Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Sprawozdanie

Sygnały i obrazy cyfrowe

Kateryna Fedoruk

272609

Gr. 7 - czwartek 13:15-15:00

1. Interpolacja sygnałów 1D

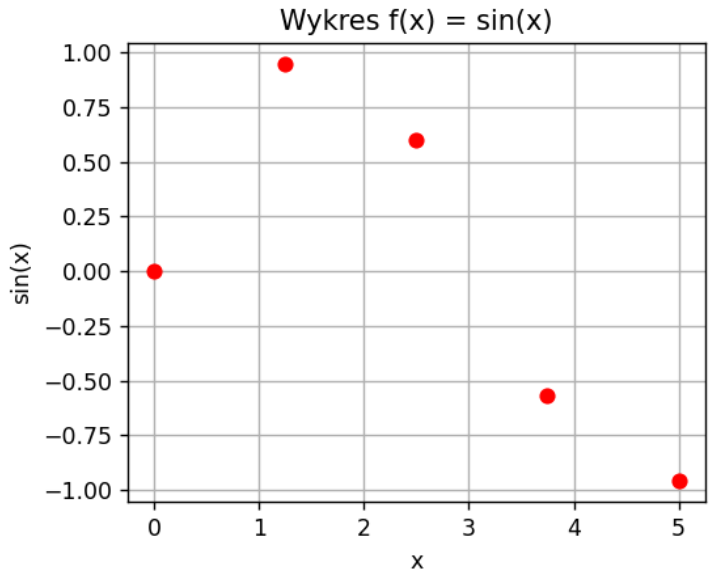
1.1 Cel ćwiczenia

Zadanie polega na przeprowadzeniu interpolacji wybranego sygnału (1D) stosując różne interpolacji (najbliższy sąsiad, liniowa, kwadratowa, wielomianami 3 stopnia). I porównanie wyników z oryginalnym sygnałem. Celem zadania jest samodzielne zaimplementowanie metod interpolacji, nie korzystając z gotowych funkcji.

1.2 Prezentacja wyników

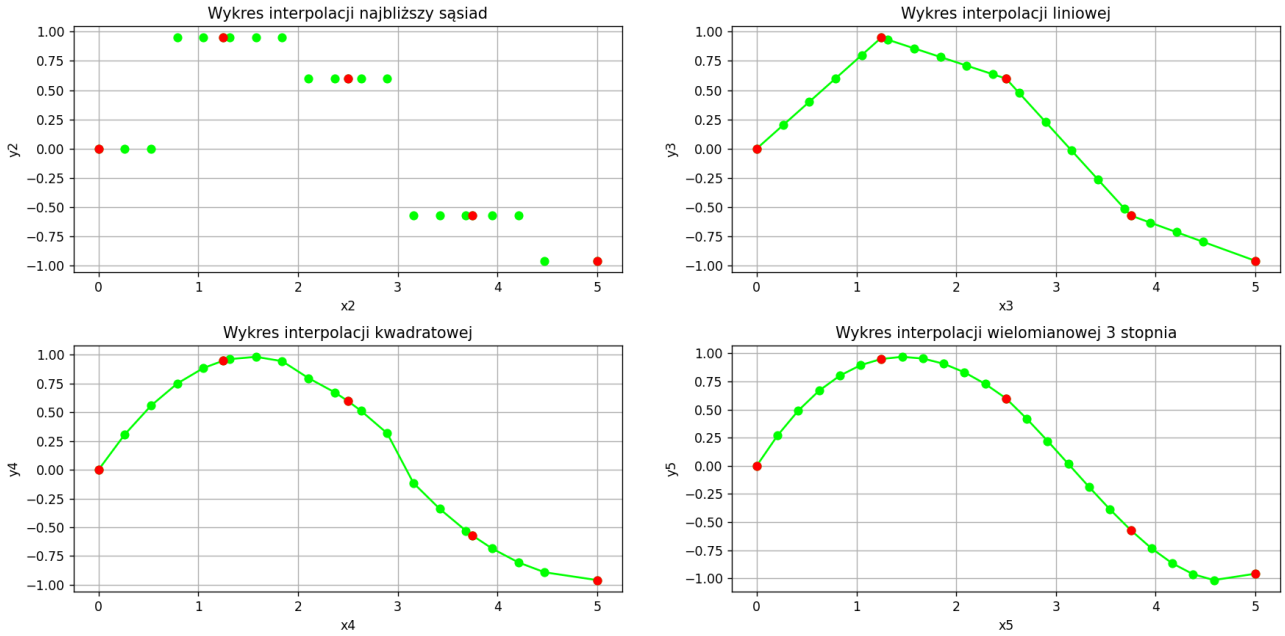
Kluczowe fragmenty kodu odnoszą się do funkcji, które odpowiadają za konkretny rodzaj interpolacji lub operację matematyczną wykorzystywaną w procesie interpolacji oraz do fragmentów odpowiedzialnych za wizualizację wyników. Wizualizacja wyników opiera się na wykorzystaniu biblioteki Matplotlib, konkretnie jej modułu pyplot natomiast funkcje korzystają głównie z biblioteki NumPy, która jest podstawową biblioteką do obliczeń naukowych w języku Python.

- Przygotowanie danych i wykresu funkcji oryginalnej:



W tej sekcji tworzone są punkty dla funkcji oryginalnej sin(x) i rysowany jest wykres tych punktów. Punkty (x1, y1) są używane jako baza dla dalszej interpolacji.

- Interpolacja



W tej sekcji punkty są interpolowane stosując różne metody (najbliższy sąsiad, liniowa, kwadratowa, wielomianami 3 stopnia) .W części, w której punkty są interpolowane metodą najbliższego sąsiada dla każdego punktu z x2 wybierana jest najbliższa wartość y z zestawu oryginalnego (x1, y1). W pozostałych interpolacjach używane są najbliższe punkty do utworzenia prostej/paraboli/wykresu wielomianu przechodzącego przez te punkty dla każdego punktu z x3/x4/x5. Ilośc najbliższych punktów różni się w zalezności od wybranej metody. Następnie rysowane są wykresy łączące te punkty.

- Przykłady wykorzystanych funkcji

1. Funkcja najblizsze_x_kwadratowa:

```
def najblizsze_x_kwadratowa(macierz_x, nowy_x):
    odleglosci = sorted([(x, abs(nowy_x - x)) for x in macierz_x], key=lambda x: x[1])[:3]
    najblizsze_x = [x[0] for x in odleglosci]
    return najblizsze_x
```

Funkcja wybiera trzy najbliższe punkty x1 do danego punktu x4, które są używane do interpolacji kwadratowej.

2. Funkcja przypisz_y_liniowa:

```
def przypisz_y_liniowa(x1, y1, x3):
    segment = np.digitize(x3, x1)  # do której sekcji (segmentu) na wykresie należy punkt x3
    if segment == 0:
        y3 = y1[0]
    elif segment == len(x1):
        y3 = y1[-1]
    else:
        x1_lower = x1[segment - 1]
        x1_upper = x1[segment]
        y1_lower = y1[segment - 1]
        y1_upper = y1[segment]
        y3 = y1_lower + (x3 - x1_lower) * (y1_upper - y1_lower) / (x1_upper - x1_lower)
    return y3
```

Funkcja oblicza wartość interpolacji liniowej dla danego punktu x3. W tym celu znajduje dwa sąsiednie punkty z x1, między którymi znajduje się x3, a następnie oblicza wartość y3 na podstawie równania prostej przechodzącej przez te punkty.

1.3 MSE

- Funkcja MSE(fragment kodu)

```
# MSE
def mse(y_tr, y_pred):
    error = y_tr - y_pred
    squared_error = error ** 2
    mean_squared_error = np.mean(squared_error)
    return mean_squared_error
```

W przedstawionej wyżej funkcji mse, najpierw obliczana jest różnica między wartościami rzeczywistymi (y_tr) a przewidywanymi (y_pred). Następnie te różnice są podnoszone do kwadratu, a średnia wartość tych kwadratów jest obliczana jako wynik końcowy.

- Porównanie wyników

```
Metoda: najbliższy sąsiad, MSE: 0.059665220082788635, Czas: 0.0010232925415039062s
Metoda: liniowa, MSE: 0.010070151665934134, Czas: 0.0010309219360351562s
Metoda: kwadratowa, MSE: 0.0026132520699176463, Czas: 0.0025572776794433594s
Metoda: wielomianowa 3 stopnia, MSE: 0.0012851835707877414, Czas: 0.0077364444732666016s
```

Na podstawie podanych wyników widzimy, że interpolacja wielomianowa 3 stopnia ma najniższy błąd średniokwadratowy (MSE) spośród wszystkich metod, co wskazuje na najwyższą dokładność tej metody. Niższe MSE oznacza, że przewidywane wartości są bliższe rzeczywistym wartościom funkcji sin(x). Chociaż metoda ta ma najdłuższy czas wykonania w porównaniu z innymi metodami, różnica w czasie nie jest znacząca (ok. 0.0077 sekundy), zwłaszcza biorąc pod uwagę wzrost dokładności. Interpolacja kwadratowa również zapewnia dobre wyniki (MSE: 0.0026) z nieco krótszym czasem wykonania, ale jej dokładność jest niższa niż interpolacja wielomianowa 3 stopnia. Interpolacje najbliższy sąsiad i liniowa mają znacznie wyższe wartości MSE, co wskazuje na mniejszą dokładność.

Podsumowując, mimo iż interpolacja wielomianowa 3 stopnia wymaga nieco więcej czasu, oferuje znacznie lepszą dokładność, co czyni ją najlepszym wyborem w kontekście tych danych.

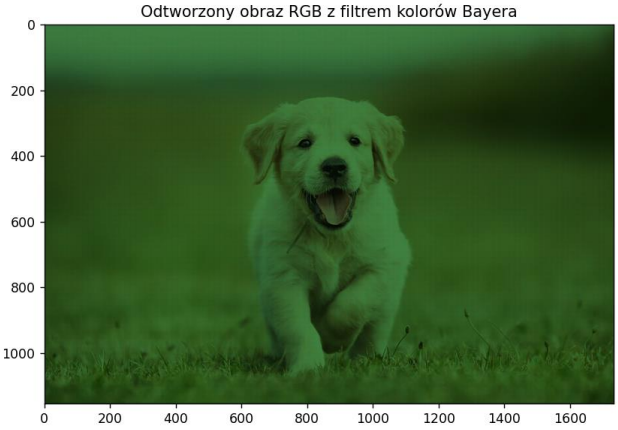
2. Zastosowania interpolacji – Demozaikowanie

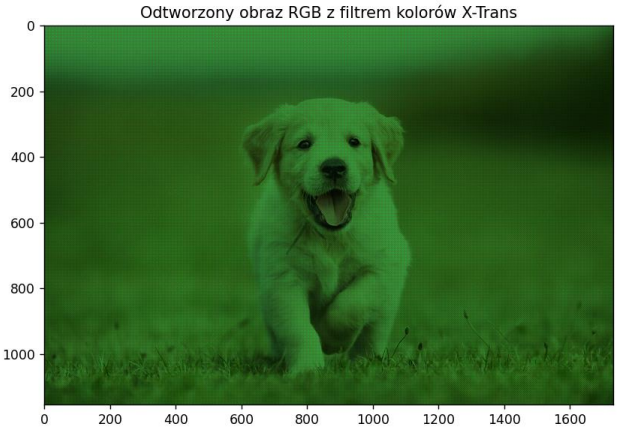
2.1 Cel ćwiczenia

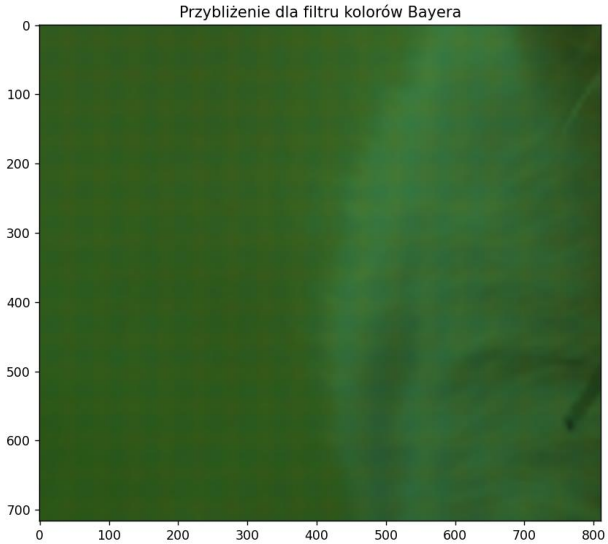
Zadanie polega na (i) zasymulowaniu działania filtrów kolorów (Bayer CFA oraz Fuji XTrans) znajdujących się na matrycach CMOS oraz (ii) opracowaniu i implementacji algorytmu demozaikowania w oparciu o funkcje interpolacji Π, Λ oraz f. Keysa.

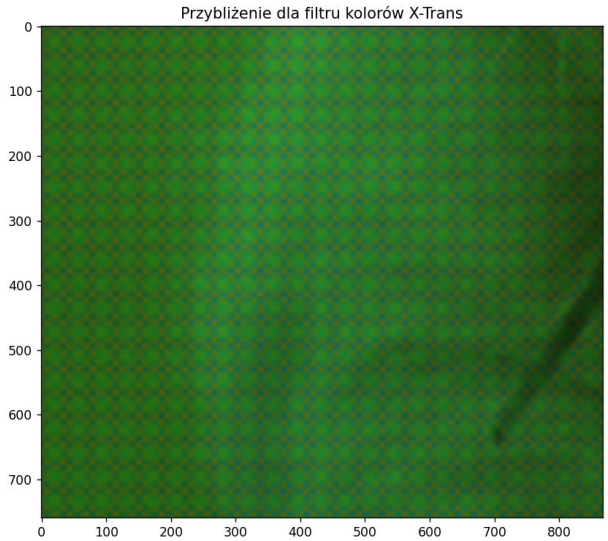
Obraz wejściowy wykorzystany w tym ćwiczeniu:

2.2 Zastosowanie masek









Wyżej są przedstawione wyniki nałożenia odpowiednich masek filtrów kolorów na obraz wejściowy. W tym celu w programie zostały wywoływane funkcje apply_bayer_mask i

apply_fuji_xtrans_mask. Maski te symulują działanie rzeczywistych sensorów obrazu w kamerach cyfrowych, które wykorzystują matrycę Bayera lub matrycę Fuji X-Trans do rejestrowania kolorów.

Fragment kodu zawierający jedną z wyżej wymienionych funkcji - apply_bayer_mask:

```python
def apply_bayer_mask(image, height, width):
    bayer_image = np.zeros((height, width, 3), dtype=np.uint8)
    for i in range(height):
        for j in range(width):
            if i % 2 == 0 and j % 2 == 1:  # Piksele czerwone
                bayer_image[i, j, 0] = image[i, j, 2]  # R
            elif i % 2 == 1 and j % 2 == 0:  # Piksele niebieskie
                bayer_image[i, j, 2] = image[i, j, 0]  # B
            else:  # Piksele zielone
                bayer_image[i, j, 1] = image[i, j, 1]  # G
    return bayer_image
```

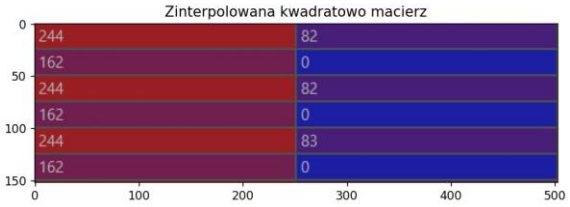
2.3 Wyniki interpolacji

Interpolacja metodą najbliższy sąsiad dla obrazu RGB z filtrem kolorów Bayera:

Interpolacja metodą liniową dla obrazu RGB z filtrem kolorów Bayera:

Interpolacja metodą najbliższy sąsiad dla obrazu RGB z filtrem kolorów X-Trans:

Nieudana interpolacja metodą kwadratową(+ najbliższy sąsiad) dla obrazu RGB z filtrem kolorów Bayera:
W tym przypadku obraz został częściowo zinterpolowany metodą kwadratową:



A później wartośći niezinterpolowane zoswtały zinterpolowane metodą - najbliższy sąsiad(jako eksperyment). Na końcu żeby uzyskać przybliżony do wejściowego wynik zastosowałąm również normalizację koloru zielonego:

2.4 MSE

```
MSE for Bayer nearest neighbor: 72.82403727961547
Execution time for Bayer nearest neighbor: 2.035904884338379
MSE for Bayer linear interpolation: 72.40079606122316
Execution time for Bayer linear interpolation: 12.570056200027466
MSE for Fuji nearest neighbor: 72.55909182640428
Execution time for Fuji nearest neighbor: 88.70830416679382
```

Na podstawie podanych wyników(nie brałam pod uwagę nieudaną interpolację kwadratową) można zauważyć, że metoda najbliższego sąsiada dla filtru Bayera jest znacznie szybsza niż interpolacja liniowa dla tego samego filtru oraz niż metoda najbliższego sąsiada dla filtru Fuji. Czas wykonania rekonstrukcji dla najbliższego sąsiada Bayera wynosi około 2 sekundy, podczas gdy dla interpolacji liniowej Bayera to ponad 12 sekund, a dla najbliższego sąsiada Fuji to niemal 89 sekund.

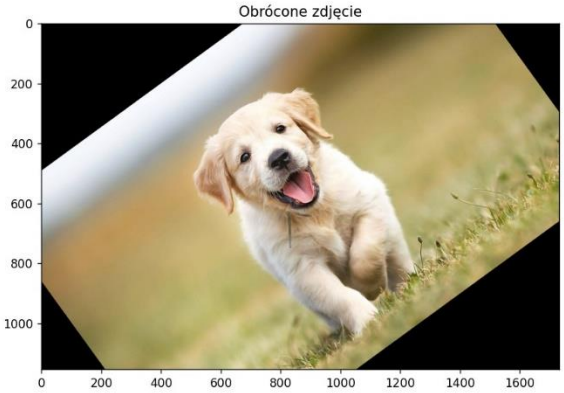
Jeśli chodzi o błąd średniokwadratowy (MSE), wyniki dla obu metod interpolacji dla filtru Bayera są zbliżone i wynoszą odpowiednio około 72.40 dla interpolacji liniowej i nieco wyższy dla Fuji najbliższego sąsiada - około 72.56. To oznacza, że w zakresie jakości rekonstrukcji obrazu

(mierzonej za pomocą MSE), metody dla filtru Bayera nie różnią się znacząco, jednak metoda dla filtru Fuji jest nieco gorsza.

Najlepszym algorytmem pod względem czasu wykonania jest najbliższy sąsiad dla filtru Bayera, który jest również konkurencyjny pod względem MSE. Dlatego, jeżeli priorytetem jest czas, to ta metoda wydaje się być najlepszym wyborem. Jednakże, jeśli jakość obrazu ma kluczowe znaczenie, warto wziąć pod uwagę, że wszystkie metody dają podobne wyniki MSE, więc różnica w jakości może być niewielka – co widać również na zinterpolowanych obrazach.

3. Obracanie obrazu

Na początku zdjęcie zostało obrócone o 36 stopni:



Następnie obrót został powtórzony 10 razy: