

UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO
W WARSZAWIE

WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY
SZKOŁA NAUK ŚCISŁYCH

Katarzyna Mitrus

Michał Słotwiński

Wprowadzenie do Przetwarzania Obrazów

Sprawozdanie z laboratorium

Prowadzący:
prof. Wojciech Mokrzycki

Warszawa, 2018

Spis treści

Spis rysunków	3
Rozdział 1. Wstęp	5
1.1 Specyfikacja wykorzystanego fortformatu obrazu	5
1.2 Instrukcja obsługi programu	5
Rozdział 2. Operacje ujednolicania obrazów	6
Rozdział 3. Operacje sumowania arytmetycznego obrazów szarych	7
3.1 Sumowanie (określonej) stałej z obrazem	7
3.2 Sumowanie dwóch obrazów	9
3.3 Mnożenie obrazu przez zadaną liczbę	11
3.4 Mnożenie obrazu przez inny obraz	13
3.5 Mieszanie obrazów z określonym współczynnikiem	16
3.6 Potęgowanie obrazu (z zadaną potęgą)	17
3.7 Dzielenie obrazu przez (zadaną) liczbę oraz przez inny obraz	17
3.8 Pierwiastkowanie obrazu	18
3.9 Logarytmowanie obrazu	18
Rozdział 4. Operacje sumowania arytmetycznego obrazów barwowych	19
4.1 Sumowanie (określonej) stałej z obrazem	19
4.2 Sumowanie dwóch obrazów	21
4.3 Mnożenie obrazu przez zadaną liczbę	24
4.4 Mnożenie obrazu przez inny obraz	27
4.5 Mieszanie obrazów z określonym współczynnikiem	29
Rozdział 5. Operacje geometryczne na obrazie	33
Rozdział 6. Operacje na histogramie obrazu szarego	34
Rozdział 7. Operacje na histogramie obrazu barwowego	35
Rozdział 8. Operacje morfologiczne na obrazach binarnych	36
Rozdział 9. Operacje morfologiczne na obrazach szarych	37
Rozdział 10. Filtrowanie liniowe i nielinowe	38
Rozdział 11. Podsumowanie	39
Bibliografia	40

Spis rysunków

3.1	(Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji	7
3.2	(Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji	8
3.3	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	9
3.4	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	10
3.5	(Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji	12
3.6	(Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji	12
3.7	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	14
3.8	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	14
3.9	(Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem alfa = 0.5, poniżej obraz wynikowy po normalizacji	16
3.10	(Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem alfa = 0.8, poniżej obraz wynikowy po normalizacji	16
4.1	(Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji	19
4.2	(Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji	20
4.3	(Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	22
4.4	(Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	22
4.5	(Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 50, obraz po normalizacji	25
4.6	(Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 100, obraz po normalizacji	25
4.7	(Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	27
4.8	(Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	28

- 4.9 (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem alfa = 0.5, poniżej obraz wynikowy po normalizacji . 30
- 4.10 (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem alfa = 0.8, poniżej obraz wynikowy po normalizacji . 30

Rozdział 1

Wstęp

Laboratoria oh oh... [1]

1.1 Specyfikacja wykorzystanego formatu obrazu

1.2 Instrukcja obsługi programu

Rozdział 2

Operacje ujednolicania obrazów

1. ujednolicenie obrazów szarych geometryczne (liczba wierszy i kolumn piksli)
2. ujednolicenie obrazów szarych rozdzielczościowe (w rastrze)
3. ujednolicenie obrazów RGB geometryczne (liczba wierszy i kolumn piksli)
4. ujednolicenie obrazów RGB rozdzielczościowe (w rastrze)

Rozdział 3

Operacje sumowania arytmetycznego obrazów szarych

Arytmetyczne operacje między pikslami p i q dwóch obrazów są używane w wielu dzia-łach przetwarzania obrazów. Przeprowadzane się je wykonując działania na pojedynczych pikslach i są uwarunkowane wymaganiami zależnymi od typu operacji. Po operacjach arytmetycznych zwykle niezbędna jest normalizacja. W przedstawionych zadaniach do normalizacji wykorzystano wzór:

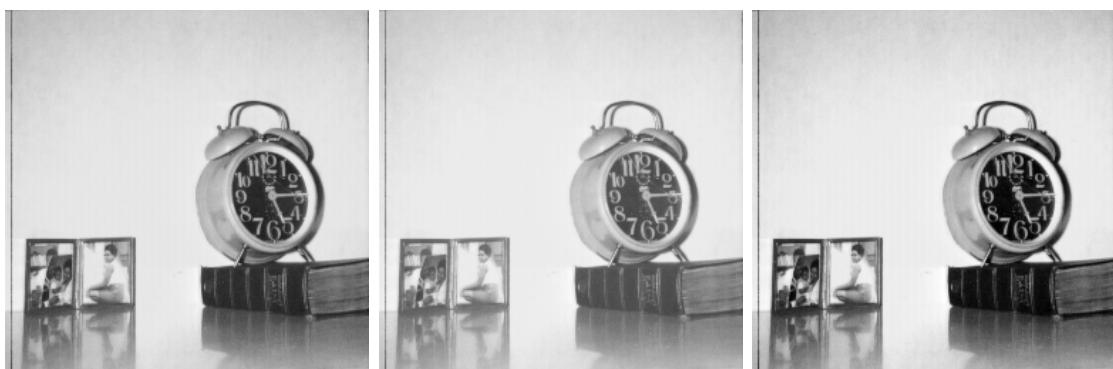
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

3.1 Sumowanie (określonej) stałej z obrazem

Algorytm sumowania obrazu szarego z określona stałą polega na dodaniu do każdej wartości pojedynczego piksla stałej liczby. Po operacji sumowania następuje normalizacja obrazu.

1. Policz sumy wartości kazdego piksla ze stałą (*const*).
2. Jeżeli jedna z tych sum jest większa niż 255 to:
 3. Wybierz największą sumę Q_{max} i policz D_{max} ze wzoru: $D_{max}[i, j] = (Q_{max}[i, j] - 255)$
 4. Oblicz $X = D_{max}/255$
 5. Policz sumy ze wzoru

$$Q[i, j] = P[i, j] - (P[i, j] * X) + const - (const * X)$$



Rysunek 3.1: (Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji



Rysunek 3.2: (Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji

Listing 3.1: Sumowanie obrazu szarego ze stałą

```

image_matrix = self.im1
width = image_matrix.shape[1]      # szereoksc
height = image_matrix.shape[0]     # wysokosc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):
        # Obliczanie sumy
        L = int(image_matrix[x][y]) + int(const)

        # Poszukiwanie maksimum
        if Q_max < L:
            Q_max = L

# Sprawdzenie czy przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255
    X = (D_max/255)
  
```

```

# Obliczenie sumy z uwzględnieniem zakresu
for y in range(height):
    for x in range(width):
        L = (image_matrix[x][y] - (image_matrix[x][y] * X)) + (
            const - (const * X))

        # Zaokrąglenie do najbliższej wartości całkowitej z
        # gory
        # i przypisanie wartości
        result_matrix[x][y] = math.ceil(L)

        # Poszukiwanie minimum i maksimum
        if f_min > L:
            f_min = L
        if f_max < L:
            f_max = L

```

3.2 Sumowanie dwóch obrazów



Rysunek 3.3: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 3.4: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji

Listing 3.2: Sumowanie obrazów szarych

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((height, width), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sumy
        L = int(image1_matrix[x][y]) + int(image2_matrix[x][y])

```

```

# Poszukiwanie maksimum
if Q_max < L:
    Q_max = L

# Sprawdzenie czy przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255
    X = (D_max/255)

# Obliczenie sumy z uwzgllednieniem zakresu
for y in range(height):
    for x in range(width):
        L = (image1_matrix[x][y] - (image1_matrix[x][y] * X)) +
            (image2_matrix[x][y] - (image2_matrix[x][y] * X))

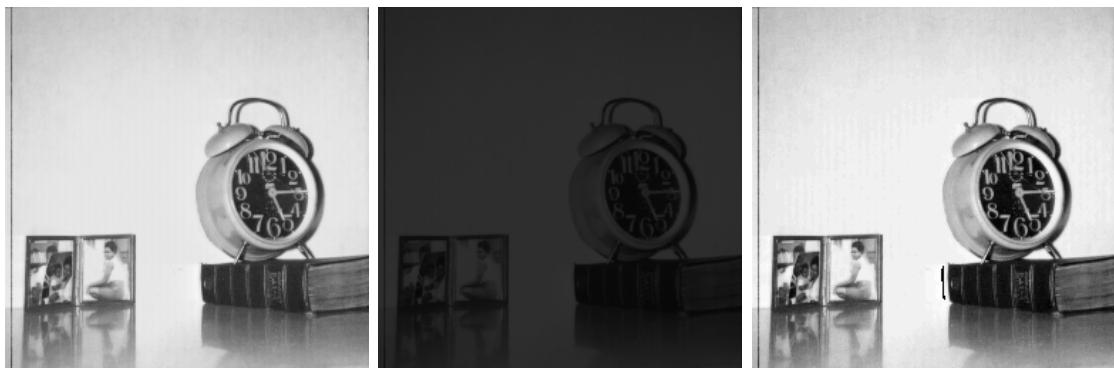
    # Zaokroglenie do najblizszej wartosci calkowitej z
    # gory
    # i przypisanie wartosci
    result_matrix[x][y] = math.ceil(L)

# Poszukiwanie minimum i maksimum
if f_min > L:
    f_min = L
if f_max < L:
    f_max = L

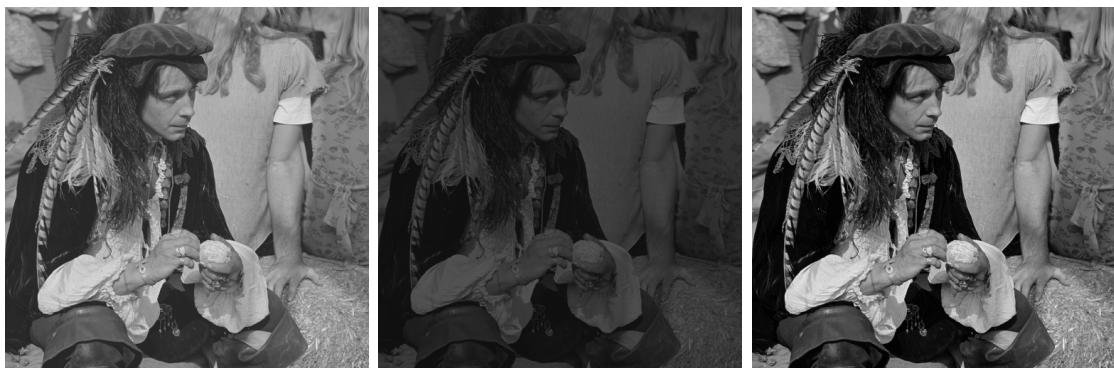
# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min) /
            (f_max - f_min))

```

3.3 Mnożenie obrazu przez zadaną liczbę



Rysunek 3.5: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji



Rysunek 3.6: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji

Listing 3.3: Mnożenie obrazu szarego przez zadaną liczbę

```

image1_matrix = self.im1
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.empty((height, width), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

# Mnożenie
for y in range(height):
    for x in range(width):
        
```

```

L = int(image1_matrix[x][y])
if L == 255:
    L = const
elif L == 0:
    L = 0
else:
    L = (int(image1_matrix[x][y]) * const) / 255

# Zaokroglenie do najblizszej wartosci calkowitej z
# gory
# i przypisanie wartosci
result_matrix[x][y] = math.ceil(L)

# Poszukiwanie minimum i maksimum
if f_min > L:
    f_min = L
if f_max < L:
    f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min)
            / (f_max - f_min))

```

3.4 Mnożenie obrazu przez inny obraz



Rysunek 3.7: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 3.8: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji

Listing 3.4: Mnożenie obrazu szarego przez inny obraz

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((height, width), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        L = int(image1_matrix[x][y])
        if L == 255:
            L = image2_matrix[x][y]
        elif L == 0:
            L = 0
        else:
            L = (int(image1_matrix[x][y]) * int(image2_matrix[x][y])) / 255

# Zaokroglenie do najblzszej wartosci calkowitej z gory
# i przypisanie wartosci
result_matrix[x][y] = math.ceil(L)

# Poszukiwanie minimum i maksimum
if f_min > L:
    f_min = L
if f_max < L:
    f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min) / (f_max - f_min))

```



Rysunek 3.9: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.5$, poniżej obraz wynikowy po normalizacji



Rysunek 3.10: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.8$, poniżej obraz wynikowy po normalizacji

Listing 3.5: Mieszanie obrazów szarych z określonym współczynnikiem

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.empty((height, width), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        L = float(image1_matrix[x][y]) * alfa + (1-alfa) *
            float(image2_matrix[x][y])

        # Zaokroglenie do najblizszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y] = math.ceil(L)

        # Poszukiwanie minimum i maksimum
        if f_min > L:
            f_min = L
        if f_max < L:
            f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min) /
            (f_max - f_min))

```

3.6 Potęgowanie obrazu (z zadana potęgą)

3.7 Dzielenie obrazu przez (zadana) liczbę oraz przez inny obraz

3.8 Pierwiastkowanie obrazu**3.9 Logarytmowanie obrazu**

Rozdział 4

Operacje sumowania arytmetycznego obrazów barwowych

Arytmetyczne operacje na obrazach barwowych przeprowadza się wykonując działania na pojedynczych pikslach i są uwarunkowane wymaganiami zależnymi od typu operacji. W przedstawionych zadaniach poruszamy się w przestrzeni barw RGB, a do normalizacji wykorzystano wzór:

$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

4.1 Sumowanie (określonej) stałej z obrazem

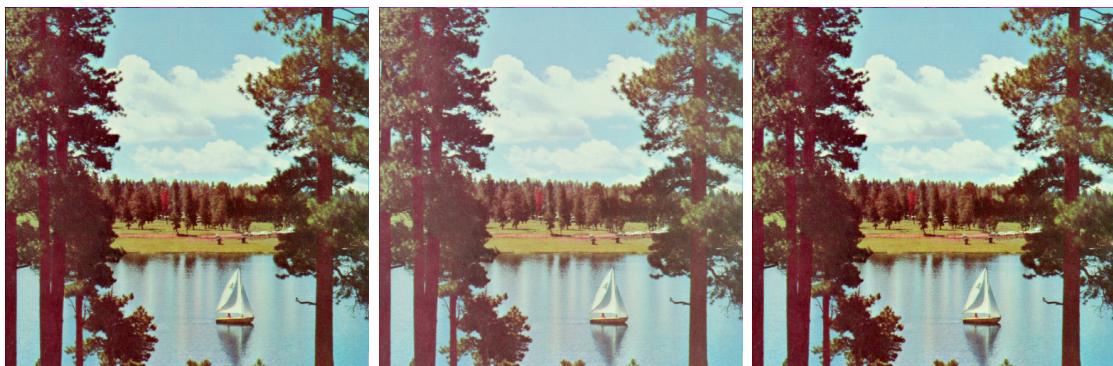
Algorytm sumowania obrazu barwowego z określona stałą polega na dodaniu do każdej składowej barwowej pojedynczego piksla stałej liczby. Po operacji sumowania następuje normalizacja obrazu.

1. Policz sumy wartości kazdego piksla ze stałą (*const*).
2. Jeżeli jedna z tych sum jest większa niż 255 to:
 3. Wybierz największą sumę Q_{max} i policz D_{max} ze wzoru: $D_{max}[i, j] = (Q_{max}[i, j] - 255)$
 4. Oblicz $X = D_{max}/255$
 5. Policz sumy ze wzoru

$$Q_R[i, j] = P_R[i, j] - (P_{1R} * X) + const - (const * X) - 1$$

$$Q_G[i, j] = P_G[i, j] - (P_G * X) + const - (const * X) - 1$$

$$Q_B[i, j] = P_B[i, j] - (P_B * X) + const - (const * X) - 1$$



Rysunek 4.1: (Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji



Rysunek 4.2: (Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji

Listing 4.1: Sumowanie obrazu barwowego ze stałą

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

result_matrix = np.empty((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R = int(image_matrix[x][y][0]) + int(const)
        G = int(image_matrix[x][y][1]) + int(const)
        B = int(image_matrix[x][y][2]) + int(const)

        # Poszukiwanie maksimum
        if Q_max < max([R, G, B]):
            Q_max = max([R, G, B])

# Sprawdzenie czy maksimum przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255

```

```

X = (D_max/255) # Obliczenie proporcji

# Obliczenie sum z uwzglednieniem zakresu
for y in range(height):
    for x in range(width):
        R = (image_matrix[x][y][0] - (image_matrix[x][y][0] * X
            )) + (const - (const * X))
        G = (image_matrix[x][y][1] - (image_matrix[x][y][1] * X
            )) + (const - (const * X))
        B = (image_matrix[x][y][2] - (image_matrix[x][y][2] * X
            )) + (const - (const * X))

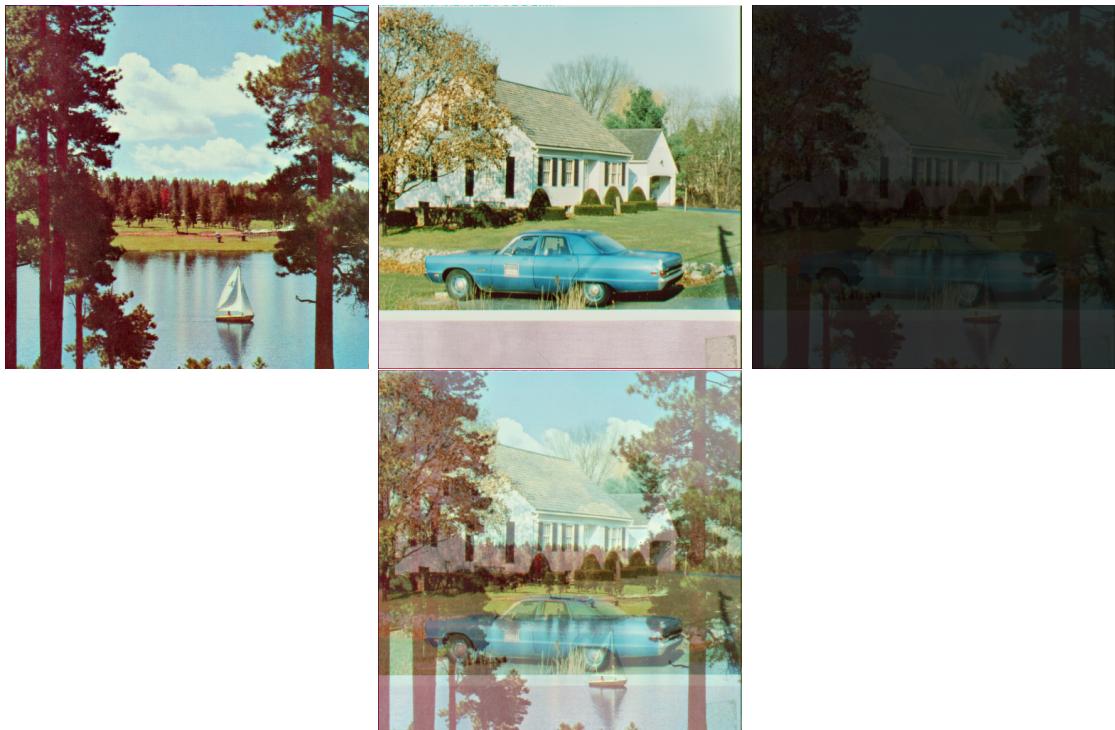
        # Zaokroglenie do najblizszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y][0] = math.ceil(R)
        result_matrix[x][y][1] = math.ceil(G)
        result_matrix[x][y][2] = math.ceil(B)

        # Poszukiwanie minimum i maksimum
        if f_min > min([R, G, B]):
            f_min = min([R, G, B])
        if f_max < max([R, G, B]):
            f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
            f_min) / (f_max - f_min))

```

4.2 Sumowanie dwóch obrazów



Rysunek 4.3: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 4.4: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji

Listing 4.2: Sumowanie obrazów barwowych

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.empty((height, width, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R = int(image1_matrix[x][y][0]) + int(image2_matrix[x][y][0])
        G = int(image1_matrix[x][y][1]) + int(image2_matrix[x][y][1])
        B = int(image1_matrix[x][y][2]) + int(image2_matrix[x][y][2])

        # Poszukiwanie maksimum
        if Q_max < max([R, G, B]):
            Q_max = max([R, G, B])

# Sprawdzenie czy maximum przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255
    X = (D_max/255) # Obliczenie proporcji

# Obliczenie sum z uwzglednieniem zakresu
for y in range(height):
    for x in range(width):
        R = (image1_matrix[x][y][0] - (image1_matrix[x][y][0] * X)) + (image2_matrix[x][y][0] - (image2_matrix[x][y][0] * X))

```

```

G = (image1_matrix[x][y][1] - (image1_matrix[x][y][1] *
    X)) + (image2_matrix[x][y][1] - (image2_matrix[x][
    y][1] * X))
B = (image1_matrix[x][y][2] - (image1_matrix[x][y][2] *
    X)) + (image2_matrix[x][y][2] - (image2_matrix[x][
    y][2] * X))

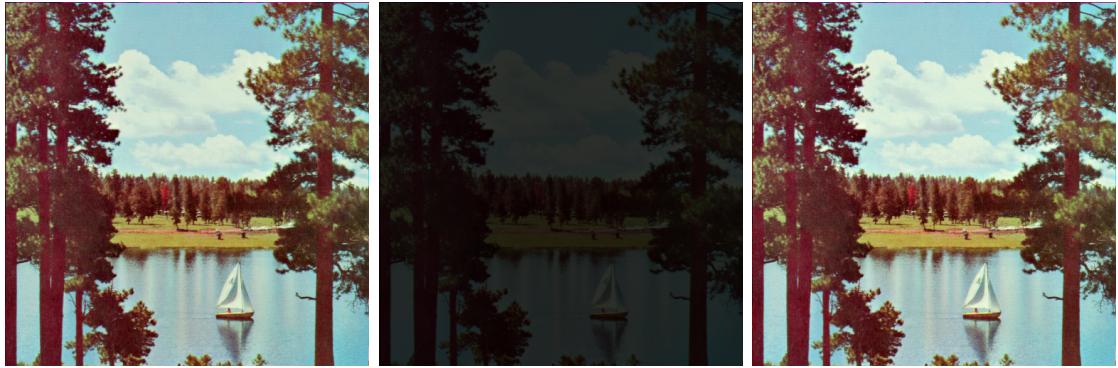
# Zaokroglenie do najblizszej wartosci calkowitej z
# gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
            f_min) / (f_max - f_min))

```

4.3 Mnożenie obrazu przez zadaną liczbę



Rysunek 4.5: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 50, obraz po normalizacji



Rysunek 4.6: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 100, obraz po normalizacji

Listing 4.3: Mnożenie obrazu barwowego przez zadaną liczbę

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

result_matrix = np.empty((height, width, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        R = int(image_matrix[x][y][0])

```

```

G = int(image_matrix[x][y][1])
B = int(image_matrix[x][y][2])

if R == 255:
    R = const
elif R == 0:
    R = 0
else:
    R = (int(image_matrix[x][y][0]) * int(const))/255

if G == 255:
    G = const
elif G == 0:
    G = 0
else:
    G = (int(image_matrix[x][y][1]) * int(const))/255

if B == 255:
    B = const
elif B == 0:
    B = 0
else:
    B = (int(image_matrix[x][y][2]) * int(const))/255

# Zaokroglenie do najblzszej wartosci calkowitej z
# gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

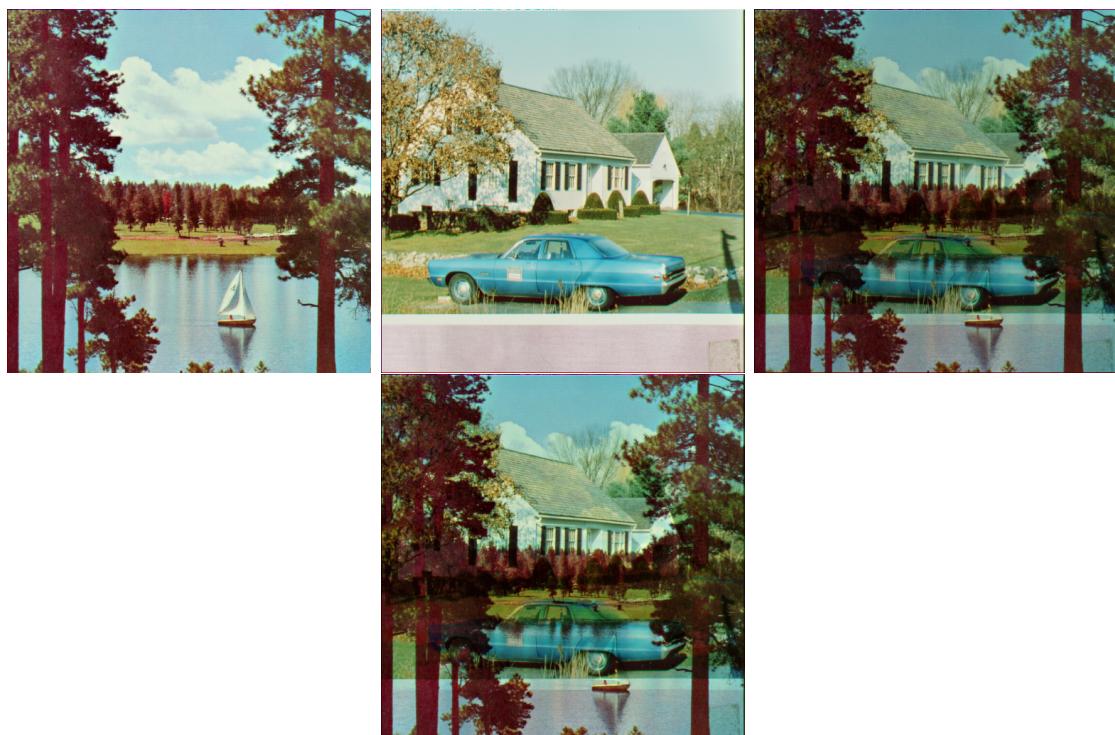
# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))

```

```
norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -  
    f_min) / (f_max - f_min))  
norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -  
    f_min) / (f_max - f_min))
```

4.4 Mnożenie obrazu przez inny obraz



Rysunek 4.7: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstający w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 4.8: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji

Listing 4.4: Mnożenie obrazu barwowego przez inny obraz

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((height, width), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        L = int(image1_matrix[x][y])
        if L == 255:
            L = image2_matrix[x][y]
        elif L == 0:
            L = 0

```

```
else:  
    L = (int(image1_matrix[x][y]) * int(image2_matrix[x]  
        ][y]))/255  
  
    # Zaokrąglenie do najbliższej wartości całkowitej z  
    # gory  
    # i przypisanie wartości  
    result_matrix[x][y] = math.ceil(L)  
  
    # Poszukiwanie minimum i maksimum  
    if f_min > L:  
        f_min = L  
    if f_max < L:  
        f_max = L  
  
# Normalizacja  
norm_matrix = np.zeros((width, height), dtype=np.uint8)  
for y in range(height):  
    for x in range(width):  
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min  
            ) / (f_max - f_min))
```

4.5 Mieszanie obrazów z określonym współczynnikiem



Rysunek 4.9: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.5$, poniżej obraz wynikowy po normalizacji



Rysunek 4.10: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.8$, poniżej obraz wynikowy po normalizacji

Listing 4.5: Mieszanie obrazów barwowych z określonym współczynnikiem

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.empty((height, width, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        R = float(image1_matrix[x][y][0]) * alfa + (1-alfa) *
            float(image2_matrix[x][y][0])
        G = float(image1_matrix[x][y][1]) * alfa + (1-alfa) *
            float(image2_matrix[x][y][1])
        B = float(image1_matrix[x][y][2]) * alfa + (1-alfa) *
            float(image2_matrix[x][y][2])

# Zaokroglenie do najblizeszej wartosci calkowitej z gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))

```

```
norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -  
    f_min) / (f_max - f_min))  
norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -  
    f_min) / (f_max - f_min))
```

4. potęgowanie obrazu (z zadaną potęgą)
5. dzielenie obrazu przez (zadaną) liczbę oraz
przez inny obraz
6. pierwiastkowanie obrazu
7. logarytmowanie obrazu

Rozdział 5

Operacje geometryczne na obrazie

1. przemieszczenie obrazu o zadany wektor
2. jednorodne i niejednorodne skalowanie obrazu
3. obracanie obrazu o dowolny kąt
4. symetrie względem osi układu i zadanej prostej
5. wycinanie fragmentów obrazu
6. kopiowanie fragmentów obrazów

Rozdział 6

Operacje na histogramie obrazu szarego

1. obliczanie histogramu
2. przemieszczanie histogramu
3. rozciąganie histogramu
4. progowanie lokalne
5. progowanie globalne

Rozdział 7

Operacje na histogramie obrazu barwowego

1. obliczanie histogramu
2. przemieszczanie histogramu
3. rozciąganie histogramu
4. progowanie 1-progowe
5. progowanie wieloprogowe
6. progowanie lokalne
7. progowanie globalne

Rozdział 8

Operacje morfologiczne na obrazach binarnych

1. okrawanie(erozja) 2. nakładanie (dylatacja) 3. otwarcie 4. zamknięcie

Rozdział 9

Operacje morfologiczne na obrazach szarych

1. okrawanie(erodzja) 2. nakładanie (dylatacja) 3. otwarcie 4. zamknięcie

Rozdział 10

Filtrowanie liniowe i nieliniowe

1. dolnoprzepustowe (dwa do wyboru)
2. górnoprzepustowe (Roberts, Prewitta, Sobella,)
3. gradientowe (kompasowe, płaskorzeźbowe kierunkowe, gradientu wektorowego VGO, gradientu wektora kierunkowego VDG).
4. medianowe
5. ekstremalne

Rozdział 11

Podsumowanie

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliografia

- [1] Wojciech S. Mokrzycki. *Wprowadzenie do przetwarzania informacji wizualnej Tom II*. Akademicka Oficyna Wydawnicza EXIT, 2012.