Sygnały i obrazy cyfrowe

sprawozdanie

|  |  |
| --- | --- |
|  |  |
| Kierunek | Informatyczne Systemy Automatyki |
| Termin zajęć | Wtorek, 9:15 – 11:00 |
| Prowadzący | Dr inż. Konrad Kluwak |
| Termin oddania sprawozdania | 10.01.2023 r. |
| Autor | Maciej Krawczyk |

[1. Cel ćwiczenia 3](#_Toc124200350)

[2. Opis ćwiczenia 3](#_Toc124200351)

[2.1. Realizacja ćwiczenia 3](#_Toc124200352)

[3. Wstęp teoretyczny 3](#_Toc124200353)

[3.1. Aliasing 3](#_Toc124200354)

[3.2. Demozaikowanie 4](#_Toc124200355)

[3.3. Skalowanie i obracanie 6](#_Toc124200356)

[4. Wyniki algorytmów 7](#_Toc124200357)

[4.1. Aliasing 7](#_Toc124200358)

[4.2. Demozaikowanie 8](#_Toc124200359)

[4.3. Skalowanie i obrażanie 13](#_Toc124200360)

1. Cel ćwiczenia

Celem pierwszego ćwiczenia było odtworzenie zjawiska **aliasingu** z wykorzystaniem obracającego się śmigła oraz sensora o sekwencyjnym odczycie.

Drugie ćwiczenie polegało na symulacji działania filtrów **Bayer CFA** oraz **Fuji X-Trans** oraz implementacji algorytmu **demozaikowania** za pomocą wybranej funkcji interpolacyjnej.

Trzecie ćwiczenie opisywało **skalowanie** i **obracanie** obrazów oraz porównaniu ich jakości po przekształceniach.

1. Opis ćwiczenia
   1. Realizacja ćwiczenia
2. Teoretyczna analiza zadań do wykonania.
3. Przygotowanie do zagadnień teoretycznych w celu zrozumienia działania poszczególnych zjawisk.
4. Przygotowanie odpowiednich algorytmów i ich implementacja w kodzie.
5. Obliczenie miar jakości i przygotowanie porównania.
6. Sporządzenie wniosków.
7. Wstęp teoretyczny
   1. Aliasing

**Aliasing** jest wynikiem błędnego procesu próbkowania obrazu. W naszym konkretnym przypadku powoduje to tzw. rolling shutter, czyli zatrzymanie się obrazu w pewnych jego miejscach, gdyż szybkość ruchu na obrazie jest większa niż częstotliwość próbkowania. Pojawia się to, gdyż obraz przez matrycę sczytywany jest sekwencyjnie, od góry do dołu w zbyt wolnym czasie.



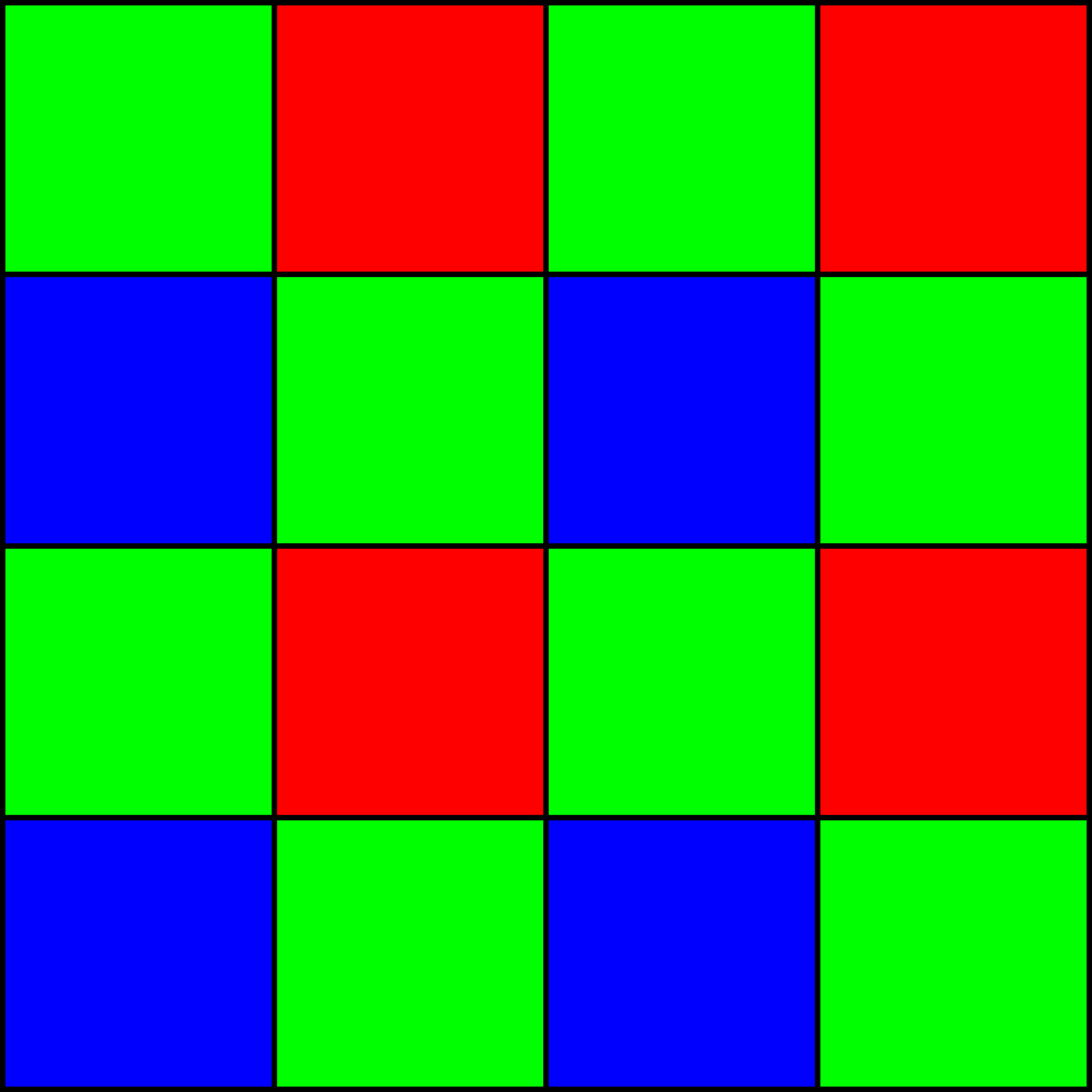
Aby uniknąć takiego zjawiska możemy skorzystać z **filtrów antyaliasingowych**, które są limiterem widma sygnału próbkowanego. Sygnał próbkowany nie powinien być większy niż połowa częstotliwości próbkowania, wtedy w teorii powinniśmy uniknąć takowego zjawiska.

* 1. Demozaikowanie

**Demozaikowanie** polega na **interpolacji** sygnału z podanej mozaiki. Matryce w aparatach przy wykonywaniu zdjęcia zapisują mozaikę w formie odpowiedniego wzoru. Każdy piksel przypisany jest do konkretnego koloru w postaci ustalonego przez twórcę matrycy danego patternu, gdzie najczęściej jest to **Bayer** w różnych jego odmianach.

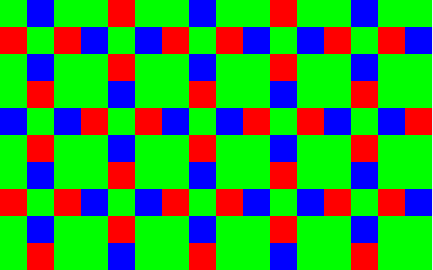
Jednym z przykładów schematu **Bayera** jest:

Która przedstawiona jest w treści zadania. Jak widzimy dominuje tu kolor zielony, którego jest dwukrotnie więcej niż reszty, tj. czerwonego czy niebieskiego.



Tak natomiast wygląda on naniesiony na obraz w przybliżeniu.

Drugim zaprezentowanym filtrem kolorów jest **Fuji X-Trans**, który prezentuje się następująco:

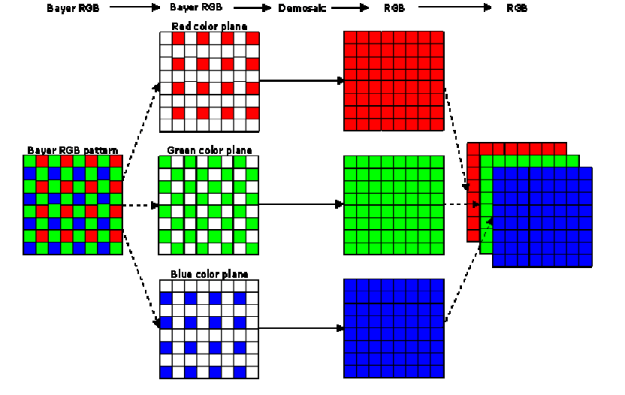


Również widzimy znaczną dominację koloru zielonego.

Wynika to z tego, że kolor zielony oddziaływuje najlepiej na ludzkie oko pod kątem oceniania jakości obrazu. Filtry te używając większej ilości zielonego koloru powodują, że obraz wydaje nam się ostrzejszy.

Tak więc wyglądają zmozaikowane obrazy, tak jak widzie je sensor aparatu. Natomiast abyśmy my zobaczyli obraz tak, jak widzimy go normalnie, np. w skompresowanym formacie JPG, muszą one być one zdemozaikowane, tj. musi nastąpić interpolacja, aby przywrócić właściwe, rzeczywiste kolory do każdego pojedynczego piksela.

Aby to wykonać należy rozdzielić kanały na R, G, B, a następnie uzupełnić wszelkie brakujące piksele w poszczególnych kanałach. Na samym końcu należy nałożyć kanały na siebie i powstanie całościowy obraz.



* 1. Skalowanie i obracanie

**Skalowanie** obrazu obejmuje zmianę jego wielkości w pikselach biorąc pod uwagę zadaną skalę. Aby pomniejszyć obraz pozbywamy się części pikseli, więc należy zastosować interpolację, która wygładzi i uśredni kolory. Natomiast przy powiększaniu musimy wytworzyć nowe piksele, do czego również potrzebna będzie interpolacja, aby wyznaczyć odpowiednie wartości kolorów na poszczególnych kanałach dla nowo utworzonych pikseli.

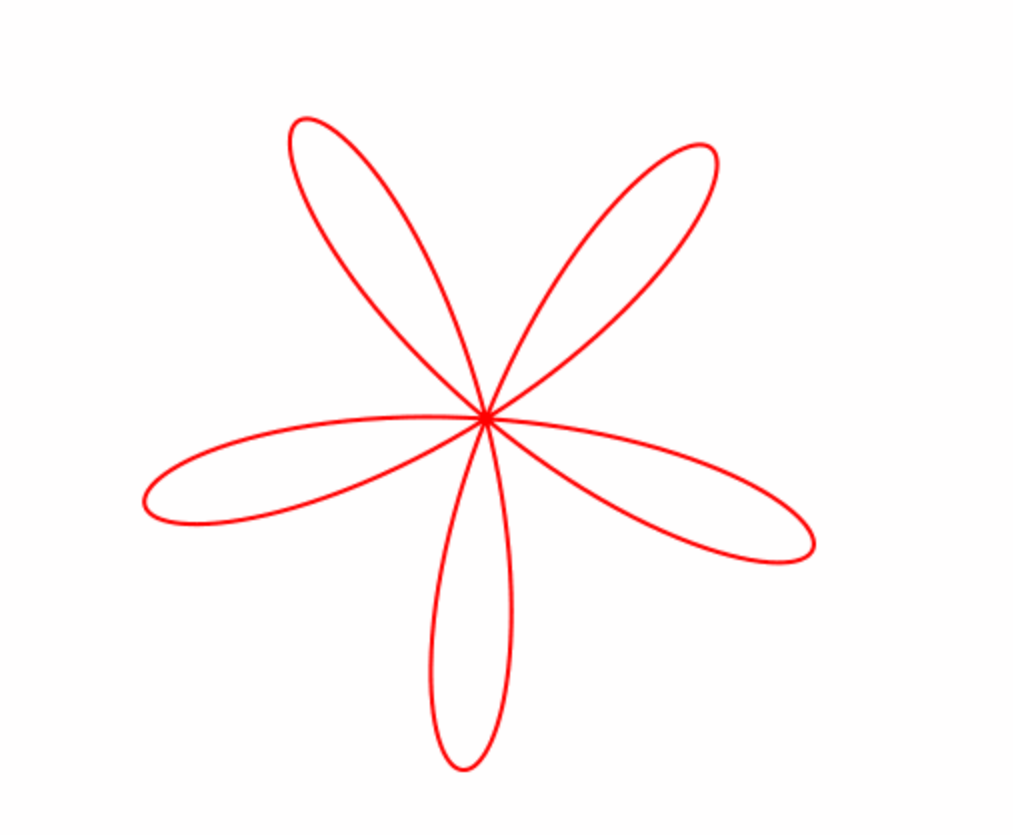
W przypadku **obrotu** obrazu musimy posłużyć się macierzą obrotu wyglądającą w następujący sposób:

dla zadanego kąta α.

Macierz obrotu należy odpowiednio przemnożyć przez każdy z pikseli i otrzymamy wynikowo obrócony obraz.

1. Wyniki algorytmów
   1. Aliasing

Mamy zadane takie, 5-cio łopatowe śmigło:



Zadane wzorem:

Podczas symulowania **aliasingu** sczytując sensorem po kolei linie otrzymamy taki wynik:

Obraz zawierający strzałka

Opis wygenerowany automatycznie

* 1. Demozaikowanie

Mając zadany taki obraz:



Rozpoczynamy mozaikowanie, czyli przetworzenie go za pomocą zadanego filtra, na początek będzie to **Bayer**.

Wynikowym obrazem jest taki obraz:



Co po odpowiednim przybliżeniu daje nam:



Czyli odpowiednio naniesioną maskę. Jak widzimy kolory podzieliły się według danego schematu.

A oto część kodu za to odpowiedzialna:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

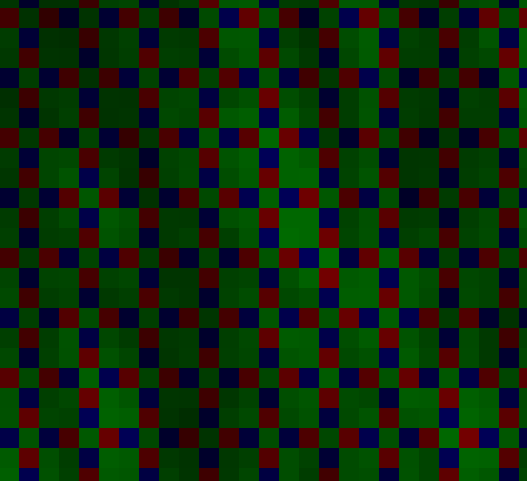
Opis wygenerowany automatycznie

Czyli według zadanej maski następuje wyzerowanie poszczególnych kolorach na pikselach, np. jeśli chcemy uzyskać kolor zielony, musimy wyzerować kolory niebieski oraz czerwony.

Natomiast przy **Fuji X-Trans** obraz wygląda następująco:



A po przybliżeniu:



I za działanie nałożenia filtra **X-Trans** odpowiadają linie kodu:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Następnym zadaniem będzie **interpolacja** obrazu. W moim przypadku użyję algorytmu **najbliższego sąsiada**. Niestety moja implementacja nie jest najbardziej efektywna i efektowna. Aby uzyskać jak najlepsze efekty należałoby skorzystać z gotowych bibliotek tak, aby użyć sprawdzonych i długo wspieranych rozwiązań.

Rezultat dla **Bayera** prezentuje się następująco:

Obraz zawierający roślina

Opis wygenerowany automatycznie

Z daleka obraz wygląda w porządku, ale po przybliżeniu możemy zauważyć, że jest on rozmyty i nie wszystkie piksele pokrywają się kolorystycznie z oryginałem.

Obok zaprezentowany został **difference** pomiędzy oryginałem, a zdemozaikowanym Bayerem.

Natomiast dla **X-Trans** wynik jest następujący:



Jak możemy zauważyć efekt jest całkiem podobny, a **difference** dość **duży**.

Tutaj znajduje się odpowiadający za to kod:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Który działa tak, że pobiera dookoła po kolei piksele w zadanej mu głębokości, a następnie robi z nich średnią i przypisuje kolejno do kanałów pojedynczego piksela.

* 1. Skalowanie i obrażanie

Aby zbadać jak zachowa się obraz podczas skalowania skorzystam już z gotowych funkcji interpolujących, z biblioteki **scipy**.

Mając zadany obraz o rozmiarze **1024px x 1024px**, wyglądający w następujący sposób:



Przeprowadzamy **skalowanie** jego rozmiaru zgodnie ze skalą **0.5**, czyli pomniejszymy obraz dwukrotnie.

Wynikowo otrzymujemy obraz o rozmiarze **512px x 512px**, wyglądający tak:



Następnie spróbujmy powiększyć go **dwukrotnie** tak, aby uzyskać pierwotny obraz. W ten sposób będziemy mogli zauważyć stratę jakości.

Wynik prezentuje się następująco:



Na pierwszy rzut oka może różnice nie są zbyt zauważalne, ale gdy przybliżymy obraz zobaczymy jak zmieniła się jego jakość.

Obraz zawierający ssak, zewnętrzne, zamknąć

Opis wygenerowany automatycznieObraz zawierający pies, rozmyty, patrzenie

Opis wygenerowany automatycznie

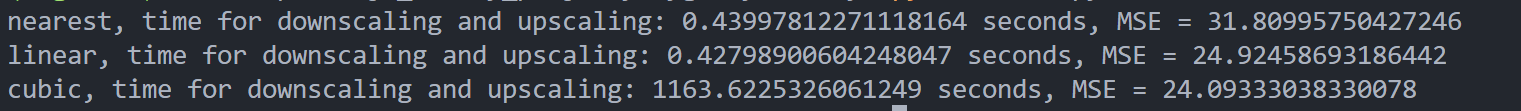
Po lewej znajduje się oryginalny obraz, a po prawej obraz po dwukrotnym skalowaniu. Różnica przez zastosowaną interpolację nie jest aż tak znacząca, ale przy obrazach o mniejszym zagęszczeniu pikseli widoczna będzie utrata jakości.

Tak natomiast wygląda kod odpowiedzialny za skalowanie obrazu:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Jeśli porównamy metody interpolacji: **nearest neighbor, bilinear, bicubic** i sprawdzimy ich **MSE** oraz czas wykonania to otrzymamy następujące wyniki:



Pokazują one, że pierwsze dwa z metod interpolacji są ze sobą zbliżone pod względem czasu wykonania i **MSE** oraz wizualnie również są do siebie podobne. Według teorii najszybszy powinien być **nearest neighbor**, ale najmniej dokładny.

**Bicubic** jak widzimy dość znacznie odbiega od pozostałych dwóch. Jest on najdokładniejszy i bierze najwięcej pikseli do interpolacji.

Wnioskując – każdy sposób interpolacji ma swoje wady i zalety i wszystko jest zależne od naszych potrzeb – jeżeli chcemy otrzymać szybki wynik albo operujemy na bardzo dużych obrazach lepszym wyborem może okazać się **nearest neighbor** bądź **bilinear**, jeżeli natomiast zależy nam na jak najdokładniejszym wyniku to odpowiedzią jest **bicubic**.

Jeżeli chodzi o obracanie obrazu skorzystałem z gotowych, oferowanych przez bibliotekę **OpenCV** metod. Obrócony obraz o 45o wygląda następująco:



Jeżeli chcielibyśmy pozbyć się czarnego pola dookoła musielibyśmy albo zwiększyć skalę obrazu albo zastosować interpolację i zinterpolować czarne piksele dookoła.

Kod odpowiedzialny za rotację obrazu wygląda następująco:

Obraz zawierający tekst

Opis wygenerowany automatycznie