Sprawozdanie – Sygnały i Obrazy Cyfrowe

Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Kierunek Informatyczne Systemy Automatyki

Wykonał: Paweł Zimoląg

Nr albumu:

Semestr zimowy 2022/2023

Kod zajęć:

Data zajęć:

Godzina zajęć:

Prowadzący:

Spis treści

Zadanie 1

Metody próbkowania sygnałów i obrazów – Aliasing 2D.

Zadanie polega na odtworzeniu zjawiska aliasingu występującego dla obiektów ruchomych z wykorzystaniem obracającego się śmigła oraz sensora o odczycie sekwencyjnym.

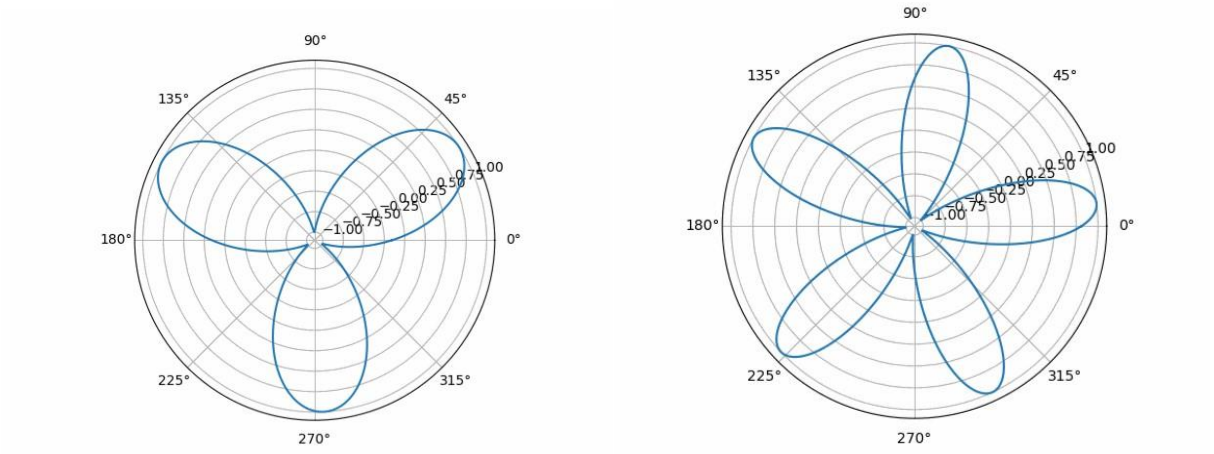
1. **Sekwencja obrazów.**

Wygenerować sekwencję M = 64 obrazów przedstawiających kręcące się śmigło z n = 3, 5 łopatkami. Np. z pomocą wykresu funkcji:

$$f(x) = sin\left(3x + \frac{m\pi}{10}\right), m = -\frac{M}{2}, \ldots, \frac{M}{2}$$

Wykreślonej we współrzędnych biegunowych.

Zrzut ekranu 1 pokazuje wygenerowane śmigła we współrzędnych biegunowych zapisane w formacie GIF. Program również zapisuje pliki mp4.



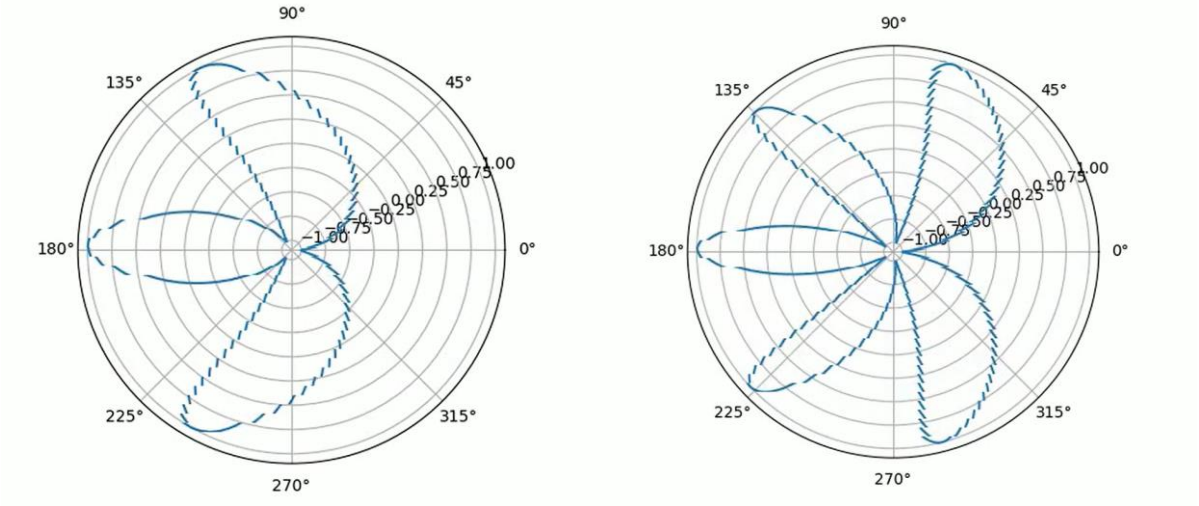
Zrzut ekranu 1

Obydwie animacje znajdują się w załączonych plikach.

2. **Sensor**

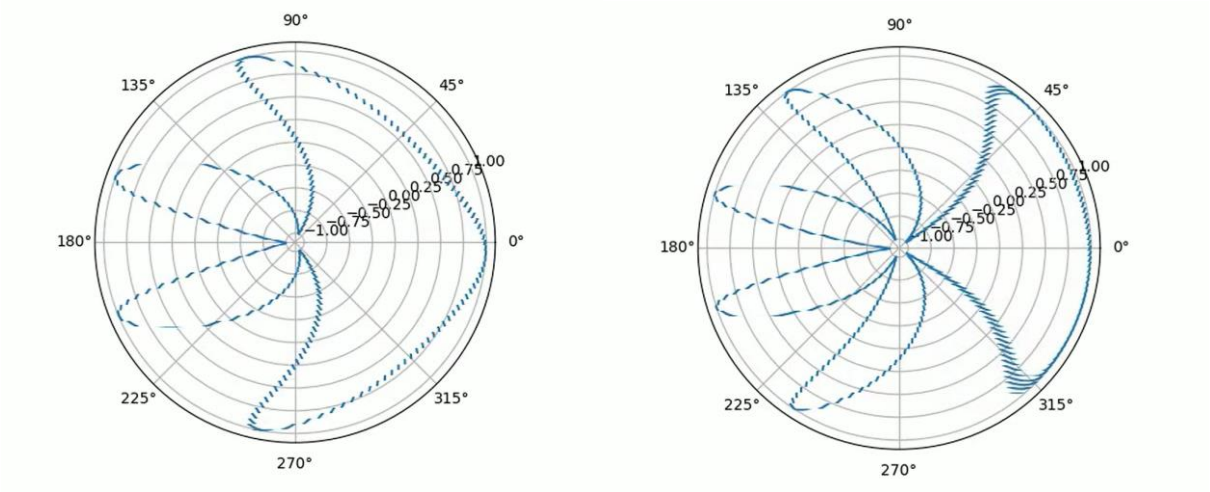
Zakładamy, że sensor ma rozdzielczość 256 na 256 pikseli. Sprawdzić szybkość sensora przyjmując, że w trakcie rejestrowania każdego obrazu sensor jest w stanie od czytać l = 1, ..., 16 linii. Należy następnie utworzyć film składający się z K = 256/l klatek uruchamiając sekwencję obrazów śmigła ”w kółko”.

Efekt końcowy dla 3 oraz 5 łopat przy odczytywaniu 8 linii.



Zrzut ekranu 2

Efekt końcowy dla 3 oraz 5 łopat przy odczytywaniu 5 linii.



Zrzut ekranu 3

Wszystkie pliki mp4 znajdują się w załączniku.

Zadanie ma na celu zasymulowanie działania zjawiska aliasingu. Podczas wykonywania zdjęcia sensor CMOS rejestruje obraz od góry do dołu. W przypadku obiektów ruszających się z większą prędkością niż sensor jest w stanie uchwycić, następuje ww. zjawisko.

Rozwiązaniem może być szybsze działanie sensora. Im więcej linii odczytuje sensor, tym mniejsze „rozmazanie” śmigła. Innym rozwiązaniem problemu może być użycie aparatu z sensorem, który ma migawkę globalna (ang. global shutter), zamiast postępowej (rolling shutter), która zapisuje obraz na matrycy całościowo, a każda klatka skanowana jest naraz.

```
y_val = np.sin(prop_nr * x_val + m * np.pi / 10)
plt.polar(x_val, y_val)
writer.grab_frame()
```

Zrzut ekranu 4

Fragment kodu, który na początku wyświetla śmigło za pomocą biblioteki matplotlib, a następnie zapisuje klatkę do końcowego filmu.

Zadanie 2

Zastosowania interpolacji – Demozaikowanie

Zadanie polega na (i) zasymulowaniu działania filtrów kolorów (Bayer CFA oraz Fuji XTrans) znajdujących się na matrycach CMOS oraz (i) opracowaniu i implementacji algorytmu demozaikowania w oparciu o funkcje interpolacji Π, Λ oraz f. Keysa.

1. **Za pomocą dowolnego schematu interpolacji, należy zaimplementować algorytmy demozaikowania, tj. odtworzyć obraz RGB z informacji zarejestrowanej przez matrycę CMOS z filtrem kolorów CFA według schematu:**
   a) **Bayera (2x2)**

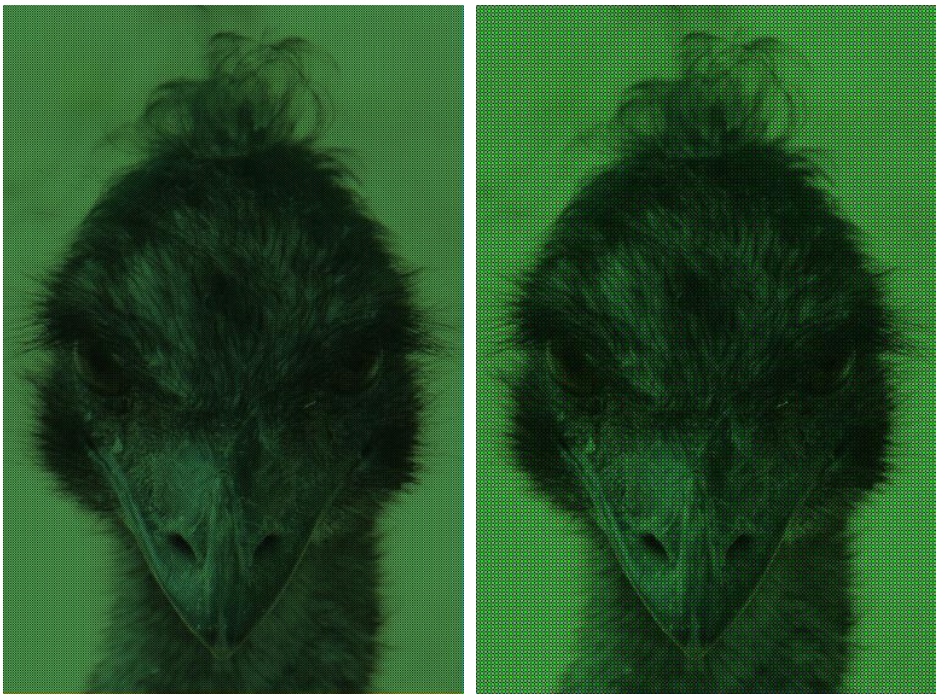
$$\begin{bmatrix} G & R \\ B & G \end{bmatrix}$$

   b) **Firmy Fuji (X-Trans 6x6)**

$$\begin{bmatrix} G & B & R & G & R & B \\ R & G & G & B & G & G \\ B & G & G & R & G & G \\ G & R & B & G & B & R \\ B & G & G & R & G & G \\ R & G & G & B & G & G \end{bmatrix}$$

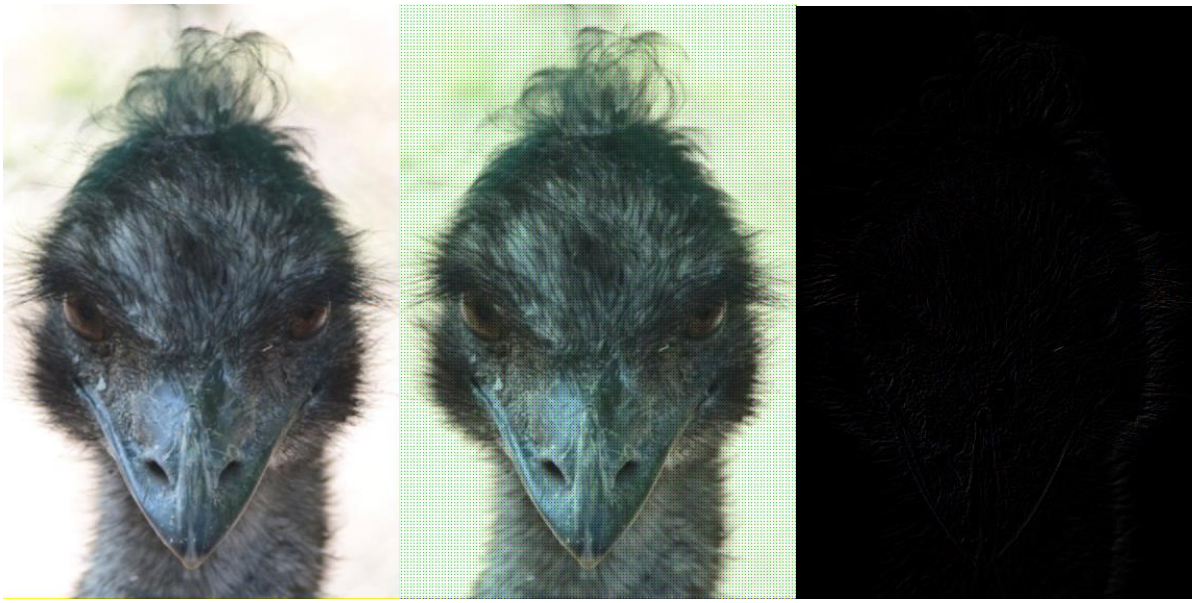
   Wskazać najlepszy algorytmy i uzasadnić wybór.

Aby wykonać mozaikę należało usunąć wartości kolorów pikseli innych niż ten, na który wskazywał dany filtr. Mozaika wykonana schematem Bayera oraz mozaika wykonana schematem X-Trans (Zrzut ekranu 5).



Zrzut ekranu 5

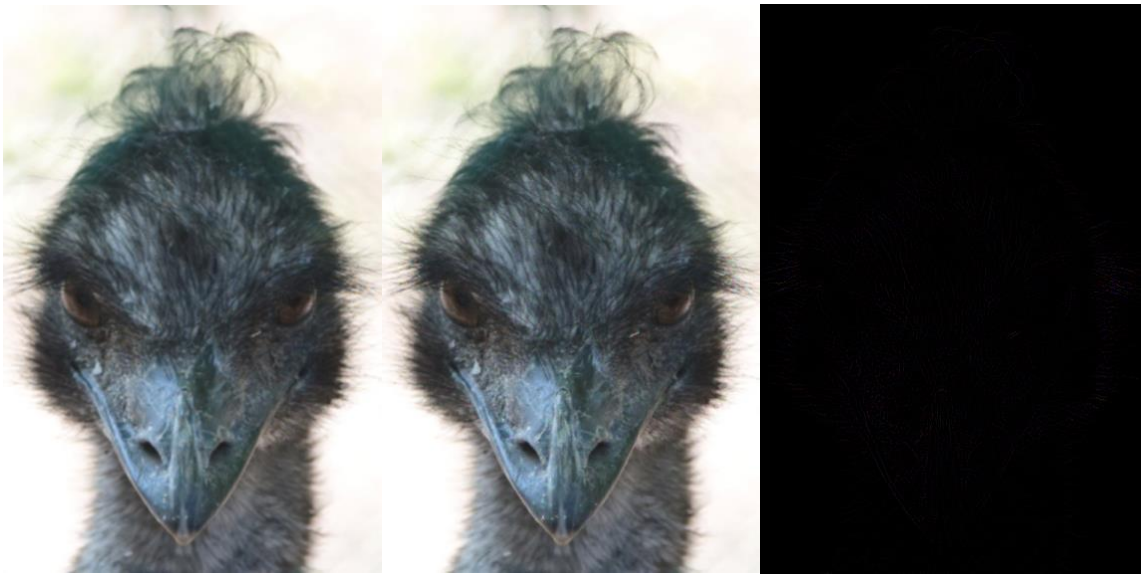
Do zdemozaikowania obrazka użyłem interpolacji najbliższego sąsiada (Π). Efekt po interpolacji na pliku z filtrem Bayera oraz pliku z filtrem X-Trans (Zrzut ekranu 6).



Zrzut ekranu 6 – demozaika filtru Bayera interpolacją najbliższego sąsiada, dwuliniową oraz różnica od oryginału

Interpolacja najbliższym sąsiadem na mozaice X-Trans sprawia, że obraz jest bardziej zielony. Wynika to z rozłożenia pikseli przy ww. filtrze (wzór na stronie 6).

Efekt po interpolacji dwuliniowej (Λ) na pliku z filtrem Bayera oraz pliku z filtrem X-Trans (Zrzut ekranu 7).



Zrzut ekranu 7 - demozaika filtru X-Trans interpolacją najbliższego sąsiada, dwuliniową oraz różnica od oryginału

Porównując efekt po interpolacji metodą najbliższego sąsiada oraz dwuliniowej możemy zauważyć różnicę w ostrości detali. Na pierwszy rzut oka obrazki różnią się nieznacznie, lecz gdy zaczniemy przyglądać się przybliżeniu możemy zwrócić uwagę na różne odcienie kolorów pikseli, które powodują lepszą ostrość obrazu. W aparatach podczas robienia zdjęć obramowanie obrazu jest często usuwane, aby przyspieszyć proces interpolacji zdjęć.

Moim zdaniem interpolacja dwuliniowa sprawdza się lepiej, ponieważ pobiera informacje o kolorach z większej ilości pikseli. Efektem jest wyraźniejszy obraz oraz ładniejsze odcienie. Minusem zaś jest dłuższy czas obróbki zdjęcia.

Wartości PSNR dla obrazków po interpolacji:

Bayer – 36,518 db

X-Trans – 36,538 db

Zadanie miało na celu zasymulowanie działania interpolacji na matrycach CMOS oraz pokazanie, w jaki sposób się to dzieje. Podczas robienia zdjęcia w rzeczywistości obraz jest czarno-biały (piksele posiadają tylko informacje o odcieniu szarego), a następnie nakładany jest filtr (np. Bayer lub X-Trans), który jest interpolowany w celu wydobycia koloru, jaki powinien znajdować się na danym pikselu.

```
if get_mosaic_pixel_type_fn(x, y) == 0:
    r_value = r_val0
elif get_mosaic_pixel_type_fn(x, y) == 1:
    g_value = g_val0
elif get_mosaic_pixel_type_fn(x, y) == 2:
    b_value = b_val0
```

Zrzut ekranu 8

Powyższy fragment kodu przedstawia moje spostrzeżenie dotyczące demozaikowania obydwu filtrów. Podczas „szukania” koloru każdego piksela, nie trzeba sumować oraz uśredniać wartości koloru wg schematu, ponieważ oryginalnie ma zapisany konkretny odcień tego koloru. Np. piksel [1,1] w schemacie Bayera jest zielony, więc odcień zieleni jest znany i nie trzeba go szukać. Daje to widoczne po przybliżeniu efekty.

Zadanie 3

Zastosowania interpolacji - skalowanie i obroty obrazów rastrowych

1. **Wybrać obraz (mapę bitową) o rozmiarze N N dla N = 512**
2. **Posługując się interpolacją zastosowaną do kolumn i wierszy obrazu:**
   a) **Pomniejszyć (a następnie powiększyć ją o wybraną krotność)**
   b) **obrócić ją o wybrany kąt**
3. **Punkty 1 i 2 wykonać posługując się funkcjami interpolującymi:**

   (a) $B_0(x) = I_{[-1/2,1/2]}(x)$
   (b) $B_1(x) = B_0(x) * B_0(x) = I_{[-1,1]} \cdot (1 - |x|)$
   (c) funkcją interpolującą Keysa z wybranym odpowiednio parametrem α

4. **Porównać czasy działania algorytmów z punktów 1-3**
5. **Porównać jakość skalowanych i obracanych obrazów:**
   a) **Subiektywnie (opisowo)**
   b) **Obiektywnie (proponując miarę jakości)**

Ad. 1:

Obraz 1 ma rozdzielczość 512x512.

Obraz 1

Ad. 2 i 3:

a)

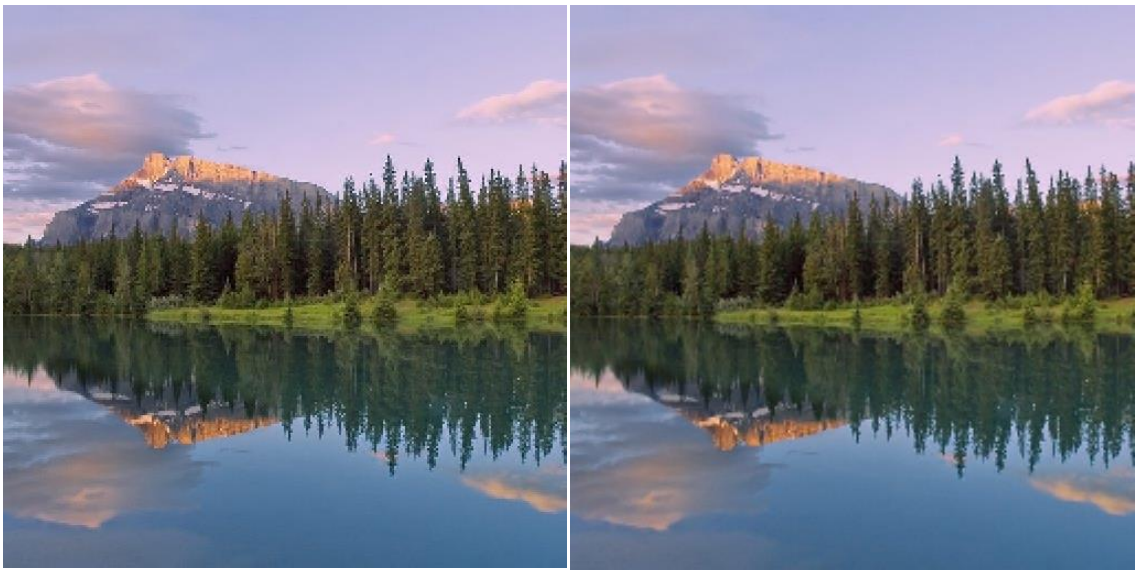
Poniżej znajdują się pomniejszone dwukrotnie obrazki zinterpolowane na każdy sposób:


Obraz 2 – interpolacja najbliższym sąsiadem oraz dwuliniowa

Obraz 3 – interpolacja dwusześcienna

Najmniej dokładny jest obraz 2, na którym została użyta interpolacja najbliższego sąsiada. Obraz 3 ma łagodniejsze, najbardziej rozmyte krawędzie. Obraz 4 jest wyraźniejszy od poprzedniego, ma ostrzejsze krawędzie, ale po przybliżeniu bardziej widoczne są piksele.

Następnie obrazki zostały powiększone dwukrotnie (przywrócone do rozmiaru 512x512).



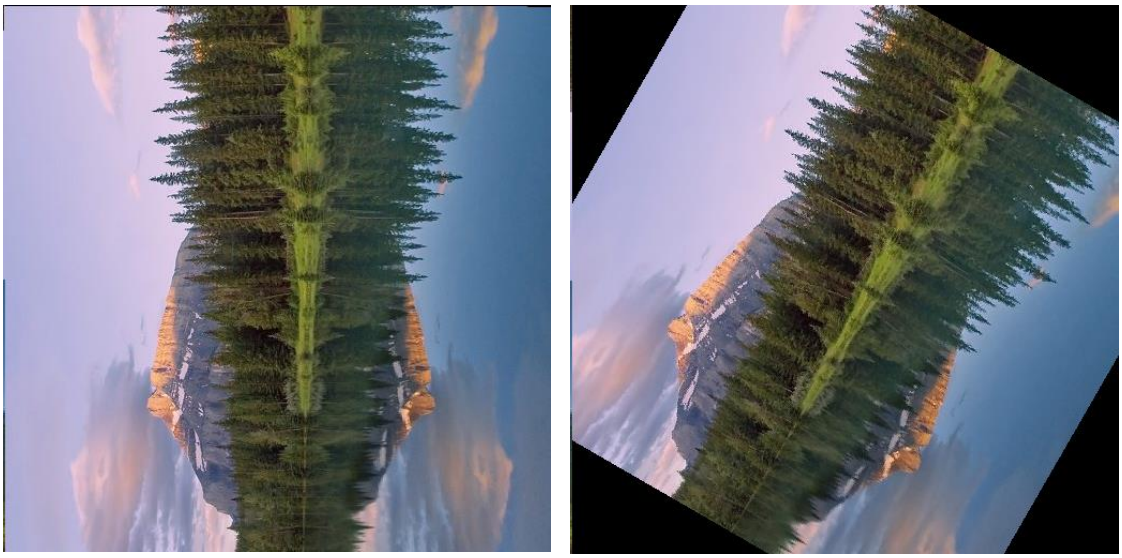
Obraz 4 – interpolacja najbliższym sąsiadem oraz dwuliniowa

Obraz 5 – interpolacja dwusześcienna

Zdecydowanie największą różnicę w jakości można dostrzec na pierwszym obrazie. Interpolacja najbliższego sąsiada daje najgorszy efekt końcowy. Pomiędzy efektem po interpolacji dwuliniowej oraz dwusześciennej ma znaczących różnic. Tak jak przy pomniejszaniu, interpolacja dwuliniowa daje efekt bardziej wygładzonych krawędzi, zaś dwusześcienna – ostrzejszych.

b)

Poniżej znajdują się dwa obrazki (Obraz 6) – obrócony o kąt 90 stopni oraz obrócony o kąt 60 stopni w lewo.



Obraz 6 – obrócenie o 90 oraz 60 stopni

Ad. 4:

Zdecydowanie najszybciej działa algorytm najbliższego sąsiada. Wymaga on najmniej obliczeń, przez co najsprawniej dokonuje interpolacji. Dwuliniowa interpolacja trwa dłużej niż najbliższego sąsiada, lecz wciąż krócej niż dwusześcienna.

Przykładowy pomiar czasu podczas trzykrotnego pomniejszania, a następnie trzykrotnego powiększania z wykorzystaniem ww. trzech algorytmów:

Najbliższy sąsiad – 1.5-2 sekundy

Dwuliniowa – ok. 2.5 sekundy

Dwusześcienna – ok. 4 sekund.

Przy niewielkim zmienianiu wielkości zdjęcia czas trwania jest przybliżony, lecz ma znaczenie przy powiększaniu np. 10-krotnym.

Ad. 5:

Jakość skalowanych obrazów porównałem w „Ad. 2 i 3” pod obrazami.

Wartości PSNR dla interpolacji po dwukrotnym zmniejszeniu, a następnie powiększeniu (w porównaniu z oryginałem):

Najbliższy sąsiad – 34,481 dB

Dwuliniowa – 35,63 dB

Dwusześcienna – 34,853 dB

```
scale_image(Image.open('interpolation.png'), -2, 0).save('down_Nearest.png')
scale_image(Image.open('down_Nearest.png'), 2, 0).save('down_up_Nearest.png')
```

Zrzut ekranu 9

Powyższy fragment kodu przedstawia sposób, w jaki sposób wywołuje się funkcję, służącą do dowolnego powiększania i pomniejszania obrazka, używając jednej z trzech zaprogramowanych interpolacji.

Zadanie 4

Aproksymacja – wygładzanie i redukcja zakłóceń

Na wybranym obrazie N x N dokonać redukcji zakłóceń za pomocą:

1) **Filtru splotowego (ruchoma średnia) o wybranej długości i wybranym kształcie funkcji jądra (np. prostokątne, trójkątne, Keysa)**
2) **Filtru medianowego**
3) **Filtru bilateralnego o wybranej długości i kształcie jądra**

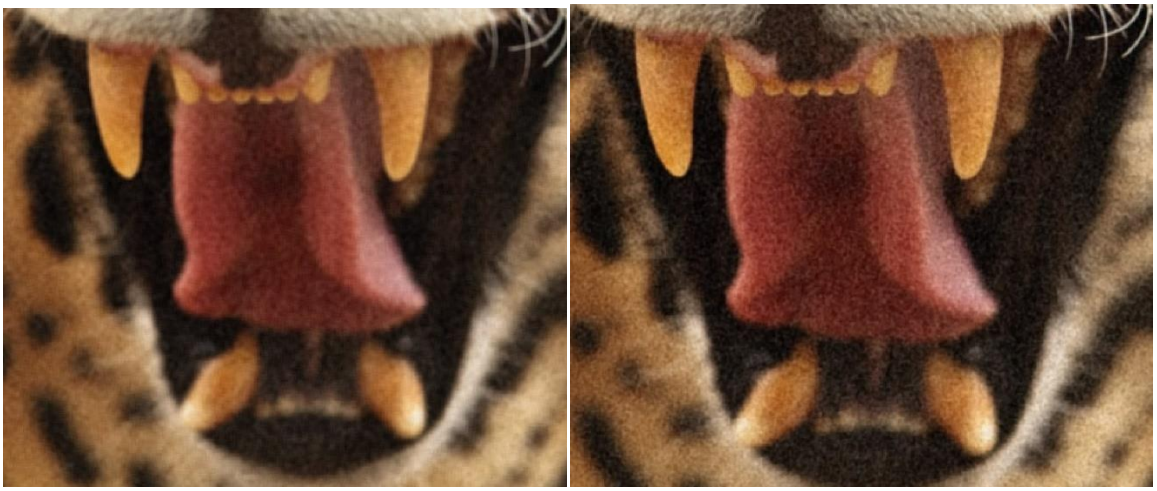
W każdym z przypadków porównać obraz zakłócony z oryginałem. Wyznaczyć błąd średniokwadratowy. Wybrać najlepsze parametry filtrów i porównać je ze sobą.

Ad. 1

Do wygładzenia za pomocą filtru splotowego użyłem 2 kerneli – jednorodnego oraz trójkątnego (Zrzut ekranu 10).

```
kernel1 = np.ones((5, 5), np.float32) / 25  #jednorodny
kernel2 = np.array([[0, 0, 1, 0, 0],
                    [0, 2, 2, 2, 0],
                    [1, 2, 5, 2, 1],
                    [0, 2, 2, 2, 0],
                    [0, 0, 1, 0, 0]])/25 #trojkatny
```

Zrzut ekranu 10



Zrzut ekranu 11 – różnica pomiędzy obrazami z zastosowanym filtrem splotowym jednorodnym oraz trójkątnym

Zrzut ekranu 12 – oryginalnym zdjęcie bez zakłóceń dla porównania

Błąd średniokwadratowy dla jednorodnego kernela: 51.0582971572876.

Błąd średniokwadratowy dla trójkątnego kernela: 59.447127978006996.

Ad. 2

Zrzut ekranu 13 – porównanie obrazu po odszumieniu filtrem medianowym a oryginalnym zdjęciem

Błąd średniokwadratowy: 60.451873143514.

Ad. 3

Zrzut ekranu 14 – porównanie obrazu po odszumieniu za pomocą filtru bilateralnego a oryginalnym zdjęciem

Błąd średniokwadratowy: 81.03747844696045.

```
if x + j in range(0, w) and y + i in range(0, h):
    r, g, b = img.getpixel((x+j, y+i))
    r_val.append(r)
    g_val.append(g)
    b_val.append(b)
```

Zrzut ekranu 15

Powyższy fragment kodu dodaje sprawdza, czy szukany piksel nie wychodzi poza obręb obrazka, a następnie dodaje go do list.