

# Sygnały i obrazy cyfrowe

## Sprawozdanie z przetwarzania obrazów

Michał Wróblewski

272488

Data wykonania sprawozdania: 29.11.2023 r.

### 1 Wstęp

W niniejszym sprawozdaniu analizuję różne techniki interpolacji danych, zastosowanie masek Bayera i X-Trans, proces demozaikowania w kontekście przetwarzania obrazów oraz obracanie i skalowanie obrazu.

### 2 Interpolacja

W pierwszej części skupiam się na analizie różnych technik interpolacji danych. Przedstawiam kod implementujący interpolację, wykorzystującą metody takie jak 'linear' i 'nearest'. Opisuję i porównuję wygenerowane wykresy, które obrazują efekty poszczególnych technik interpolacji.

#### 2.1 Kod ogólny

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Funkcja do interpolacji najbliższego sąsiada
5 # ...
6
7 # Funkcja do interpolacji liniowej
8 # ...
9
10 # Dane wejściowe
11 x1 = np.linspace(0, 5, 10) # Przykładowe wartości x
12 y1 = np.sin(x1) # Wartości y dla funkcji sinus
13 x2 = np.linspace(0, 5, 100) # Nowe wartości x dla interpolacji
14
15 # Tworzenie wykresu dla (x1, y1)
16 plt.plot(x1, y1, '-r')
```

```

17 plt.xlabel('x1')
18 plt.ylabel('y1')
19 plt.title('Wykres x1 na y1')
20 plt.show(block=False)
21
22 # Interpolacja najbliższego sąsiada
23 # ...
24
25 # Tworzenie nowego wykresu dla (x2, y2) - Interpolacja najbliższego
26     ← sąsiada
27 # ...
28 # Interpolacja liniowa
29 # ...
30
31 # Tworzenie nowego wykresu dla (x2, y2) - Interpolacja liniowa
32 # ...

```

## 2.2 Funkcja do interpolacji najbliższego sąsiada

```

1 # Funkcja do interpolacji najbliższego sąsiada
2 def nearest_neighbor_interpolation(x, y, x_new):
3     y_new = [] # Tworzenie pustej listy na nowe wartości y
4     for xi in x_new: # Przechodzenie przez nowe wartości x
5         closest_idx = np.argmin(np.abs(x - xi)) # Znalezienie indeksu
9         ← najbliższego sąsiada
6         y_new.append(y[closest_idx]) # Dodanie wartości y dla
9         ← najbliższego sąsiada
7     return np.array(y_new) # Zwrócenie nowych wartości y jako tablicy

```

## 2.3 Funkcja do interpolacji liniowej

```

1 # Funkcja do interpolacji liniowej
2 def linear_interpolation(x, y, x_new):
3     y_new = [] # Tworzenie pustej listy na nowe wartości y
4     for xi in x_new: # Przechodzenie przez nowe wartości x
5         if xi <= x[0]: # Sprawdzenie, czy wartość x_new jest mniejsza
9         ← niż pierwsza wartość w x
6             y_new.append(y[0]) # Jeśli tak, dodanie pierwszej wartości
9             ← y
7         elif xi >= x[-1]: # Sprawdzenie, czy wartość x_new jest
9             ← większa niż ostatnia wartość w x

```

```

8         y_new.append(y[-1]) # Jeśli tak, dodanie ostatniej
9             ← wartości y
10    else:
11        idx = np.where(x > xi)[0][0] # Znalezienie indeksu, gdzie
12            ← wartość x przekracza x_new
13        x_left, x_right = x[idx - 1], x[idx] # Określenie
14            ← sąsiednich wartości x
15        y_left, y_right = y[idx - 1], y[idx] # Określenie
16            ← sąsiednich wartości y
17        slope = (y_right - y_left) / (x_right - x_left) #
18            ← Obliczenie nachylenia prostej
19        y_interp = y_left + slope * (xi - x_left) # Obliczenie
20            ← interpolowanej wartości y
21        y_new.append(y_interp) # Dodanie interpolowanej wartości
22            ← y
23    return np.array(y_new) # Zwrócenie nowych wartości y jako tablicy

```

## 2.4 Interpolacja najbliższego sąsiada - użycie i wykres

```

1 # Interpolacja najbliższego sąsiada
2 y2_nearest = nearest_neighbor_interpolation(x1, y1, x2) # Wywołanie
3             ← funkcji interpolacji najbliższego sąsiada
4
5 # Tworzenie nowego wykresu dla (x2, y2) - Interpolacja najbliższego
6             ← sąsiada
7 plt.figure()
8 plt.plot(x2, y2_nearest, '-b')
9 plt.xlabel('x2')
10 plt.ylabel('y2')
11 plt.title('Interpolacja najbliższego sąsiada - Wykres x2 na y2')
12 plt.show(block=False)

```

## 2.5 Interpolacja liniowa - użycie i wykres

```

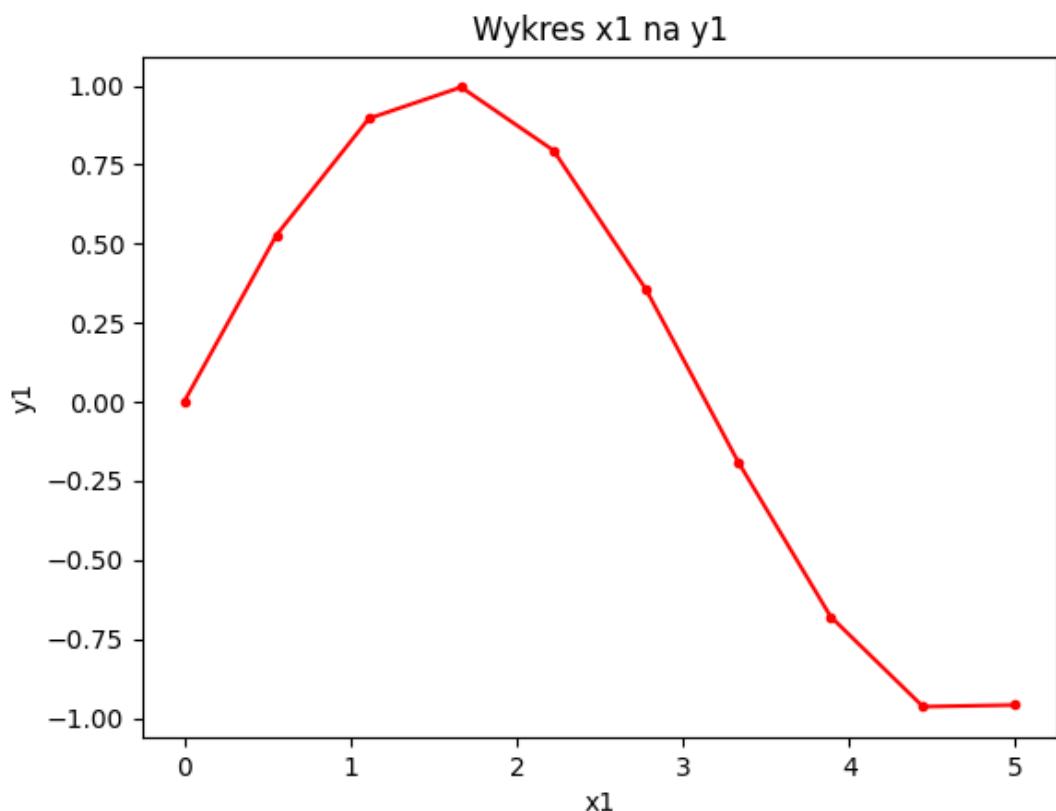
1 # Interpolacja liniowa
2 y2_linear = linear_interpolation(x1, y1, x2) # Wywołanie funkcji
3             ← interpolacji liniowej
4
5 # Tworzenie nowego wykresu dla (x2, y2) - Interpolacja liniowa
6 plt.figure()
7 plt.plot(x2, y2_linear, '-g')
8 plt.xlabel('x2')
9 plt.ylabel('y2')

```

```
9 plt.title('Interpolacja liniowa - Wykres x2 na y2')
10 plt.show()
```

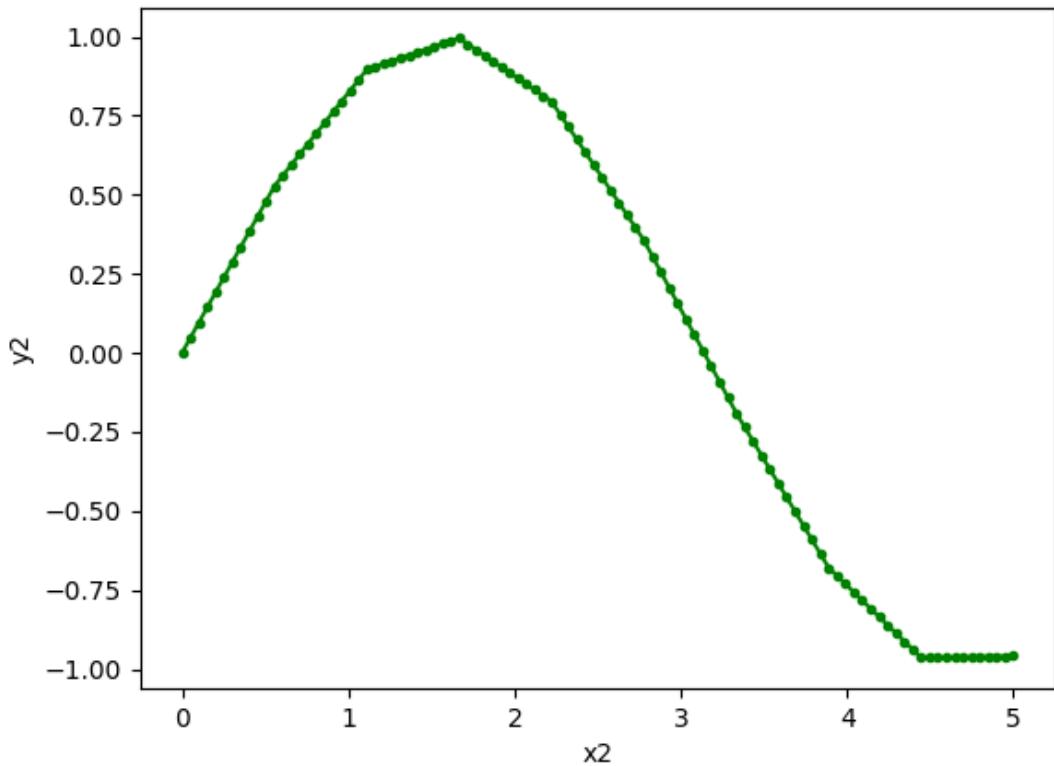
## 2.6 Wykresy interpolacji

Wykresy przedstawiają proces interpolacji danych.



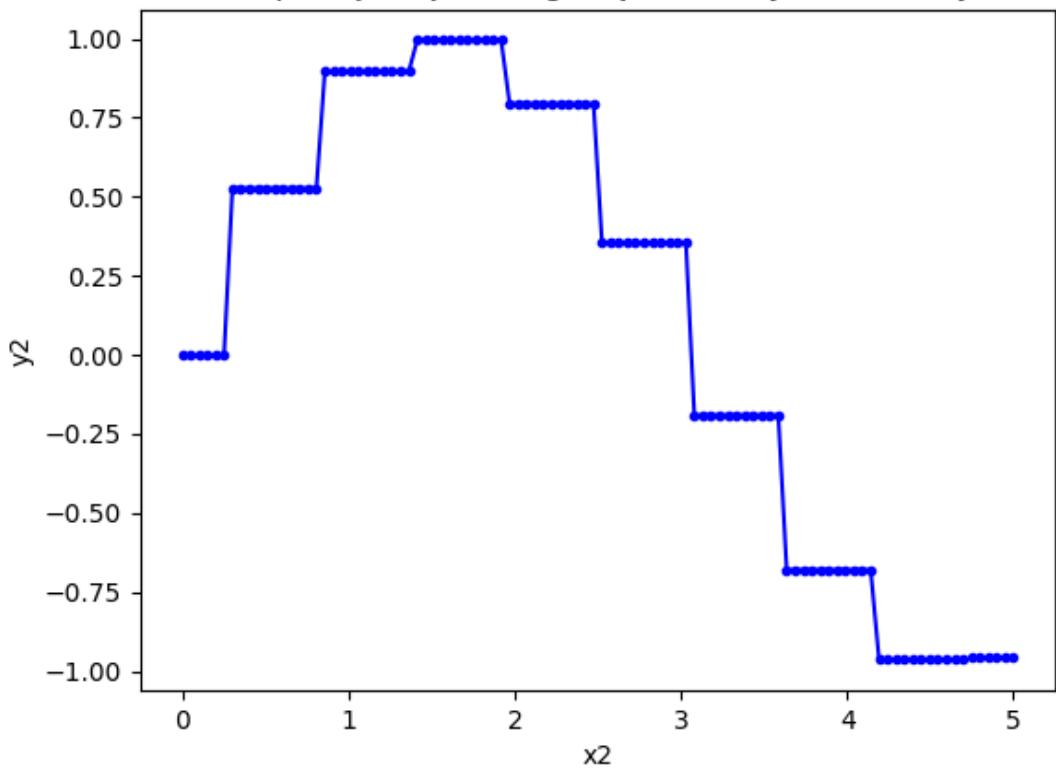
Rysunek 1: Wykres x1 na y1

Interpolacja liniowa - Wykres x2 na y2



Rysunek 2: Wykres x2 na y2 metodą interpolacji liniowej

Interpolacja najbliższego sąsiada - Wykres  $x_2$  na  $y_2$



Rysunek 3: Wykres  $x_2$  na  $y_2$  metodą najbliższego sąsiada

### 3 Przykład zastosowania maski Bayera i X-Trans

W drugiej części mojego sprawozdania przedstawiam kod, który demonstruje stosowanie maski Bayera i X-Trans na obrazie.

#### 3.1 Kod ogólny

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Wczytanie obrazu z pliku 'demo.bmp' za pomocą biblioteki OpenCV
#   (cv2.imread) w domyślnym formacie BGR oraz konwersja przestrzeni
#   na RGB za pomocą funkcji cv2.cvtColor, co pozwala na poprawne
#   interpretowanie kanałów kolorów.
6
7 im = cv2.cvtColor(cv2.imread('demo.bmp'), cv2.COLOR_BGR2RGB)
8
9 # Maska Bayera
10 # ...
11
12 # Maska X-Trans
13 # ...
14
15 # Wyświetlenie oryginalnego i przefiltrowanego obrazu
16 # ...
```

##### 3.1.1 Maska Bayera

```
1 # Zdefiniowanie maski Bayera jako tablicy numpy, która określa, które
#   piksele obrazu należą do poszczególnych kanałów kolorów (czerwony,
#   zielony, niebieski).
2
3 BayerMask = np.array([[ [0, 1], [0, 0] ],
#                         [[1, 0], [0, 1]],
#                         [[0, 0], [1, 0]]], np.uint8)
4
5
6
7 # Utworzenie pustego obrazu o takim samym rozmiarze jak wczytany obraz
#   za pomocą np.zeros_like.
8
9 filtered_image = np.zeros_like(im)
10
11 height, width, _ = im.shape
```

```

12
13 # Iteracja przez każdy piksel obrazu (height, width) oraz przez każdy
14   → kanał koloru (k = 0 dla czerwonego, k = 1 dla zielonego, k = 2 dla
15   → niebieskiego).
16
17 for i in range(height):
18   for j in range(width):
19     for k in range(3): # Przypisanie przefiltrowanej wartości
20       → piksela do filtered_image poprzez pomnożenie wartości
21       → piksela obrazu im przez odpowiadającą mu wartość z maski
22       → Bayera BayerMask.
23     filtered_image[i, j, k] = BayerMask[k, i % 2, j % 2] *
24       → im[i, j, k]

```

### 3.1.2 Maska X-Trans

```

1 XTransMask = np.array([[ [0, 0, 0, 0, 1, 0],
2                           [1, 0, 1, 0, 0, 0],
3                           [0, 0, 0, 0, 1, 0],
4                           [0, 1, 0, 0, 0, 0],
5                           [0, 0, 0, 1, 0, 1],
6                           [0, 1, 0, 0, 0, 0]], #R
7                           [[1, 0, 1, 1, 0, 1],
8                           [0, 1, 0, 0, 1, 0],
9                           [1, 0, 1, 1, 0, 1],
10                          [1, 0, 1, 1, 0, 1],
11                          [0, 1, 0, 0, 1, 0],
12                          [1, 0, 1, 1, 0, 1]], #G
13                           [[0, 1, 0, 0, 0, 0],
14                           [0, 0, 0, 1, 0, 1],
15                           [0, 1, 0, 0, 0, 0],
16                           [0, 0, 0, 0, 1, 0],
17                           [1, 0, 1, 0, 0, 0],
18                           [0, 0, 0, 0, 1, 0]]], np.uint8) #B
19
20 # Użycie maski X-Trans na obrazie
21 filtered_image_xtrans = np.zeros_like(im)
22
23 height, width = im.shape[:2]
24
25 for h in range(height):
26   for w in range(width):
27     for c in range(3): # Dla każdego kanału koloru

```

```
28         filtered_image_xtrans[h, w, c] = XTransMask[c, h % 6, w %
29             % 6] * im[h, w, c]
```

### 3.1.3 Reszta kodu - wyświetlenie

```
1 # Wyświetlenie oryginalnego i przefiltrowanego obrazu
2 plt.figure(figsize=(15, 6))
3
4 plt.subplot(1, 3, 1)
5 plt.title('Oryginalny obraz')
6 plt.imshow(im)
7 plt.axis('off')
8
9 plt.subplot(1, 3, 2)
10 plt.title('Przefiltrowany obraz z maską Bayera')
11 plt.imshow(filtered_image)
12 plt.axis('off')
13
14 plt.subplot(1, 3, 3)
15 plt.title('Przefiltrowany obraz z maską X-Trans')
16 plt.imshow(filtered_image_xtrans)
17 plt.axis('off')
18
19 plt.tight_layout()
20 plt.show()
```

## 3.2 Wyjaśnienie kodu na podstawie maski Bayera

W powyższym kodzie użyto bibliotek: cv2 do wczytania obrazu, numpy do operacji na tablicach oraz matplotlib.pyplot do wyświetlenia obrazów. Następnie:

- Wczytano obraz i przekonwertowano go na przestrzeń kolorów RGB.
- Zdefiniowano maskę Bayera określającą przypisanie pikseli do kanałów kolorów.
- Zastosowano maskę Bayera na obrazie poprzez iterację po pikselach i kanałach koloru.
- Wyświetlono oba obrazy - oryginalny i przefiltrowany.

## 4 Demozaikowanie

Proces demozaikowania

### 4.1 Kod ogólny

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 im = cv2.cvtColor(cv2.imread('demo.bmp'), cv2.COLOR_BGR2RGB)
6
7 # Kod z maski Bayera
8 # ...
9
10 # Kod z maski X-Trans
11 # ...
12
13 # Demozaikowanie Bayera
14 # ...
15
16 # Demozaikowanie X-Trans
17 # ...
18
19 # Wyświetlanie
20 # ...
21
```

#### 4.1.1 Funkcja demozaikowania obrazu na podstawie maski Bayera

```
1 # Funkcja demozaikowania obrazu na podstawie maski Bayera
2 def closest_neighBayer(IMG, BAYERMASK):
3     height, width, _ = IMG.shape
4     demosaiced_image = np.zeros_like(IMG, dtype=np.uint8)    #
5         → Inicjalizacja pustego obrazu docelowego
6
7     for w in range(height):
8         for s in range(width):
9             for k in range(3):    # Iteracja przez kanały koloru
10                 → (czarny, zielony, niebieski)
11                 if BAYERMASK[k, w % 2, s % 2] == 1:    # Jeśli maska
12                     → Bayera określa obecność koloru w danym pikselu
13                     demosaiced_image[w, s, k] = IMG[w, s, k]    # Ustaw
14                     → wartość piksela na wartość oryginalną
```

```

11     else:
12         closest_w = w + 1 if w % 2 == 0 else w - 1 # 
13         ↵ Znajdź najbliższy piksel w poziomie
14         closest_s = s + 1 if s % 2 == 0 else s - 1 # 
15         ↵ Znajdź najbliższy piksel w pionie
16         if 0 <= closest_w < height and 0 <= closest_s <
17             ↵ width:
18             demosaiced_image[w, s, k] = IMG[closest_w,
19             ↵ closest_s, k] # Ustaw wartość na piksel
20             ↵ najbliższego
21     else:
22         # Jeśli piksel jest poza zakresem, ustaw kolor
23         ↵ na biały
24     demosaiced_image[w, s, 0] = demosaiced_image[w,
25     ↵ s, 1] = demosaiced_image[w, s, 2] = 255
26
27 return demosaiced_image # Zwróć obraz docelowy po demozaikowaniu

```

#### 4.1.2 Funkcja demozaikowania obrazu na podstawie maski X-Trans

```

1 # Funkcja demozaikowania obrazu na podstawie maski X-Trans
2 def closest_neighXTrans(XIMG, XTRANSMASK):
3     height, width, _ = XIMG.shape
4     demosaiced_image = np.zeros_like(XIMG, dtype=np.uint8) #
5     ↵ Inicjalizacja pustego obrazu docelowego
6
7     for w in range(height):
8         for s in range(width):
9             for k in range(3): # Iteracja przez kanały koloru
10                 ↵ (czarny, zielony, niebieski)
11                 if XTRANSMASK[k, w % 6, s % 6] == 1: # Jeśli maska
12                     ↵ X-Trans określa obecność koloru w danym pikselu
13                     demosaiced_image[w, s, k] = XIMG[w, s, k] # Ustaw
14                     ↵ wartość piksela na wartość oryginalną
15     else:
16         closest_w = w + 1 if w % 2 == 0 else w - 1 #
17         ↵ Znajdź najbliższy piksel w poziomie
18         closest_s = s + 1 if s % 2 == 0 else s - 1 #
19         ↵ Znajdź najbliższy piksel w pionie
20         if 0 <= closest_w < height and 0 <= closest_s <
21             ↵ width:

```

```

15         demosaiced_image[w, s, k] = XIMG[closest_w,
16             ↵ closest_s, k] # Ustaw wartość na piksel
17             ↵ najblższy
18     else:
19         # Jeśli piksel jest poza zakresem, ustaw kolor
20             ↵ na biały
21         demosaiced_image[w, s, 0] = demosaiced_image[w,
22             ↵ s, 1] = demosaiced_image[w, s, 2] = 255
23
24     return demosaiced_image # Zwróć obraz docelowy po demosaikowaniu

```

#### 4.1.3 Reszta kodu - użycie i wyświetlenie

```

1 # Demosaikowanie obrazu Bayera
2 demosaiced_bayer = closest_neighBayer(im, BayerMask)
3
4 # Demosaikowanie X-Trans
5 demosaiced_xtrans = closest_neighXTrans(im, XTransMask)
6
7 # Wyświetlenie oryginalnego i przefiltrowanego obrazu
8 plt.figure(figsize=(15, 6))
9
10 plt.subplot(1, 4, 1)
11 plt.title('Oryginalny obraz')
12 plt.imshow(im)
13 plt.axis('off')
14
15 plt.subplot(1, 4, 2)
16 plt.title('Przefiltrowany obraz z maską Bayera')
17 plt.imshow(filtered_image_bayer)
18 plt.axis('off')
19
20 plt.subplot(1, 4, 3)
21 plt.title('Przefiltrowany obraz z maską X-Trans')
22 plt.imshow(filtered_image_xtrans)
23 plt.axis('off')
24
25 plt.subplot(1, 4, 4)
26 plt.title('Demosaikowanie obrazu Bayera')
27 plt.imshow(demosaiced_bayer)
28 plt.axis('off')
29
30 plt.tight_layout()

```

```
31 plt.show()
```

## **4.2 Demonstracja**

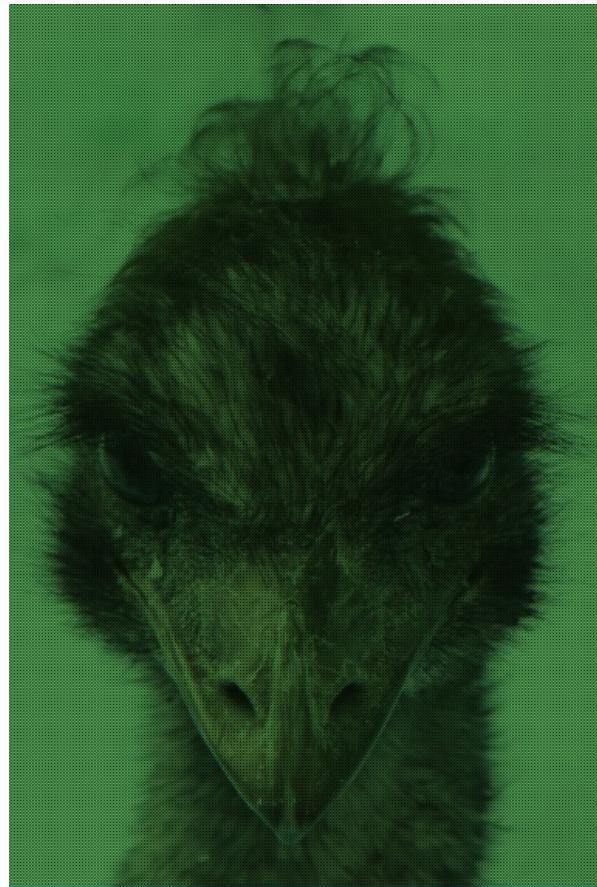
Obrazy po procesie demozaikowania

### **4.2.1 Obraz oryginalny**



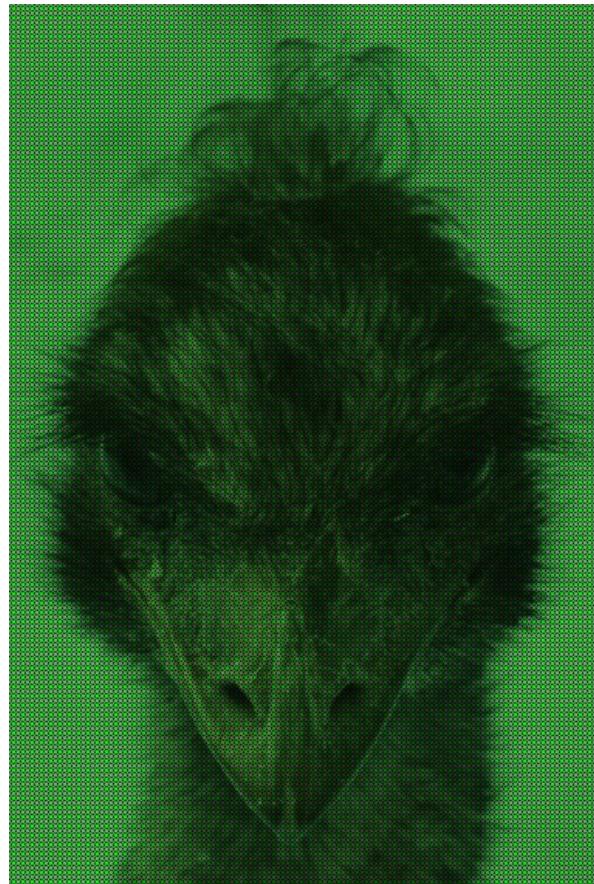
Rysunek 4: Obraz oryginalny

#### 4.2.2 Obraz przefiltrowany z maską Bayera



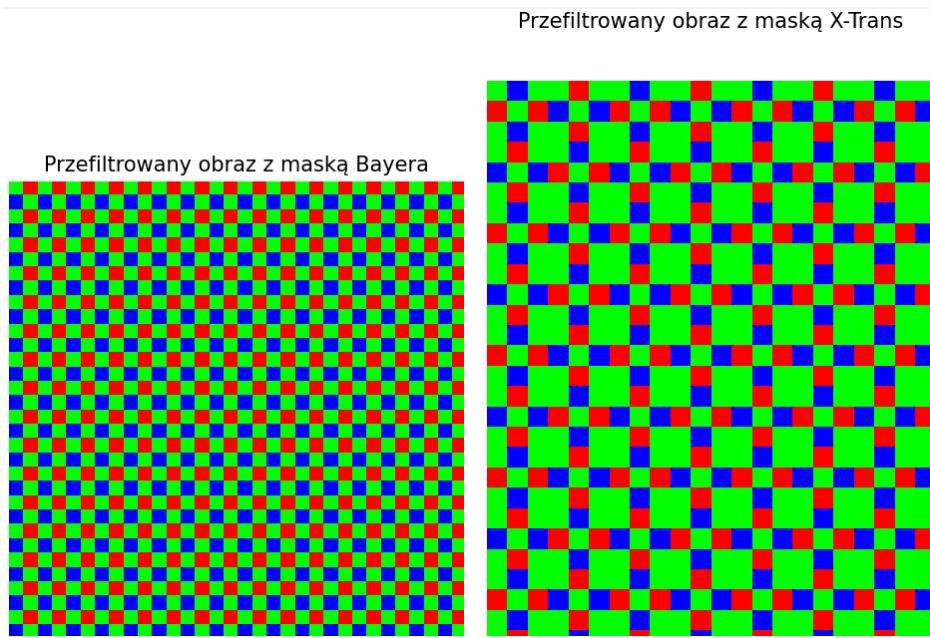
Rysunek 5: Obraz przefiltrowany z maską Bayera

#### 4.2.3 Obraz przefiltrowany z maską X-Trans



Rysunek 6: Obraz przefiltrowany z maską X-Trans

#### 4.2.4 Porównanie masek



Rysunek 7: Porównanie masek

#### 4.2.5 Demozaikowany obraz Bayera



Rysunek 8: Demozaikowany obraz Bayera

#### 4.2.6 Demozaikowany obraz X-Trans



Rysunek 9: Demozaikowany obraz X-Trans

## 5 Obracanie i Skalowanie Obrazu

W trzeciej części zajmuję się obracaniem i skalowaniem obrazu. Przedstawiam kod, który demonstruje funkcje zmniejszania, powiększania oraz obracania obrazu, ilustrując ich zastosowanie.

### 5.1 Kod ogólny

```
1 # Importowanie bibliotek
2 import numpy as np
3 import math
4 import cv2
5 import matplotlib.pyplot as plt
6
7 # Wczytanie obrazu
8 img = cv2.cvtColor(cv2.imread('demo.jpg'), cv2.COLOR_BGR2RGB)
9 original_image = img.copy() # Utworzenie kopii oryginalnego obrazu
10
11 # Funkcja do zmniejszania obrazu o połowę
12 def resize_down(image):
13     # Kod funkcji resize_down
14     # ...
15
16 # Funkcja do powiększania obrazu o dwukrotność
17 def resize_up(image):
18     # Kod funkcji resize_up
19     # ...
20
21 # Funkcja do obracania obrazu o zadany kąt (w stopniach)
22 def rotate_image(Im, angle):
23     # Kod funkcji rotate_image
24     # ...
25
26 # Funkcja do interpolacji przywracającej obraz do oryginalnego
27 # rozmiaru
27 def interpolate_up(image, original_shape):
28     # Kod funkcji interpolate_up
29     # ...
30
31 # Użycie funkcji do zmniejszenia i powiększenia obrazu
32 # ...
33
34 # Przywrócenie obrazu do oryginalnego rozmiaru po interpolacji
```

```

35  # ...
36
37  # Obrót obrazu o 30 stopni przy użyciu macierzy odwrotnej
38  # ...
39
40  # Wyświetlenie obrazów w odpowiednich miejscach
41  # ...

```

### 5.1.1 Funkcja do zmniejszania obrazu o połowę

```

1  # Funkcja do zmniejszania obrazu o połowę
2  def resize_down(image):
3      height, width = image.shape[:2]
4      return image[:, :, ::2]  # Zmniejszenie obrazu o połowę w każdym
   ↵ wymiarze

```

### 5.1.2 Funkcja do powiększania obrazu o dwukrotność

```

1  # Funkcja do powiększania obrazu o dwukrotność
2  def resize_up(image):
3      height, width = image.shape[:2]
4      new_height = height * 3
5      new_width = width * 3
6      result = np.zeros((new_height, new_width, 3), dtype=np.uint8)  #
   ↵ Nowa macierz na powiększony obraz
7      # Iteracja po nowym obrazie i przypisanie wartości pikseli na
   ↵ podstawie obrazu zmniejszonego
8      for i in range(new_height):
9          for j in range(new_width):
10             result[i, j] = image[i // 3, j // 3]  # Przypisanie piksela
   ↵ z obrazu zmniejszonego
11
12  return result

```

### 5.1.3 Funkcja do obracania obrazu o zadany kąt (w stopniach)

```

1  # Funkcja do obrotu obrazu z zastosowaniem interpolacji
2  def rotate_image_with_interpolation(image, angle_degrees):
3      angle_radians = math.radians(angle_degrees)
4      cosine_angle = math.cos(angle_radians)
5      sine_angle = math.sin(angle_radians)
6      height, width = image.shape[:2]
7      channels = image.shape[2] if len(image.shape) > 2 else 1  # Liczba
   ↵ kanałów
8

```

```

9      new_height = round(abs(image.shape[0] * cosine_angle) +
10     ↵ abs(image.shape[1] * sine_angle)) + 1
11
12     new_width = round(abs(image.shape[1] * cosine_angle) +
13     ↵ abs(image.shape[0] * sine_angle)) + 1
14
15     rotated_image = np.zeros((new_height, new_width, channels),
16     ↵ dtype=np.uint8)
17
18
19     original_center_height = (image.shape[0] - 1) / 2
20     original_center_width = (image.shape[1] - 1) / 2
21     new_center_height = (new_height - 1) / 2
22     new_center_width = (new_width - 1) / 2
23
24
25     for w in range(new_height):
26         for k in range(new_width):
27             # Obliczenie współrzędnych piksela obrazu po rotacji
28             x_rotated = (k - new_center_width) * cosine_angle + (w -
29             ↵ new_center_height) * sine_angle
30             y_rotated = (w - new_center_height) * cosine_angle - (k -
31             ↵ new_center_width) * sine_angle
32
33             # Dostosowanie do współrzędnych obrazu przed rotacją
34             x_rotated += original_center_width
35             y_rotated += original_center_height
36
37
38             # Sprawdzenie, czy piksel mieści się w oryginalnych
39             ↵ wymiarach obrazu
40             if 0 <= x_rotated < width - 1 and 0 <= y_rotated < height -
41             ↵ 1:
42                 x1, y1 = int(x_rotated), int(y_rotated)
43                 x2, y2 = x1 + 1, y1 + 1
44
45
46             # Obliczenie wag dla interpolacji dwuliniowej
47             alpha = x_rotated - x1
48             beta = y_rotated - y1
49
50
51             # Interpolacja dwuliniowa dla każdego kanału
52             for c in range(channels):
53                 rotated_image[w, k, c] = (1 - alpha) * (1 - beta) *
54                 ↵ image[y1, x1, c] + alpha * (1 - beta) *
55                 ↵ image[y1, x2, c] \

```

```

41           + (1 - alpha) * beta * image[y2,
42           ↵ x1, c] + alpha * beta *
43           ↵ image[y2, x2, c]
44
45     # Interpolacja obrazu po rotacji, aby przywrócić go do
46     ↵ oryginalnych wymiarów
47     restored_resized_image = interpolate_up(rotated_image, image.shape)
48
49     return restored_resized_image

```

#### 5.1.4 Funkcja do interpolacji przywracającej obraz do oryginalnego

```

1  # Funkcja do interpolacji przywracającej obraz do oryginalnego
2  ↵ rozmiaru
3  def interpolate_up(image, original_shape):
4      new_height, new_width = image.shape[:2]
5      original_height, original_width = original_shape[:2]
6
7      result = np.zeros((original_height, original_width, 3),
8          ↵ dtype=np.uint8)
9
10     # Obliczenie współczynników do interpolacji
11     x_ratio = original_width / new_width
12     y_ratio = original_height / new_height
13
14     for i in range(original_height):
15         for j in range(original_width):
16             # Wyznaczenie odpowiadającego piksela z obrazu
17             ↵ zmniejszonego
18             x = int(j / x_ratio)
19             y = int(i / y_ratio)
20
21             # Przypisanie wartości piksela z obrazu zmniejszonego do
22             ↵ odpowiadającego mu piksela w obrazie powiększonym
23             result[i, j] = image[y, x]
24
25     return result

```

#### 5.1.5 Reszta kodu - użycie i wyświetlenie

```

1  # Użycie funkcji do zmniejszenia i powiększenia obrazu
2  smaller_image = resize_down(img.copy())
3  larger_image = resize_up(smaller_image.copy())
4

```

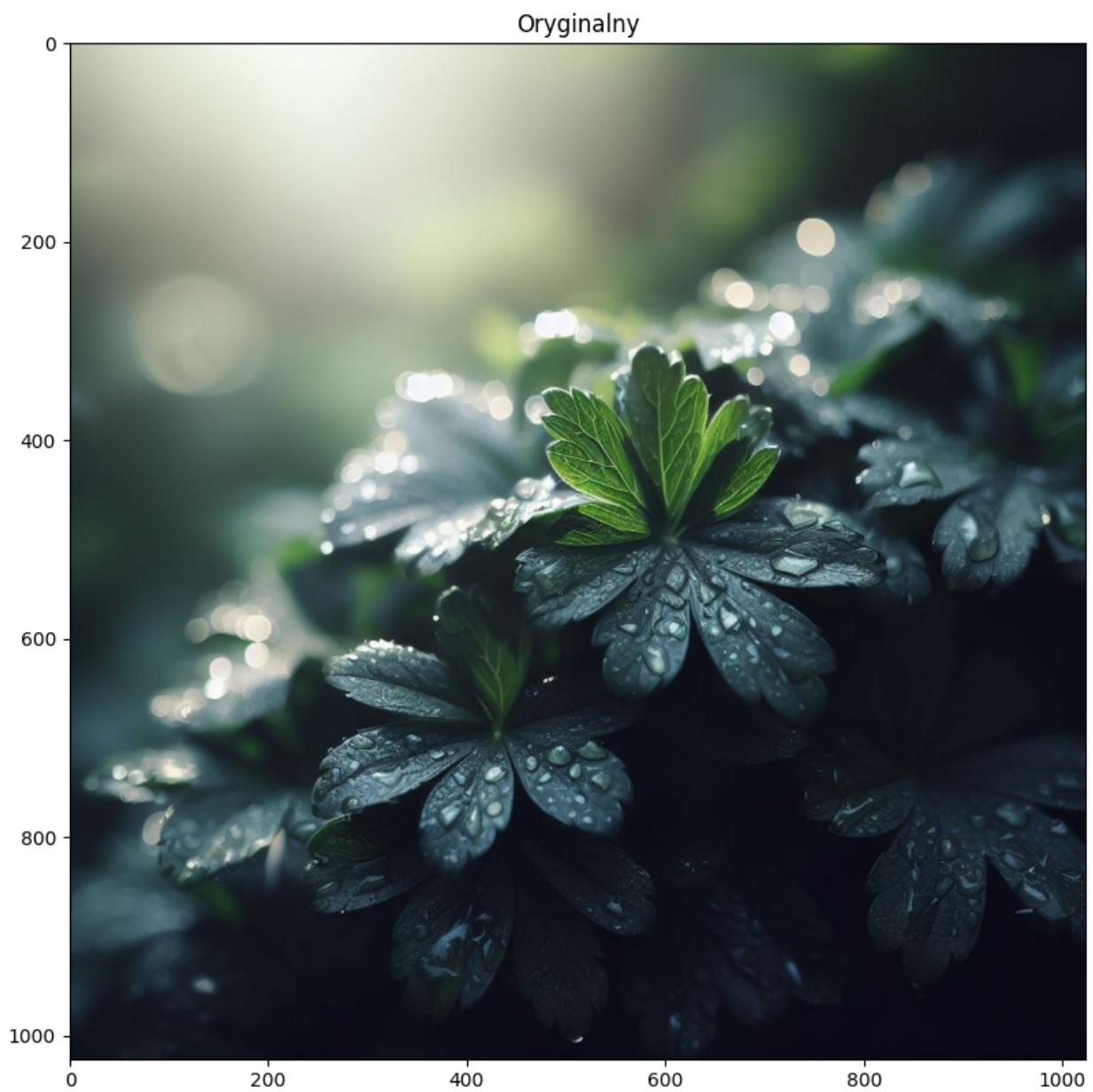
```

5  # Przywrócenie obrazu do oryginalnego rozmiaru po interpolacji
6  resized_image = interpolate_up(larger_image, img.shape)
7
8  # Obrót obrazu o 30 stopni przy użyciu macierzy odwrotnej
9  rotated_image = rotate_image_with_interpolation(original_image, 30)
10
11 # Oryginalny obraz
12 plt.imshow(original_image)
13 plt.title('Oryginalny')
14 plt.show()
15
16 # Pomniejszony obraz
17 plt.imshow(smaller_image)
18 plt.title('Pomniejszony')
19 plt.show()
20
21 # Powiększony obraz
22 plt.imshow(larger_image)
23 plt.title('Powiększony')
24 plt.show()
25
26 # Obraz przeskalowany
27 plt.imshow(restored_resized_image)
28 plt.title('Przeskalowany')
29 plt.show()
30
31 # Obrócony o 30 stopni obraz
32 plt.imshow(rotated_image)
33 plt.title('Obrócony o 30 stopni')
34 plt.show()

```

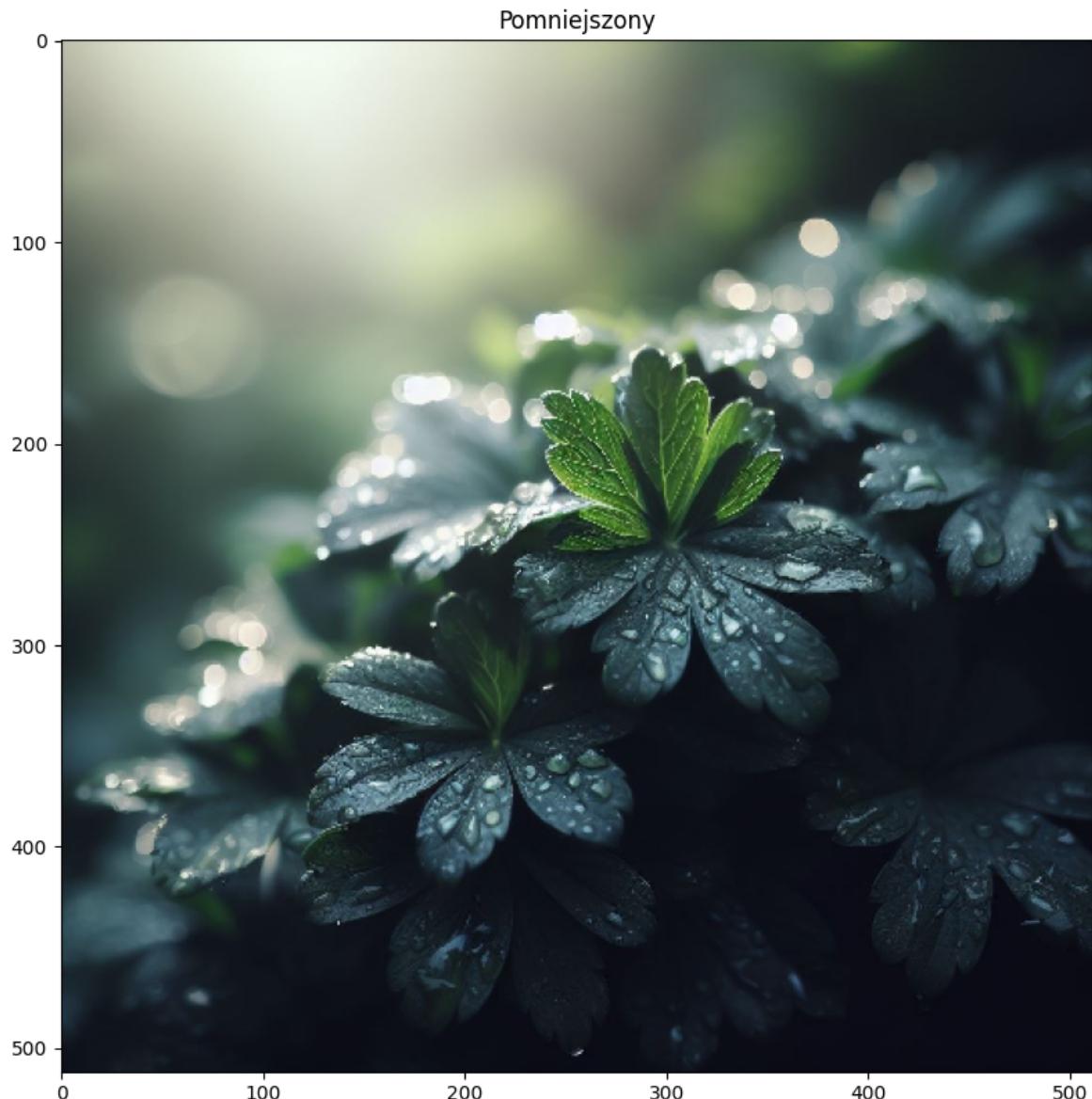
## 5.2 Demonstracja

### 5.2.1 Oryginalny



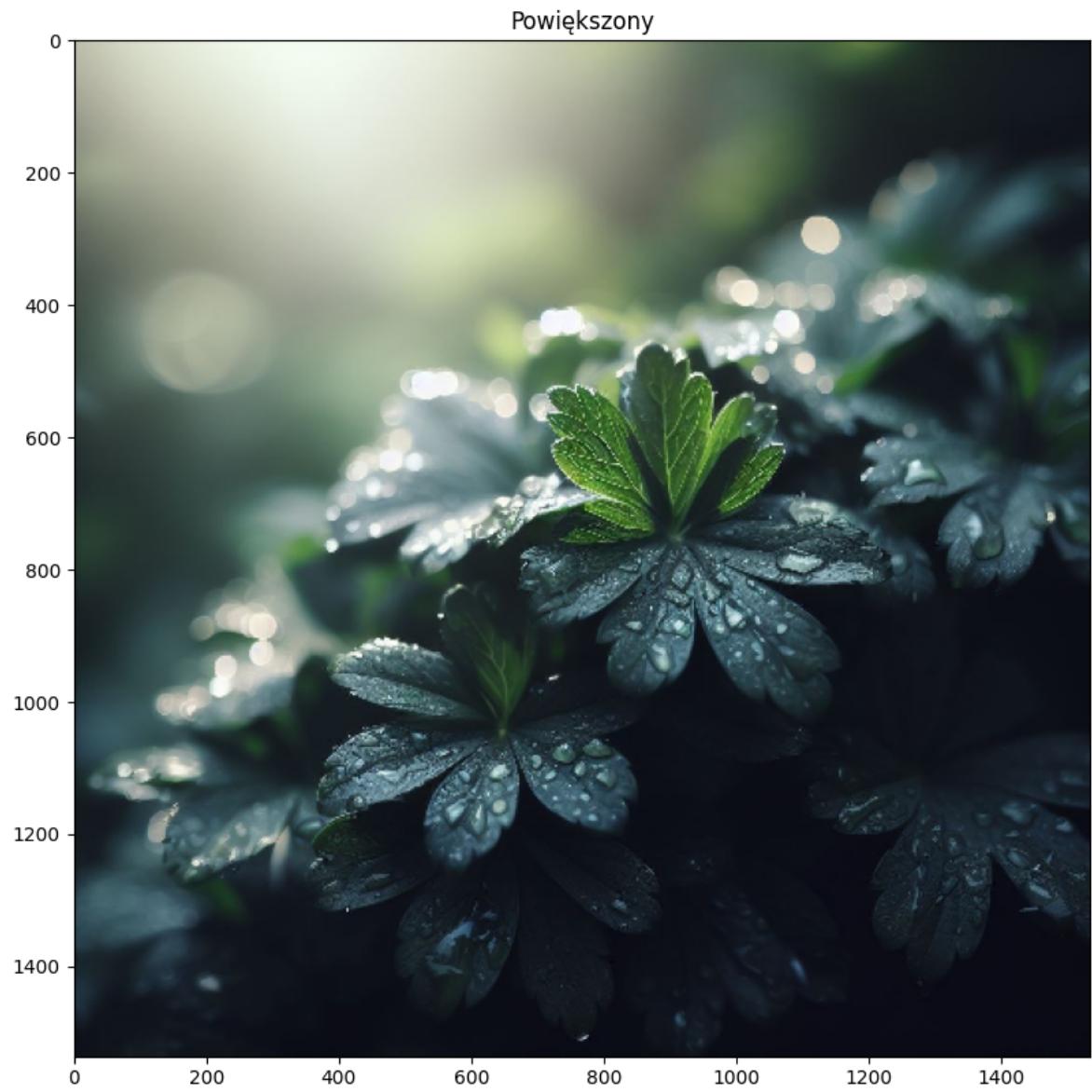
Rysunek 10: Oryginalny obraz

### 5.2.2 Pomniejszony



Rysunek 11: Pomniejszony obraz

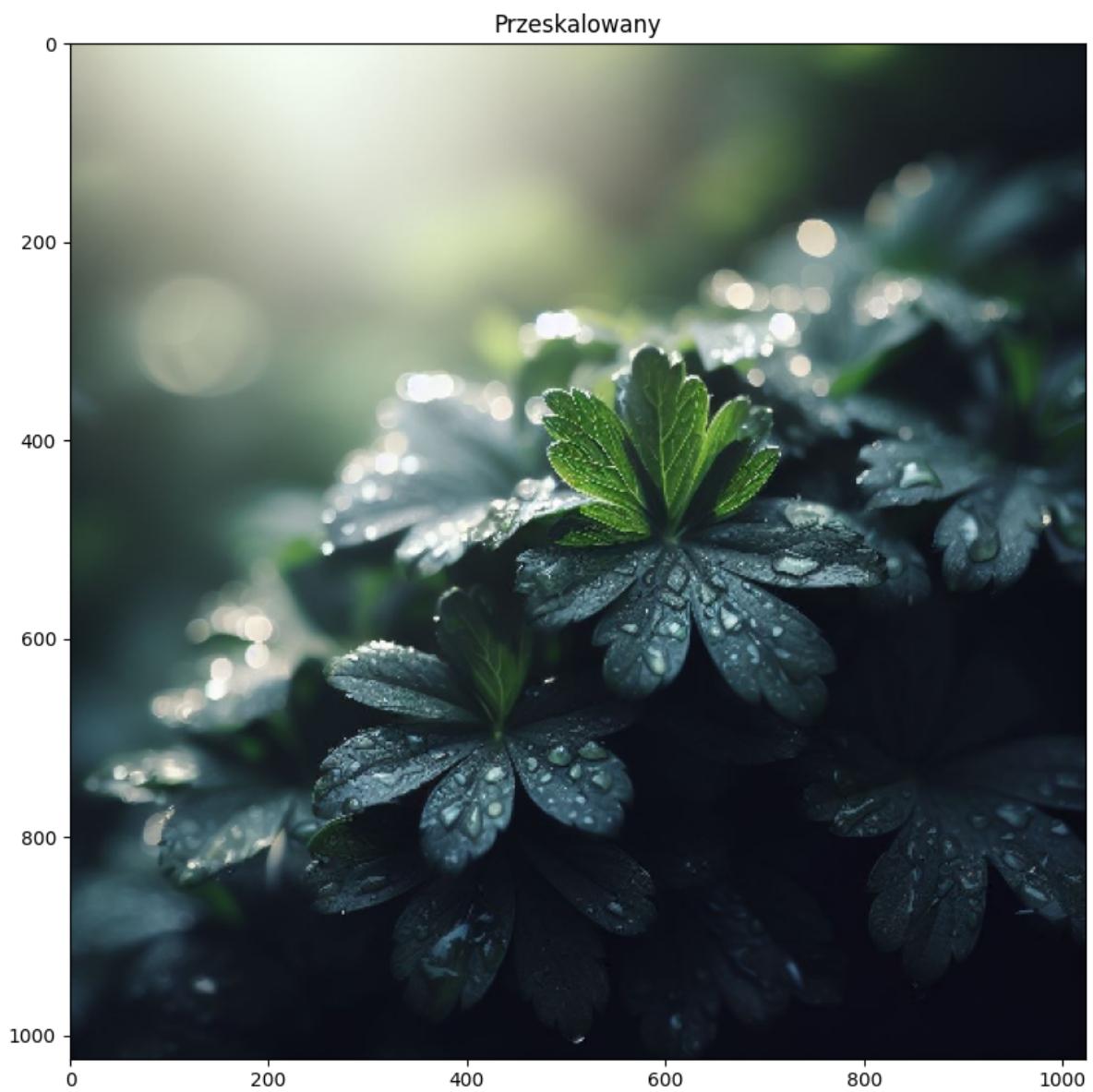
### 5.2.3 Powiększony



Rysunek 12: Powiększony obraz

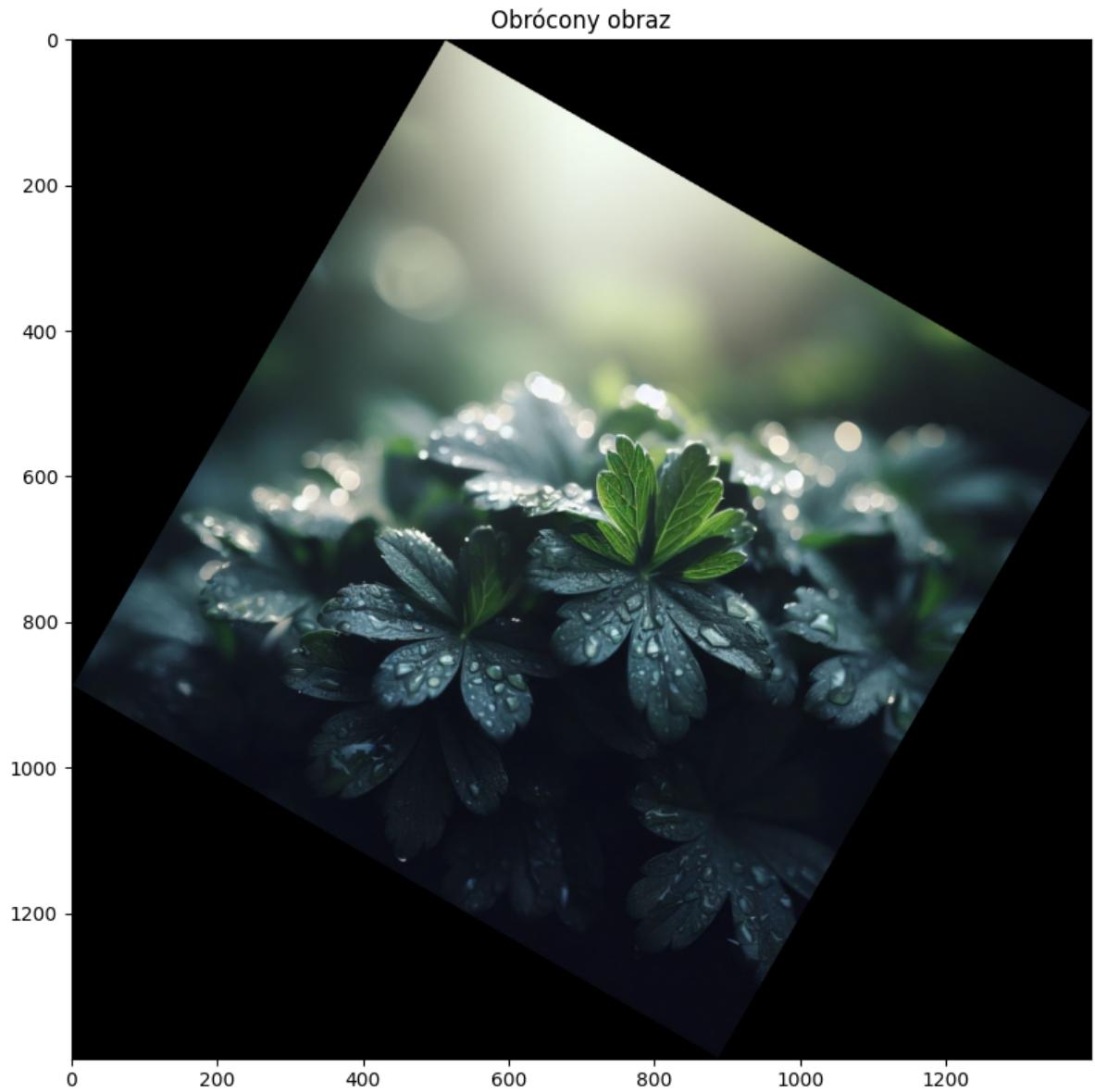
#### 5.2.4 Przeskalowany

Obraz przeskalowany powstał z powiększenia zmniejszonego obrazu, a następnie przywrócono go do oryginalnych rozmiarów.



Rysunek 13: Obraz przeskalowany

### 5.2.5 Obrócony o 30 stopni



Rysunek 14: Obraz obrócony o 30 stopni