

Sprawozdanie z projektu - symulator tomografu komputerowego

1 Skład grupy

- Zuzanna Piniarska 136782
- Mateusz Kałamoniak 136730

2 Opis projektu

2.1 Zastosowany model tomografu: równoległy

2.2 Zastosowany język programowania i biblioteki

Język: **Python3**

Biblioteki:

1. skimage
2. numpy
3. PIL
4. sklearn
5. scipy
6. pydicom
7. matplotlib

2.3 Główne funkcje programu

2.3.1 Pozyskiwanie odczytów dla poszczególnych detektorów

```
def transform(self, image):
    sinogram = np.zeros((self.steps, self.detectors), dtype='float64')
    self.get_circle(image)
    x, y = image.shape
    x_circle, y_circle, r_circle = self.circle
    for s in range(self.steps):
        for d in range(self.detectors):
            x1 = r_circle * np.cos(np.deg2rad(s + (self.l_angle / 2)
            - d * (self.l_angle / (self.detectors - 1))))
            y1 = r_circle * np.sin(np.deg2rad(s + (self.l_angle / 2)
            - d * (self.l_angle / (self.detectors - 1))))
            x1 = int(x1) + (x // 2)
            y1 = int(y1) + (y // 2)

            x2 = r_circle * np.cos(
                np.deg2rad(s + 180 - (self.l_angle / 2)
                + d * (self.l_angle / (self.detectors - 1))))
            y2 = r_circle * np.sin(
                np.deg2rad(s + 180 - (self.l_angle / 2)
                + d * (self.l_angle / (self.detectors - 1))))
            x2 = int(x2) + (x // 2)
            y2 = int(y2) + (y // 2)

            xx, yy = RadonTransform.bresenham(x1, x2, y1, y2)
            sinogram[s, d] = sum(image[xx, yy])
    return sinogram
```

2.3.2 Filtrowanie sinogramu

```
def get_mask(self):
    mask_size = int(np.floor(self.detectors / 2))
    mask = np.zeros(mask_size)
    mask_center = int(np.floor(mask_size / 2))
    for i in range(mask_size):
        j = i - mask_center
        if j % 2 != 0:
            mask[i] = (-4 / np.pi ** 2) / (j ** 2)
    mask[mask_center] = 1
```

```
return mask
```

```
def filtering(self, image):  
    filtered = np.zeros((self.steps, self.detectors))  
    mask = self.get_mask()  
  
    for step in range(self.steps):  
        filtered[step] = signal.convolve(image[step], mask,  
                                         mode='same', method='direct')  
    return filtered
```

2.3.3 Normalizacja obrazu

```
def normalize_pixels(self, image, pixels, size):  
    for x in range(size):  
        for y in range(size):  
            if pixels[x, y] > 0:  
                image[x, y] = image[x, y] / pixels[x, y]  
    return image
```

2.3.4 Obliczanie RMSE

```
def RMSE(image, inversed_image):  
    image = image.copy()  
    inversed_image = inversed_image.copy()  
    x, y = image.shape  
    for i in range(x):  
        for j in range(y):  
            image[i][j] /= 255  
            inversed_image[i][j] /= 255  
    return metrics.mean_squared_error(image, inversed_image,  
                                     multioutput='uniform_average', squared=False)
```

2.3.5 Odczyt i zapis w formacie DICOM

```
@staticmethod  
def read_dicom(file):  
    dataset = dicom.dcmread(file)  
    print(file)  
    print(dataset.PatientName)
```

```

print(dataset.StudyDate)
print(dataset.ImageComments)

@staticmethod
def write_dicom(filename, file, name, date, comments):
    dataset = dicom.dcmread(file)
    dataset.PatientName = name
    dataset.StudyDate = date
    dataset.ImageComments = comments
    dataset.save_as(filename)
    print('DICOM file saved')

```

2.4 Wyniki eksperymentu

W przypadku eksperymentu ze zmienną ilością iteracji wykonywania transformaty radona, większa liczba kroków znacznie wpływała na dokładność obrazu. Według mnie najlepsza czytelność została osiągnięta dla 6. obserwacji - dla wartości 540. Wartość RMSE wynosi wtedy 2,15. Najniższa uzyskana wartość RMSE to 2,10 jednak te obrazy nieznacznie się względem siebie różnią. Można śmiało stwierdzić jednak, że zbyt mała ilość iteracji przy wykonywaniu algorytmu jest szkodliwa dla uzyskanego obrazu i nie należy wykonywać mniej niż 360 kroków.

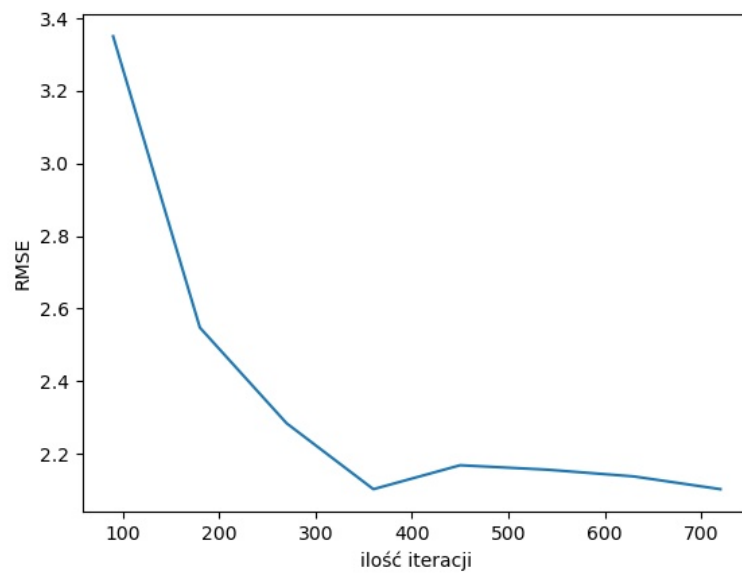
Dla parametru rozpiętości katowej optymalna wartość błędu średniokwadratowego jest osiągnięta dla 90 stopni. Większe wartości prowadzi do wyższej wartości błędu ze względu na zbyt szeroki kat co prowadzi do powstawania widocznych linii na przetworzonym obrazie

Wraz ze wzrostem ilości detektorów błąd średniokwadratowy ciągle maleje. Dla 180 detektorów wynik działania transformaty jest dobrą aproksymacją oryginalnego obrazu. Od około 270 detektorów zmiana wizualna jest nieznaczna. Rośnie za to czas wykonywania obliczeń, dlatego ilość detektorów równa 180 lub 270 można uznać za optymalną.

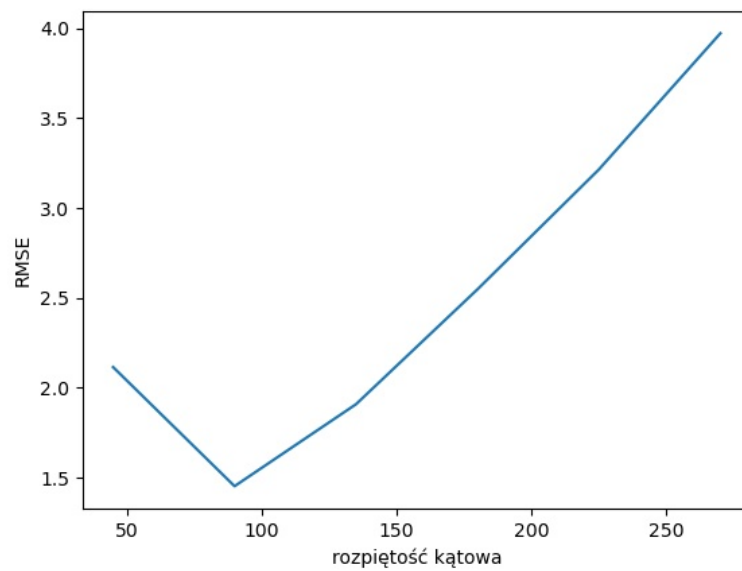
Poniżej znajdują się wykresy wizualizujące wyniki eksperymentu.

Wykonany został też eksperyment, w którym dla takich samych parametrów wykonano obraz z oraz bez filtrowania sinogramu. Obraz bez filtrowania wygląda znacznie gorzej a wartość RMSE wynosi aż 83.76277468645526. Nie jest to korzystny wynik. Zastosowanie filtra przy tych samych parametrach dało błąd średniokwadratowy równy 1.5092168255763534 - znacznie lepszy wynik. Eksperyment dowodzi tego, że filtrowanie sinogramu jest konieczną operacją przy wykonywaniu obliczeń. Jego brak bardzo pogorszył jakość końcowego efektu przetwarzania.

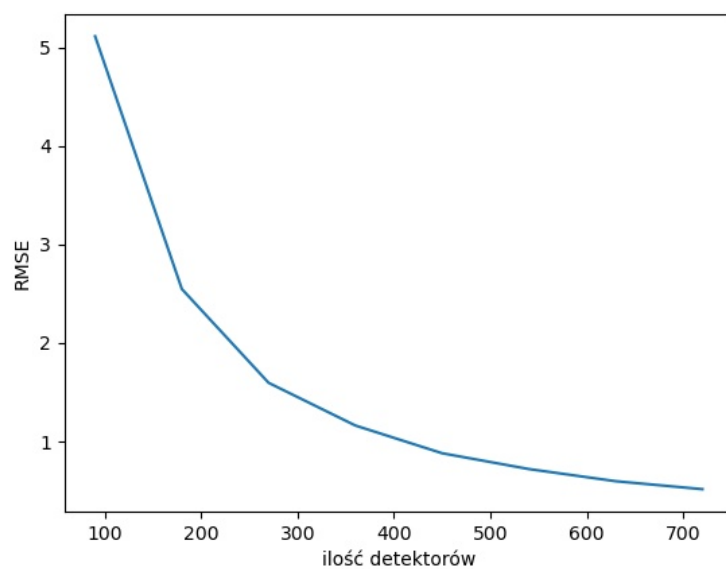
Poniżej znajdują się zdjęcia finalnych obrazów z filtrem oraz bez filtra na sinogramie.



Rysunek 1: Wykres RMSE ze zmiana ilości kroków



Rysunek 2: Wykres RMSE ze zmiana rozpiętości katowej



Rysunek 3: Wykres RMSE ze zmiana ilości detektorów

