Sprawozdanie z zajęć Inżynieria obrazów
Zajęcia nr 1

Autor sprawozdania: Mateusz Błach

Grupa: Czwartek nieparzysty 11:15

Spis treści

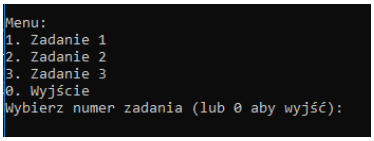
1 Realizacja zadania

1.1 Użyte technologie

W celu realizacji zadania został użyty język programowania Python wraz z bibliotekami numpy oraz opencv-python. Do konwersji projektu w wykonywalny plik exe została użyta biblioteka pyinstaller.

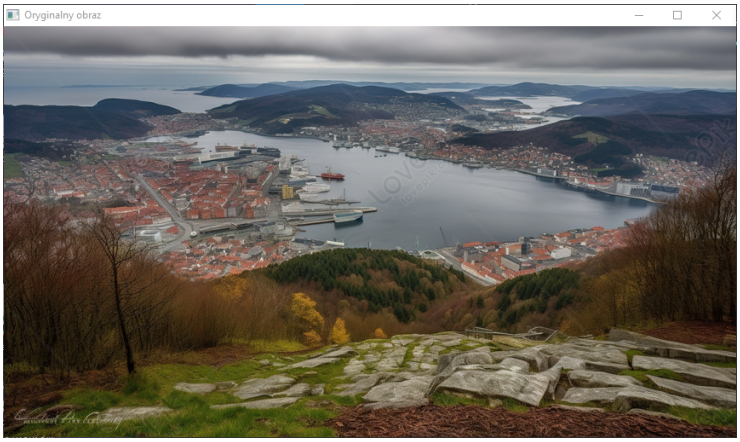
1.2 Obsługa programu

Program jest możliwy do uruchomienia w wresji exe. Po uruchomieniu programu mamy do dyspozycji proste konsolowe menu:



Rysunek 1: Menu Programu

Po uruchomieniu każdego z zadania wyświetlane są okna z obrazem. W każdym z zadań otwiera się oryginalny obraz:



Rysunek 2: Oryginalny obraz

Aby zamknąć otwarte okna z obrazami należy nacisnąć przycisk spacji.

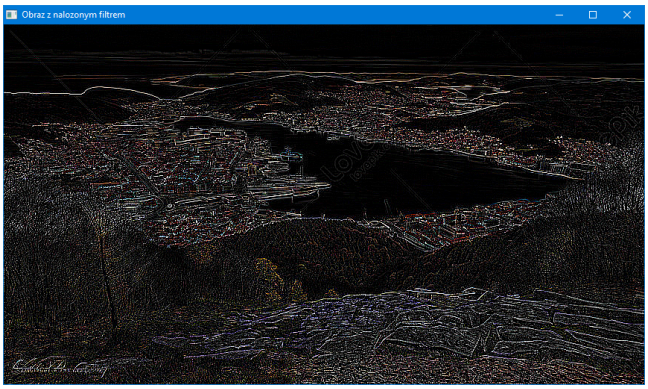
1.3 Zadanie 1

Celem pierwszego zadanie było wykonanie filtru górnoprzepustowego, czyli tak zwanego detektora krawędzi. Na podstawie oryginalnego obrazu filtr wykrywa krawędzie z określoną dokładnością, dokładność filtru określona była przez poniższą maskę:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Rysunek 3: Maska

Otrzymany rezultat prezentuje się tak:



Rysunek 4: Obraz z zastosowanym filtrem górnoprzepustowym

W programie zostało to zrealizowane w następny sposób:

1. Wczytanie obrazu za pomocą funkcji imread z biblioteki opencv-python. Funkcja ta jako argument przyjmuje ścieżke do pliku.
2. Zdefiniowanie maski za pomocą macierzy używając funkcji array z biblioteki numpy.
3. Zastosowanie filtru i zapisanie wyniku do nowej zmiennej. Do nałożenia filtru została użyta funkcja filter2D z bibilioteki opencv-python. Funkcja ta przyjmuje 3 argumenty: obraz oryginalny, wartość głębi bitowej (w przypadku -1 zostanie ona taka sama jak w obrazie wejściowym) i filtr.
4. Wyświetlenie obrazu orignalnego i przekształconego

```python
# Wczytanie obrazu
image = cv2.imread('widok.jpg')

# Maska gornoprzepustowa (detektor krawedzi)
kernel = np.array([[-1, -1, -1],
                   [-1, 8, -1],
                   [-1, -1, -1]])

# Zastosowanie maski na obrazie za pomoca funkcji filtrujacej
filtered_image = cv2.filter2D(image, -1, kernel)

# Wyswietlenie oryginalnego i przefiltrowanego obrazu
cv2.imshow('Oryginalny obraz', image)
cv2.imshow('Obraz z nalozonym filtrem', filtered_image)
```

Listing 1: Kod zadania 1

1.4 Zadanie 2

Celem drugiego zadania było przekształcenie kolorów obrazu. Dla obrazka kolorowego RGB [0-255;0-255;0-255] należało dokonać konwersji na format zmiennoprzecinkowy RGB [0-1.0;0-1.0;0-1.0]. Po dokonanej konwersji należało wyznaczyć nowe wartości kolorów na podstawie wzoru:

$$\begin{bmatrix} R_{new} \\ G_{new} \\ B_{new} \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.689 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Rysunek 5: Wzór wyznaczenia nowych wartości RGB

Otrzymany rezultat prezentuje się tak:

Rysunek 6: Obraz z przekształconymi kolorami

W programie zostało to zrealizowane w następny sposób:

1. Wczytanie obrazu za pomocą funkcji imread z biblioteki opencv-python. Funkcja ta jako argument przyjmuje ścieżke do pliku.
2. Konwersja kolorów na system zmiennoprzecinkowy.
3. Zdefiniowanie macierzy przekształcenia obrazu.
4. Transformacja obrazu.
5. Wyświetlenie obrazu orignalnego i przekształconego

```
# Wczytanie obrazu
image = cv2.imread('widok.jpg')

# Konwersja kolorow z [0-255;0-255;0-255] na
[0-1.0;0-1.0;0-1.0]
image_float = image.astype(np.float32) / 255.0

# Macierz przeksztalcenia kolorww
transformation_matrix = np.array([[0.393, 0.769, 0.189],
                                  [0.349, 0.689, 0.168],
                                  [0.272, 0.534, 0.131]])

# Przeksztalcenie wartosci kolorow zgodnie z podanym wzorem
transformed_image = np.clip(np.dot(image_float,
transformation_matrix), 0, 1.0)

# Wyswietlenie oryginalnego i przeksztalconego obrazu
cv2.imshow('Oryginalny obraz', image)
cv2.imshow('Obraz po transformacji kolorow', (transformed_image
 * 255).astype(np.uint8))
```

Listing 2: Kod zadania 2

1.5 Zadanie 3

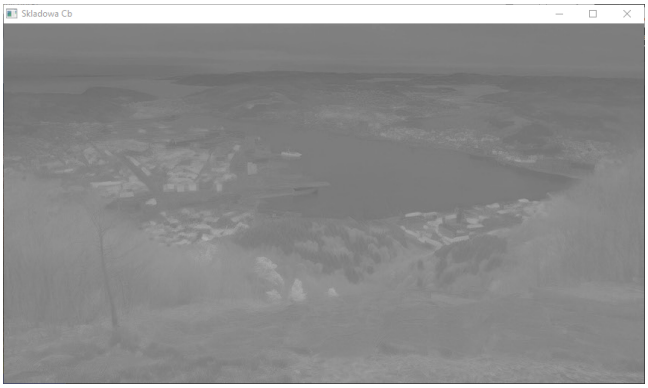
Celem trzeciego zadania było dokonanie konwersji obrazu RGB do modelu barw YCbCr. Każdy piksel musiał zostać wyznaczony na podstawie poniższego wzoru:

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ 0.500 & -0.418 & -0.082 \\ -0.168 & -0.331 & 0.500 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

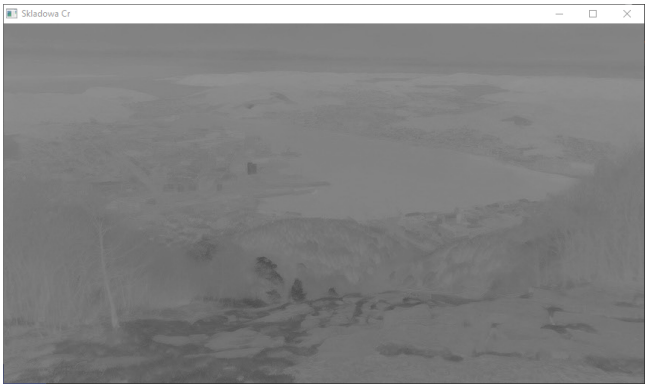
Rysunek 7: Wzór wyznaczenia nowych kolorów YCrCb

YCbCr to model przestrzeni kolorów, używany do cyfrowego przesyłania oraz przechowywania obrazów i wideo. Wykorzystuje do tego trzy typy danych: Y – składową luminancji, Cb – składową różnicową chrominancji Y-B, stanowiącą różnicę między luminancją a niebieskim, oraz Cr – składową chrominancji Y-R, stanowiącą różnicę między luminancją a czerwonym. Kolor zielony jest uzyskiwany na podstawie tych trzech wartości.

Należało wyświetlić każdą składową Y, Cr, Cb w odcieniach szarości oraz obraz po konwersji odwrotnej:

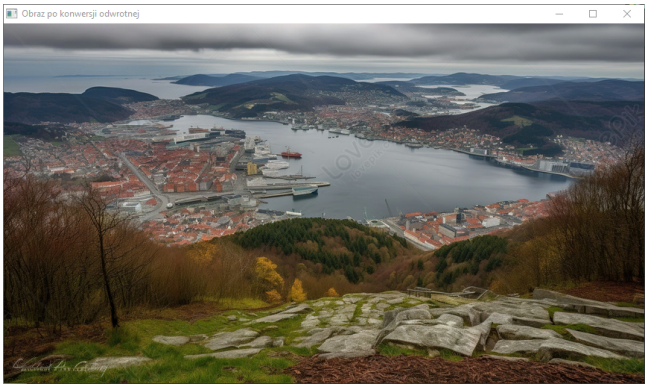


Rysunek 8: Składowa Cb



Rysunek 9: Składowa Cr

Rysunek 10: Składowa Y



Rysunek 11: Obraz po konwersji odwrotnej

W programie zostało to zrealizowane w następny sposób:

1. Wczytanie obrazu za pomocą funkcji imread z biblioteki opencv-python. Funkcja ta jako argument przyjmuje ścieżke do pliku.
2. Zdefiniowanie macierzy przekształcenia obrazu z RBD do YCrCb.
3. Zdefiniowanie macierzy stałych potrzebnych do przekształcenia.
4. Przekształcenie obrazu przy użyciu funkcji dot z biblioteki numpy. Która mnoży zadanie macierze.
5. Zastosowanie ograniczenia nowych wartości do przedziału 0-255. W tym celu użyto funkcji clip z biblioteki numpy, ktora przyjmuje obraz oraz wartości wedle których zostaje nałożony limit.
6. Rozdzielenie składowych Y, Cb, Cr za pomocą funkcji split z biblioteki opencv-python.

7. Wyświetlenie obrazu orignalnego i składowych Y, Cb, Cr
8. Dokonanie konwersji do obrazu RGB.
9. Wyświetlenie obrazu po konwersji.

```
# Wczytanie obrazu
image = cv2.imread('widok.jpg')

# Macierz przeksztalcenia kolorow z RGB do YCrCb
conversion_matrix = np.array([[0.229, 0.587, 0.114],
                              [0.500, -0.418, -0.082],
                              [-0.168, -0.331, 0.500]])

# Stale dodawane do wyniku
constants = np.array([0, 128, 128])

# Przeksztalcenie wartosci kolorow zgodnie z podanym wzorem
ycbcr_image = np.dot(image, conversion_matrix.T) + constants

# Ograniczenie wartosci kolorow do przedzialu [0, 255]
ycbcr_image = np.clip(ycbcr_image, 0, 255).astype(np.uint8)

# Wyodrebnienie skladowych Y, Cb, Cr
Y, Cr, Cb = cv2.split(ycbcr_image)

# Wyswietlenie oryginalnego obrazu i skladowych Y, Cb, Cr w
odcieniach szarosci
cv2.imshow('Oryginalny obraz', image)
cv2.imshow('Skladowa Y', Y)
cv2.imshow('Skladowa Cb', Cb)
cv2.imshow('Skladowa Cr', Cr)

# Konwersja z powrotem do przestrzeni kolorowej RGB
inverse_conversion_matrix = np.linalg.inv(conversion_matrix)
rgb_image = np.dot(ycbcr_image - constants,
inverse_conversion_matrix.T)

# Ograniczenie wartosci kolorow do przedzialu [0, 255]
rgb_image = np.clip(rgb_image, 0, 255).astype(np.uint8)

# Wyswietlenie obrazu po konwersji odwrotnej
cv2.imshow('Obraz po konwersji odwrotnej', rgb_image)
```

Listing 3: Kod zadania 3