\images\\*.jpg")
# results_folder = r"C:\SEM6\PCPO\p9\images\undistorted\\"
images
9] ✓ 0.0s
['C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (1).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (10).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (11).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (2).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (3).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (4).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (5).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (6).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (7).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (8).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image (9).png',
 'C:\\SEM6\\PCPO\\p9\\images_2\\MicrosoftTeams-image.png']
```

```
objpoints = []
imgpoints = []
for img_path in images:
    pixel_coords_4_points = []
    our, detected, size = get_corresponding_points(img_path, scale_percent=70, gridsize=100, squares_w=15, squares_h=23)
    objpoints.append(our)
    imgpoints.append(detected)
1] ✓ 2m 39.2s
```

```
for array in objpoints:
    print(array.shape)
for array in imgpoints:
    print(array.shape)
1] ✓ 0.0s
(166, 3)
(145, 3)
(155, 3)
(142, 3)
(100, 3)
(95, 3)
(75, 3)
(119, 3)
(144, 3)
(123, 3)
(141, 3)
(84, 3)
(166, 1, 2)
(145, 1, 2)
(155, 1, 2)
(142, 1, 2)
(100, 1, 2)
(95, 1, 2)
(75, 1, 2)
(119, 1, 2)
(144, 1, 2)
(123, 1, 2)
(141, 1, 2)
(84, 1, 2)
```

```

ret, cameraMatrix, dist, rvecs, tvecs = cv2.calibrateCamera(objectPoints=objpoints, imagePoints=imgpoints, imageSize=size, cameraMatrix=None, distCoeffs=None)
ret, cameraMatrix, dist, rvecs, tvecs
] ✓ 0.1s

(3.5588502364509678,
array([[1.11688774e+03, 0.0000000e+00, 6.19654247e+02],
       [0.0000000e+00, 1.11131557e+03, 6.36953524e+02],
       [0.0000000e+00, 0.0000000e+00, 1.0000000e+00]]),
array([[-0.22454995, 0.4165418 , 0.00621792, -0.00659081, -0.29873302]]),
(array([[ 0.38897692],
       [ 0.15845153],
       [-0.0050568 ]]),
array([[-0.28136919],
       [-0.91968708],
       [ 0.14474734]]),
array([[ 0.2141417 ],
       [ 0.69949942],
       [ 0.14601798]]),
array([[ 0.287866007],
       [ 0.63458022],
       [ 0.05981014]]),
array([[ 0.18452532],
       [-0.555585404],
       [-0.08277893]]),
array([[-0.18833798],
       [-0.01282422],
       [-0.07958083]]),
array([[ 0.00964982],
       [-0.75938252],
...
      [-16.30898967],
      [154.700839 ]]),
array([[-53.96956374],
      [-34.29186421],
      [195.91561463]]))

```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)..

```

for img_path in images:
    img = cv2.imread(img_path)

    # scale_percent = 15
    # w = int(img.shape[1] * scale_percent / 100)
    # h = int(img.shape[0] * scale_percent / 100)
    h, w = img.shape[:2]

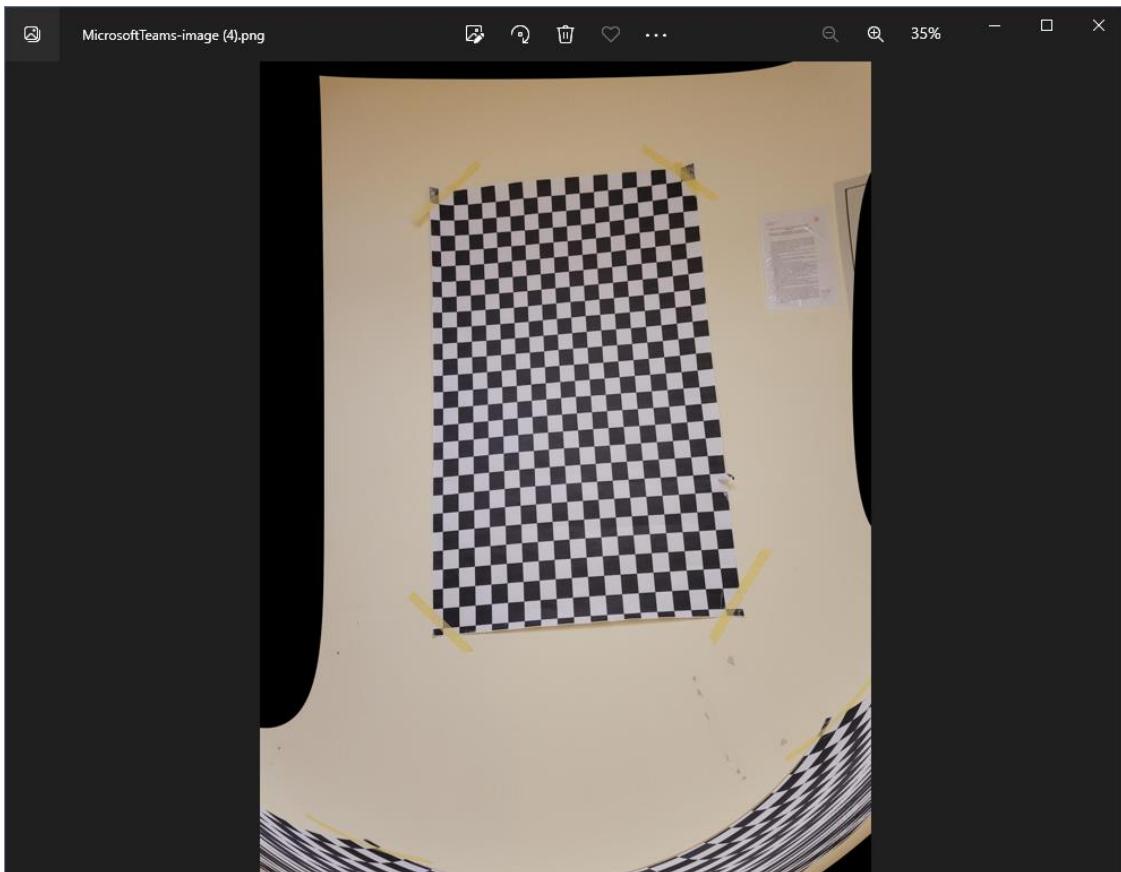
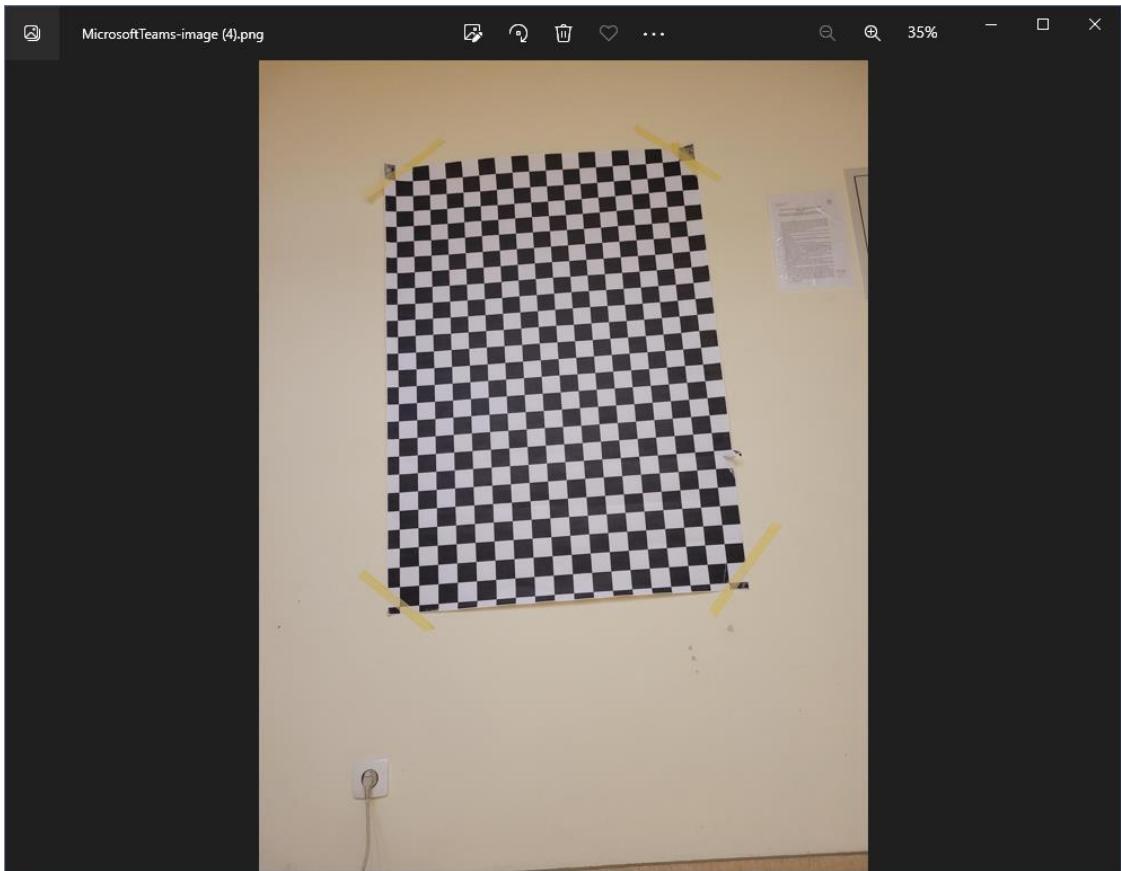
    newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist, (w, h), 1, (w, h))

    # undistort
    dst = cv2.undistort(img, cameraMatrix, dist, None, newCameraMatrix)

    # # crop the image
    # x, y, w, h = roi
    # dst = dst[y : y + h, x : x + w]

    name = img_path[img_path.rfind('\\')+1 : ]
    cv2.imwrite(results_folder + name, dst)
]
```

✓ 1.0s



Oprócz tego, przygotowałem kod do kalibracji kamery (folder `cameraCalibration`), który używa innych funkcji z opencv (m.in. `cv2.findChessboardCorners`). Kod w `getImages.py` pozwala na zrobienie screenshotów z obrazów rejestrowanych przez kamerkę internetową. Należy zrobić kilkanaście screenshotów trzymanej sachownicy w różnych pozycjach i orientacjach. Kod w `calibration.py` jest odpowiedzialny za wzięcie tych screenshotów do analizy, wykonanie kalibracji kamery i zapisanie zundistortowanych wyników.