

shift

March 22, 2025

Znalezienie przesunięcia dla każdego przekaźnika aby dopasować do siebie wartości. Bazowym przekaźnikiem jest nieuszkodzony_1.

```
[1]: from config import PIXEL_DATA_PATH_PREFIX, PROCESSED_DATA_PATH_PREFIX
import pandas as pd
import numpy as np
import scipy.signal
import cv2

def load_csv(file_path):
    """ Wczytuje plik CSV i zwraca DataFrame """
    df = pd.read_csv(file_path)
    df = df[['photo_id', 'pixels_brightness_sum']] # Pobieramy tylko potrzebne
    ↪kolumny
    df = df.set_index('photo_id').sort_index() # Ustawiamy photo_id jako indeks
    return df

def find_best_shift(base_series, target_series):
    """ Znajduje najlepsze przesunięcie target_series względem base_series """
    base_values = base_series.values - np.mean(base_series.values)
    target_values = target_series.values - np.mean(target_series.values)

    # Korelacja
    correlation = scipy.signal.correlate(base_values, target_values,
    ↪mode="full")
    shift_index = np.argmax(correlation) - (len(target_values) - 1)

    return shift_index

# Ścieżka bazowego przekaźnika według, którego przesuwamy indeksację pozostałych
base_path = f"{PIXEL_DATA_PATH_PREFIX}/nieuszkodzone/nieuszkodzony_1.csv"
base_df = load_csv(base_path)

shift_values = {}

# Znalezienie przesunięcia dla każdego przekaźnika
for state in ["nieuszkodzon", "uszkodzon"]:
    for relay_id in range(1, 6):
```

```

target_path = f"{PIXEL_DATA_PATH_PREFIX}/{state}e/{state}y_{relay_id}.
↪CSV"
target_df = load_csv(target_path)

best_shift = find_best_shift(base_df, target_df)
shift_values[f"{state}y_{relay_id}"] = best_shift

```

Przesunięcia względem przekaźnika *nieuszkodzony_1*: - *nieuszkodzony_1*: 0 - *nieuszkodzony_2*: 0 - *nieuszkodzony_3*: 3 - *nieuszkodzony_4*: 0 - *nieuszkodzony_5*: 0 - *uszkodzony_1*: -3 - *uszkodzony_2*: -15 - *uszkodzony_3*: 0 - *uszkodzony_4*: 1 - *uszkodzony_5*: 0

Przesunięcie plików .csv z sumami wartościami pikseli

```

[2]: from config import PIXEL_DATA_PATH_PREFIX, SHIFTED_PIXEL_DATA_PATH_PREFIX
import pandas as pd
import os

# Tworzenie folderów, jeżeli nie istnieją
os.makedirs(f"{SHIFTED_PIXEL_DATA_PATH_PREFIX}/nieuszkodzone", exist_ok=True)
os.makedirs(f"{SHIFTED_PIXEL_DATA_PATH_PREFIX}/uszkodzone", exist_ok=True)

for state in ["nieuszkodzon", "uszkodzon"]:
    for relay_id in range(1, 6):
        file_name = f"{state}y_{relay_id}"

        target_path = f"{PIXEL_DATA_PATH_PREFIX}/{state}e/{file_name}.csv"
        target_df = load_csv(target_path)

        # Znalezienie przesunięcia
        shift = shift_values[file_name]

        # Przesunięcie danych
        shifted_df = target_df.shift(periods=shift)

        # Zapisanie w odpowiedniej ścieżce
        shifted_path = f"{SHIFTED_PIXEL_DATA_PATH_PREFIX}/{state}e/{file_name}.
↪CSV"
        shifted_df.to_csv(shifted_path)

```

Wykresy z pliku `pixels.ipynb` ale dla przesuniętych danych:

```

[3]: import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import pandas as pd
from config import SHIFTED_PIXEL_DATA_PATH_PREFIX

# Formatuje oś Y, aby nie było notacji naukowej
def format_axis(y, pos):
    return '{:,}'.format(int(y)).replace(',', ' ')

```

```

# Przetwarza dane dla przekaźników uszkodzonych i nieuszkodzonych
for state in ["nieuszkodzon", "uszkodzon"]:
    for relay_id in range(1, 6):
        path = f"{SHIFTED_PIXEL_DATA_PATH_PREFIX}/{state}e/{state}y_{relay_id}.
        ↪CSV"

        # Wczytuje dane z pliku CSV
        df = pd.read_csv(path)

        photo_ids = df.iloc[:, 0] # Pobiera numery zdjęć
        photo_ids = [f"{i:03}" for i in photo_ids] # Formatuje numery zdjęć do
        ↪postaci trzycyfrowej (np. "001")
        pixels_sums = df.iloc[:, 1].to_list() # Pobiera sumy wartości pikseli

        fig, ax1 = plt.subplots(figsize=(16, 8)) # Tworzy wykres

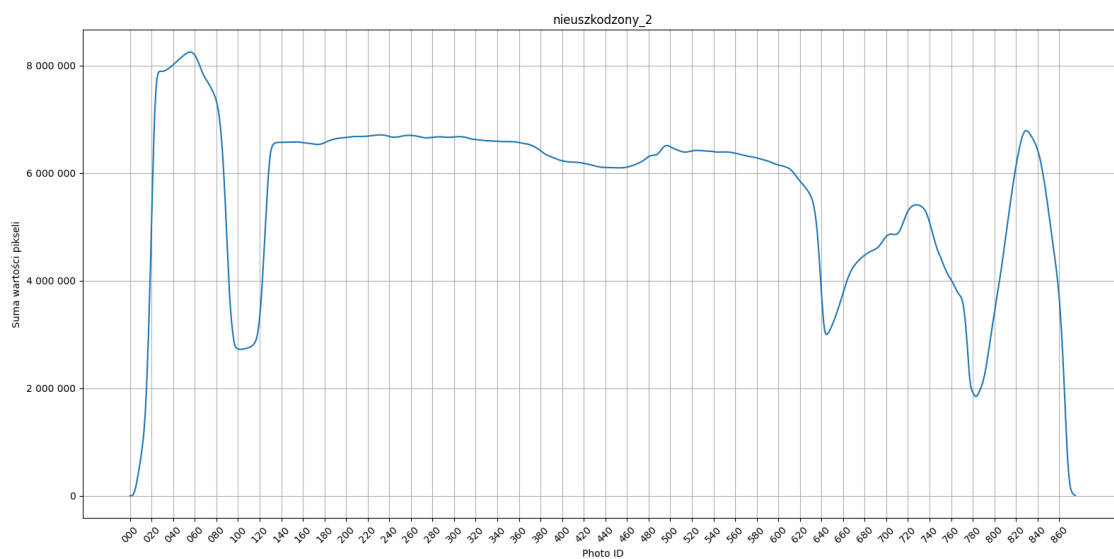
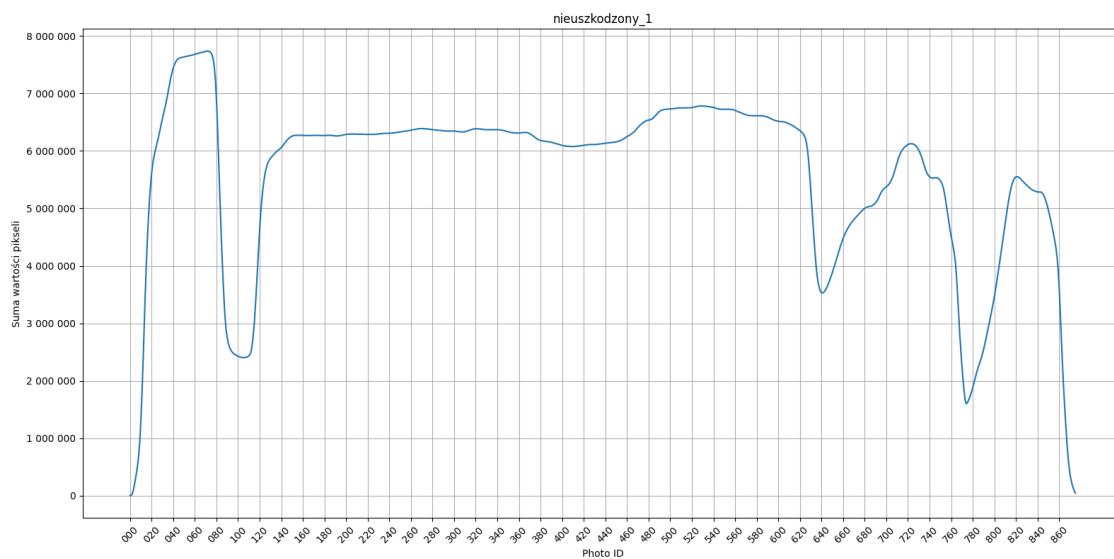
        # Wykres dla sum wartości pikseli
        ax1.plot(photo_ids, pixels_sums, label="Suma wartości pikseli",
        ↪color="tab:blue")
        ax1.set_xlabel("Photo ID")
        ax1.set_ylabel("Suma wartości pikseli")
        ax1.tick_params(axis='y')
        ax1.yaxis.set_major_formatter(FuncFormatter(format_axis)) #
        ↪Formatowanie osi Y

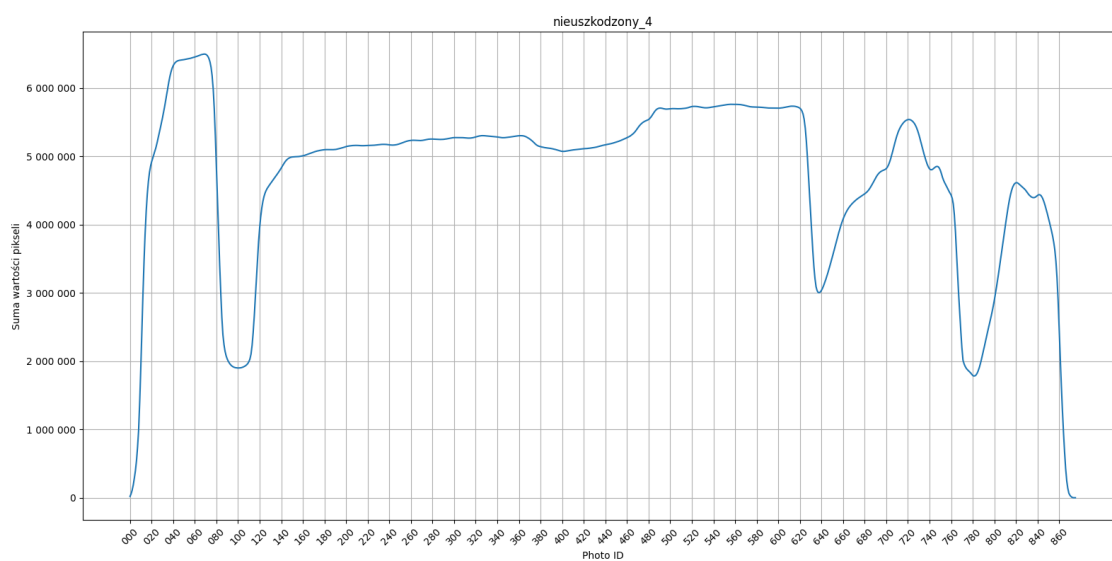
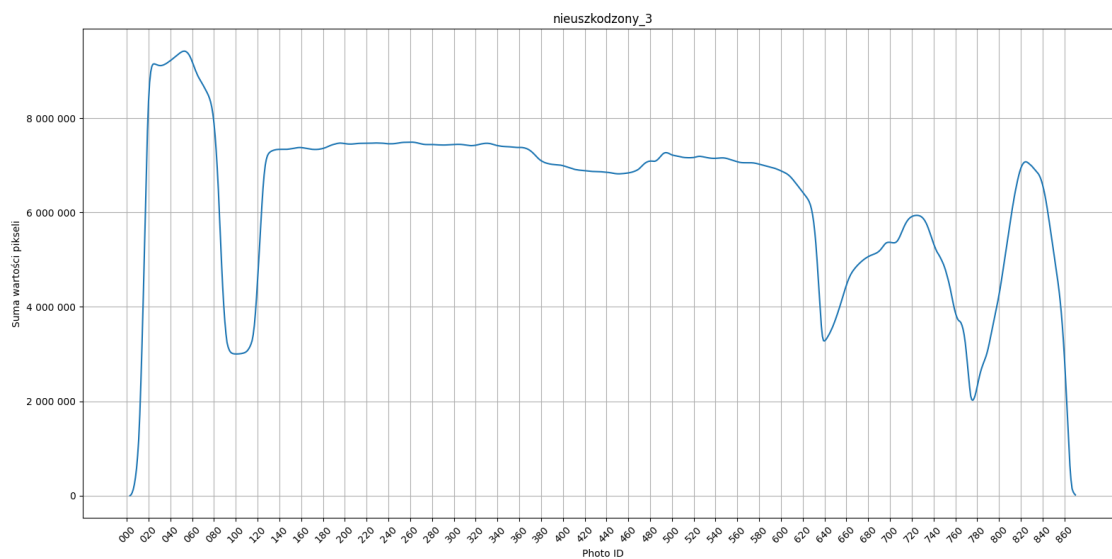
        plt.title(f"{state}y_{relay_id}") # Tytuł wykresu

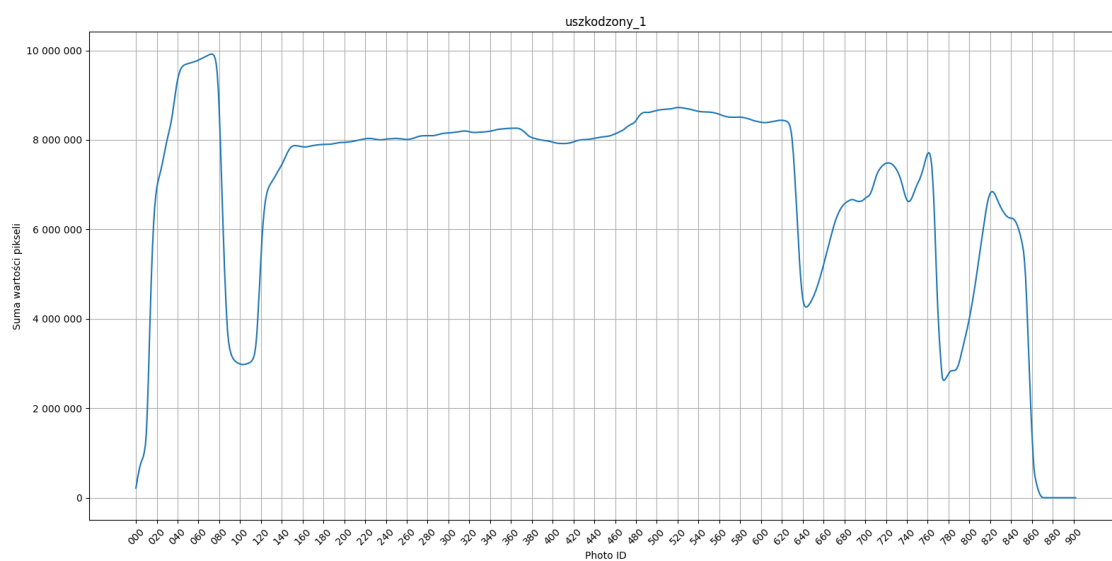
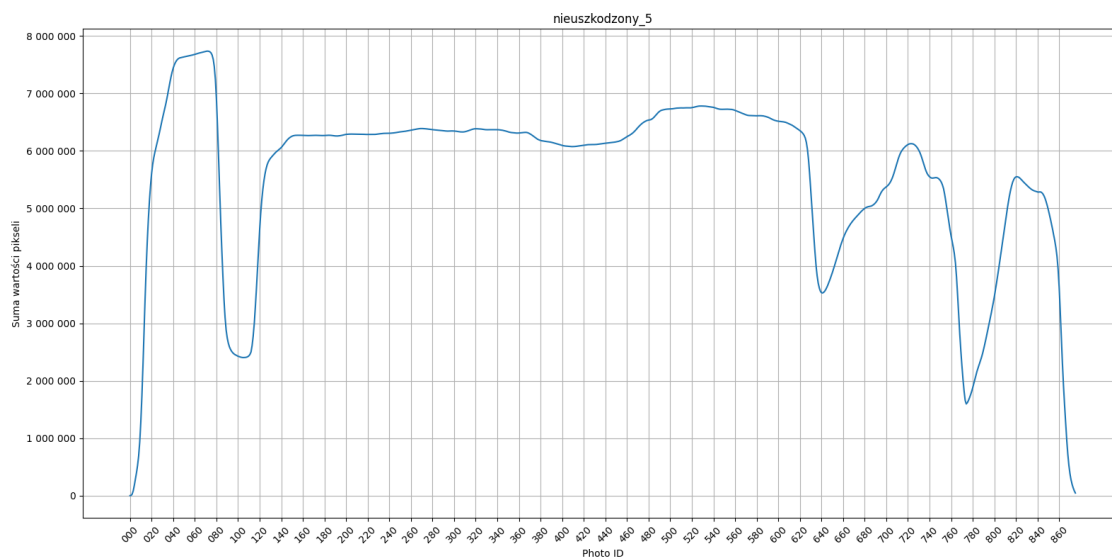
        # Ustawia oznaczenia na osi X co 20 zdjęć, aby poprawić czytelność
        ax1.set_xticks(range(0, len(photo_ids), 20))
        ax1.set_xticklabels([photo_ids[i] for i in range(0, len(photo_ids),
        ↪20)], rotation=45)

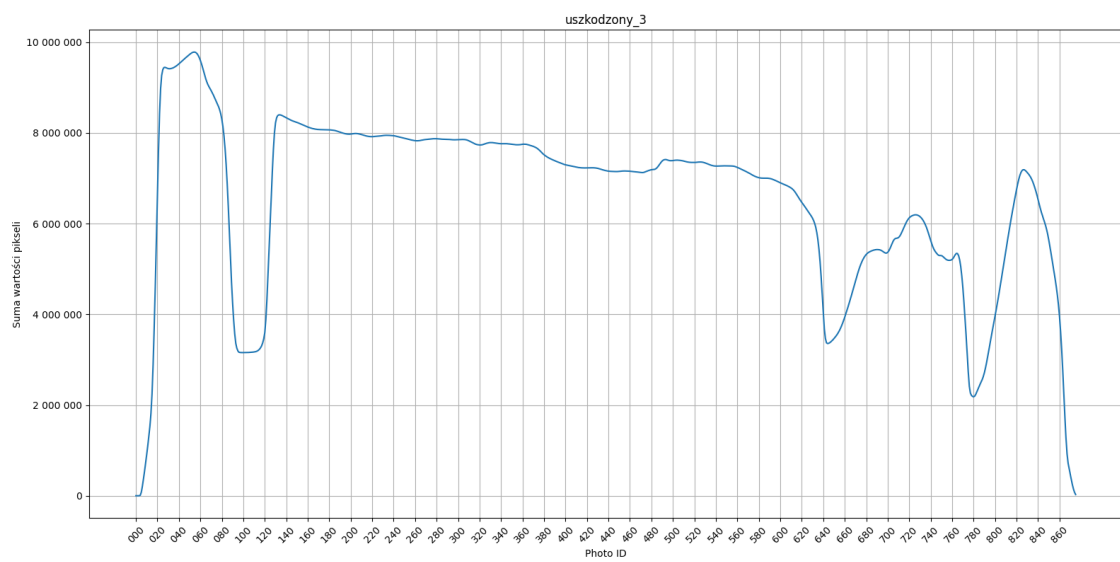
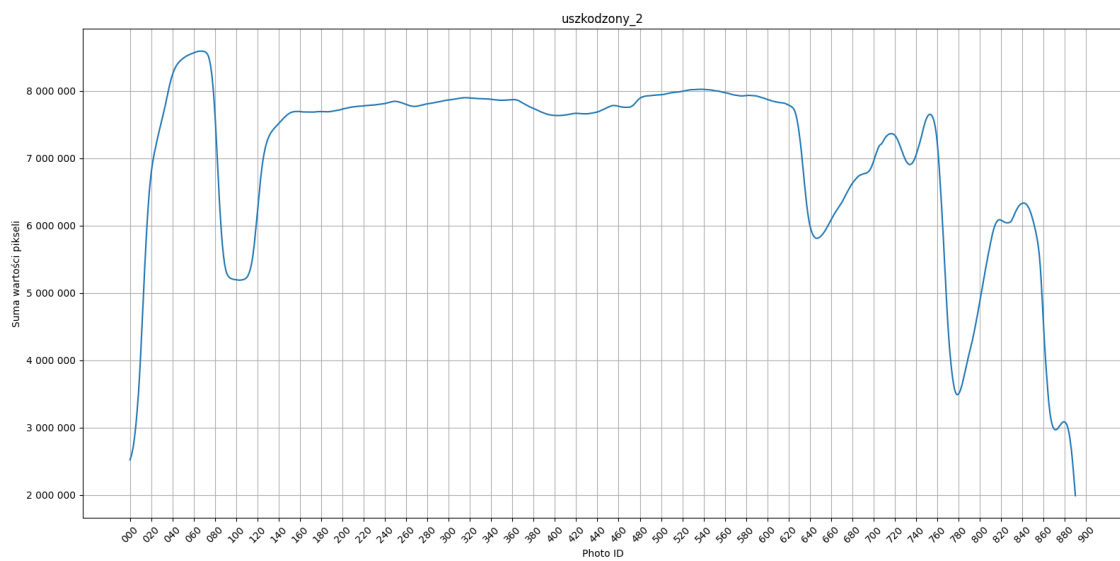
        ax1.grid(True) # Dodaje siatkę do wykresu
        plt.tight_layout()
        plt.show() # Wyświetla wykres

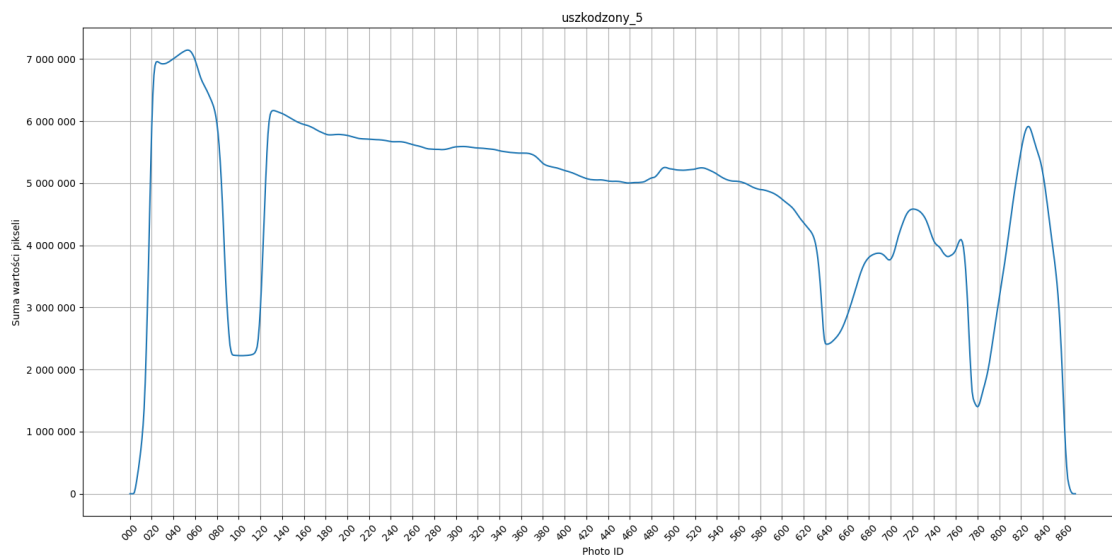
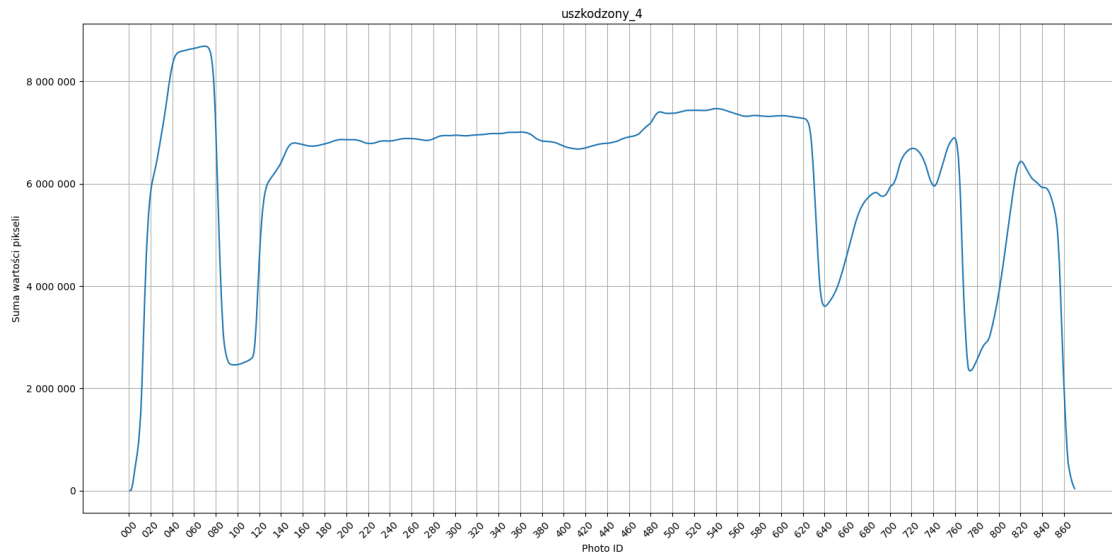
```











```
[4]: from matplotlib.lines import Line2D

# Formatuje oś Y, aby nie było notacji naukowej
def format_axis(y, pos):
    return '{:,}'.format(int(y)).replace(',', ' ')

plt.figure(figsize=(16, 8))

# red_shades = np.array([
#     (255, 99, 71), # Czerwony - Tomato
```

```

#     (255, 69, 0),    # Czerwony - Red-Orange
#     (220, 20, 60),   # Czerwony - Crimson
#     (139, 0, 0),     # Czerwony - Dark Red
#     (128, 0, 0)      # Czerwony - Maroon
# ]) / 255

# blue_shades = np.array([
#     (135, 206, 250),  # Niebieski - Light Sky Blue
#     (70, 130, 180),   # Niebieski - Steel Blue
#     (0, 0, 255),      # Niebieski - Blue
#     (0, 0, 139),      # Niebieski - Dark Blue
#     (25, 25, 112)     # Niebieski - Midnight Blue
# ]) / 255

# Przetwarza dane dla przekaźników uszkodzonych i nieuszkodzonych
for state in ["nieuszkodzon", "uszkodzon"]:
    for relay_id in range(1, 6):
        path = f"{SHIFTED_PIXEL_DATA_PATH_PREFIX}/{state}e/{state}y_{relay_id}.
↳ csv"

        # Wczytuje dane z pliku CSV
        df = pd.read_csv(path)

        photo_ids = df.iloc[:, 0] # Pobiera numery zdjęć
        photo_ids = [f"{i:03}" for i in photo_ids] # Formatuje numery zdjęć do
↳ postaci trzycyfrowej (np. "001")
        pixels_sums = df.iloc[:, 1].to_list() # Pobiera sumy wartości pikseli

        # color = red_shades[relay_id - 1] if state == "uszkodzon" else
↳ blue_shades[relay_id - 1]
        color = (1, 0, 0) if state == 'uszkodzon' else (0, 0, 1)

        plt.plot(photo_ids, pixels_sums, c=color)
        plt.xlabel("PhotoID")
        plt.ylabel("Suma wartości pikseli")
        plt.tick_params(axis='y')
        plt.gca().yaxis.set_major_formatter(FuncFormatter(format_axis))

        plt.title("Przesunięte obrazy")

        plt.gca().set_xticks(range(0, len(photo_ids), 20))
        plt.gca().set_xticklabels([photo_ids[i] for i in range(0,
↳ len(photo_ids), 20)], rotation=45)

        plt.grid(True)

legend_elements = [

```

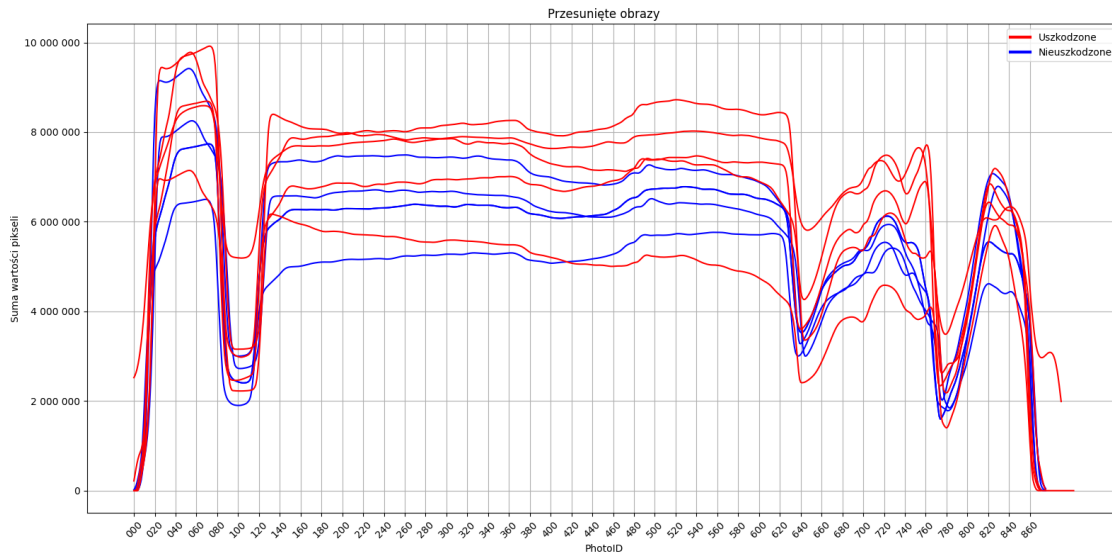
```

Line2D([0], [0], color='red', lw=3, label='Uszkodzone'),
Line2D([0], [0], color='blue', lw=3, label='Nieuszkodzone'),
]

plt.legend(handles=legend_elements, loc='best')

plt.tight_layout()
plt.show() # Wyświetla wykres

```



```

[5]: def get_photo(state, relay_id, id, folder_path, shift_values):
    shifted_id = id - shift_values[f'{state}y_{relay_id}'].item()
    full_path = f"{folder_path}/{state}e/{state}y_{relay_id}/
    ↳{state}y_{relay_id}-{shifted_id:03}.bmp"
    img = cv2.imread(full_path)
    return img

def img_to_grayscale(img):
    return 0.2989 * img[:, :, 0] + 0.5870 * img[:, :, 1] + 0.1140 * img[:, :, 2]

# Przekaznik o najniższej sumie wartości pikseli dla PhotoID = 600 +~_
↳przesunięcie
img1 = get_photo('uszkodzon', 5, 600, PROCESSED_DATA_PATH_PREFIX, shift_values)

# Przekaznik o najwyższej sumie wartości pikseli dla PhotoID = 600 +~_
↳przesunięcie
img2 = get_photo('uszkodzon', 1, 600, PROCESSED_DATA_PATH_PREFIX, shift_values)

# Transformacje do skali szarości

```

```

img1 = img_to_grayscale(img1).astype(np.uint8)
img2 = img_to_grayscale(img2).astype(np.uint8)

# Obraz o przekaźnika o najniższej sumie wartości pikseli na poziomie cewki (ID_
↪120-620)
plt.imshow(img1, cmap='gray')
plt.title("Obraz 1 - przekaźnik o najniższej sumie wartości pikseli")
plt.axis('off')
plt.show()

# Obraz o przekaźnika o najwyższej sumie wartości pikseli na poziomie cewki (ID_
↪120-620)
plt.imshow(img2, cmap='gray')
plt.title("Obraz 2 - przekaźnik o najwyższej sumie wartości pikseli")
plt.axis('off')
plt.show()

# Binaryzacja z thresholdem 0
img3 = np.where(img1 > 0, 255, 0)
img4 = np.where(img2 > 0, 255, 0)

# Obliczanie różnic
result1 = np.maximum(img2 - img1, 0)
result2 = np.maximum(img4 - img3, 0)

# Obraz różnic w wartościach pikseli (wartości 0-255) między przekaźnikami o_
↪najwyższej i najniższej sumie wartości pikseli na poziomie cewki
plt.imshow(result1, cmap='gray')
plt.title("Obraz 2 - Obraz 1")
plt.axis('off')
plt.show()

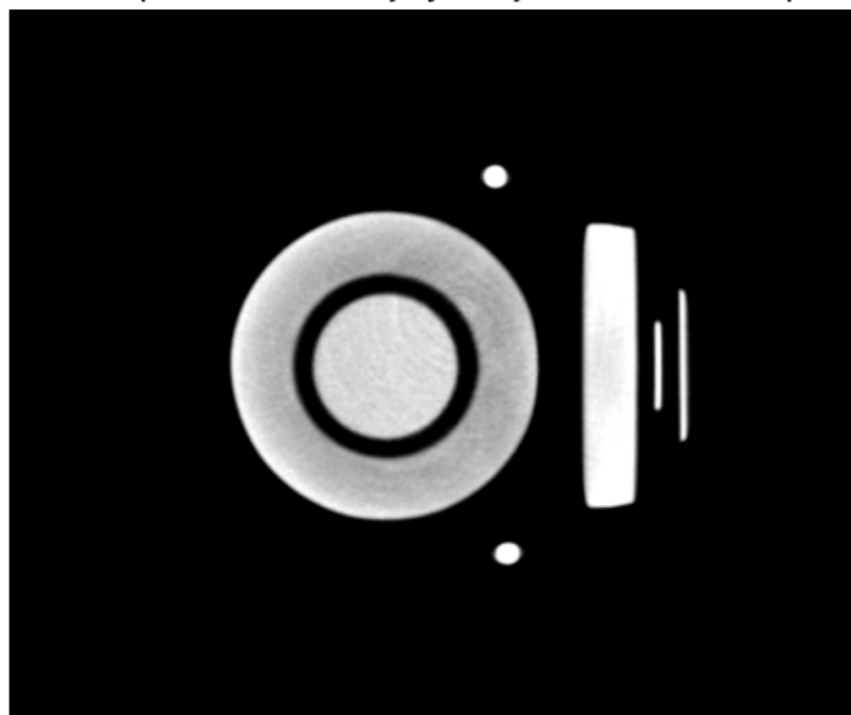
# Obraz różnic w występowaniach pikseli (wartości 0 albo 255) między_
↪przekaźnikami o najwyższej i najniższej sumie wartości pikseli na poziomie_
↪cewki
plt.imshow(result2, cmap='gray')
plt.title("Obraz 2 - Obraz 1")
plt.axis('off')
plt.show()

```

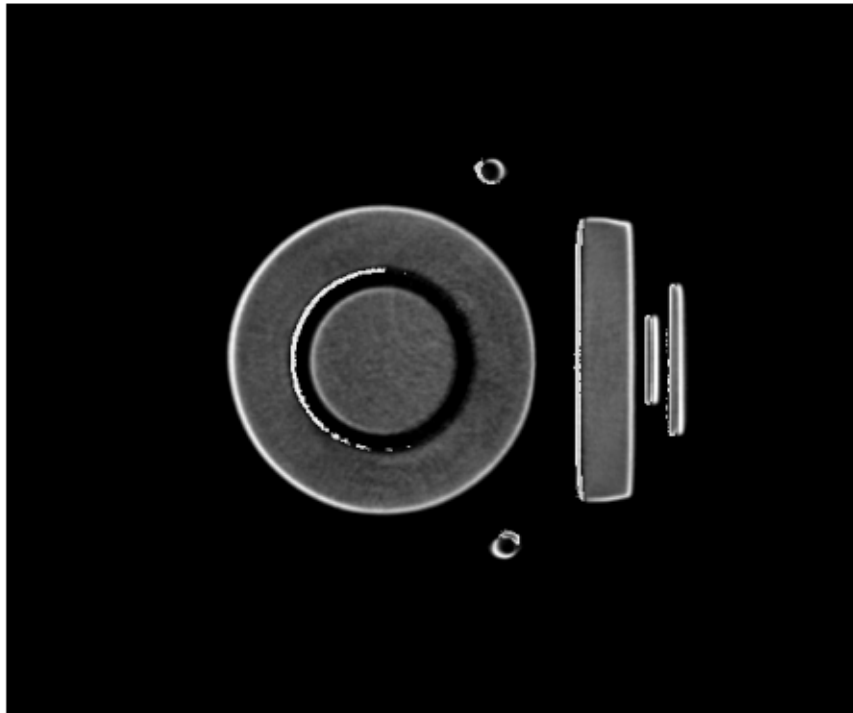
Obraz 1 - przekaźnik o najniższej sumie wartości pikseli



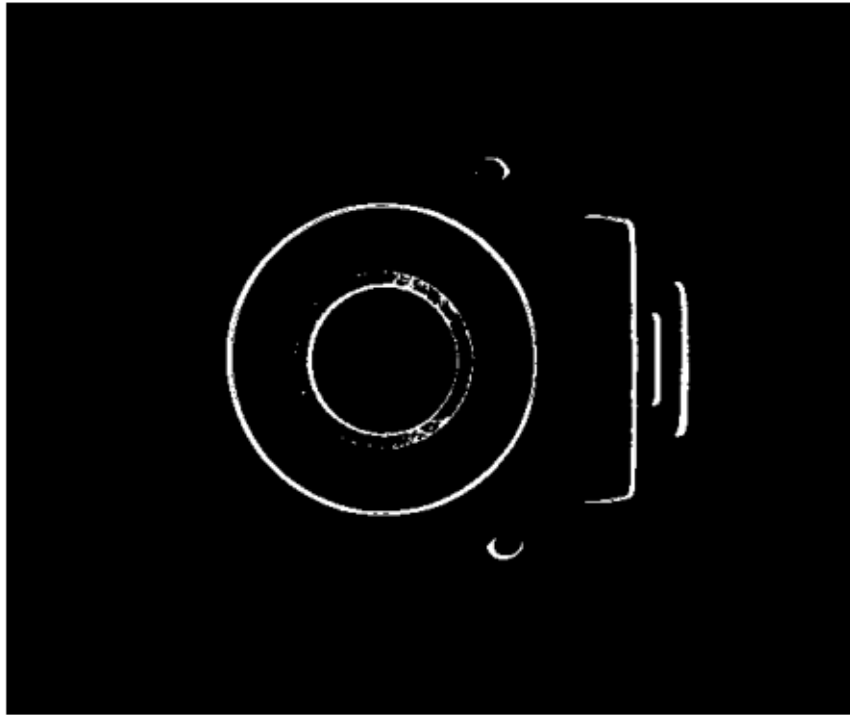
Obraz 2 - przekaźnik o najwyższej sumie wartości pikseli



Obraz 2 - Obraz 1



Obraz 2 - Obraz 1



0.0.1 Dla uśrednionych danych dla przekaźników nieuszkodzonych i uszkodzonych

```
[6]: from config import SHIFTED_PIXEL_DATA_PATH_PREFIX
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mticker

def load_and_average_data(status: str) -> pd.DataFrame:
    """Wczytuje i uśrednia dane dla określonego statusu przekaźnika (uszkodzony/
    ↪nieuszkodzony)."""
    data_list = [
        pd.read_csv(f"{SHIFTED_PIXEL_DATA_PATH_PREFIX}/{status}/
    ↪{'nieuszkodzony' if status == 'nieuszkodzone' else 'uszkodzony'}_{relay_id}.
    ↪csv")
        for relay_id in range(1, 6)
    ]
    df = pd.concat(data_list, axis=1).T.groupby(level=0).mean().T
    df["status"] = "Nieuszkodzony" if status == "nieuszkodzone" else
    ↪"Uszkodzony"
    return df
```

```

# Wczytanie danych
df_nieuszkodzony = load_and_average_data("nieuszkodzone")
df_uszkodzony = load_and_average_data("uszkodzone")

# Połączenie zbiorów
df = pd.concat([df_nieuszkodzony, df_uszkodzony], ignore_index=True)

pd.options.display.float_format = lambda x: f"{x:,.2f}".replace(",", " ") #_
    ↪Dwa miejsca po przecinku, spacja jako separator
print("Statystyki dla nieuszkodzonych:")
print(df[df["status"] == "Nieuszkodzony"]["pixels_brightness_sum"].describe().
    ↪drop("count"))
print("\nStatystyki dla uszkodzonych:")
print(df[df["status"] == "Uszkodzony"]["pixels_brightness_sum"].describe().
    ↪drop("count"))

# Formatowanie osi
def format_axis(value, _):
    return f'{int(value):,}'.replace(',', ' ')

# Wykres liniowy
plt.figure(figsize=(12, 6))
sns.lineplot(x=df_nieuszkodzony["photo_id"],_
    ↪y=df_nieuszkodzony["pixels_brightness_sum"], label="Nieuszkodzony",_
    ↪color="blue")
sns.lineplot(x=df_uszkodzony["photo_id"],_
    ↪y=df_uszkodzony["pixels_brightness_sum"], label="Uszkodzony", color="red")

plt.title("Zmiana Sumy Jasności Pikseli w Zależności od Photo ID")
plt.xlabel("Photo ID")
plt.ylabel("Suma Jasności Pikseli")
plt.legend()
plt.grid()

plt.gca().yaxis.set_major_formatter(mticker.FuncFormatter(format_axis))

plt.show()

# Wykres histogramów
plt.figure(figsize=(10, 6))
sns.histplot(df[df["status"] == "Nieuszkodzony"]["pixels_brightness_sum"],_
    ↪kde=True, color="blue", label="Nieuszkodzony", bins=25)

```

```

sns.histplot(df[df["status"] == "Uszkodzony"]["pixels_brightness_sum"],
             kde=True, color="red", label="Uszkodzony", bins=25)

plt.legend()
plt.title("Histogram Sumy Jasności Pikseli (Uśrednione)")
plt.xlabel("Suma Jasności Pikseli")
plt.ylabel("Częstotliwość")

plt.gca().xaxis.set_major_formatter(mticker.FuncFormatter(format_axis))

plt.show()

# Wykres Boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x="status", y="pixels_brightness_sum", data=df, palette="Set2",
            hue="status")

plt.title("Boxplot Sumy Jasności Pikseli (Uśrednione)")
plt.xlabel("Status")
plt.ylabel("Suma Jasności Pikseli")

plt.gca().yaxis.set_major_formatter(mticker.FuncFormatter(format_axis))

plt.show()

```

Statystyki dla nieuszkodzonych:

```

mean    5 594 568.18
std      1 520 600.42
min       4 889.25
25%      5 122 655.30
50%      6 241 431.00
75%      6 415 892.25
max      7 880 316.40
Name: pixels_brightness_sum, dtype: float64

```

Statystyki dla uszkodzonych:

```

mean    6 182 673.06
std      1 862 462.30
min       0.00
25%      5 705 279.80
50%      7 028 826.60
75%      7 259 853.40
max      8 761 440.20
Name: pixels_brightness_sum, dtype: float64

```

