

,

Informatyka w medycynie

Tomograf

30 marca 2021

Autorzy: **Jan Kolanowski** 141247
Zuzanna Trafas 141329

Zastosowany model tomografu: równoległy

Zastosowany język programowania: Python (streamlit, numpy, math, scipy.fftpack, pydicom, Pillow, matplotlib)

1 Opis funkcji programu

1.1 Pozyskiwanie odczytów dla poszczególnych detektorów

```
def radon_transform(image, alpha, detector_number, span):
    scan_number = int(180 / alpha)
    angles = np.linspace(0, 180, scan_number)
    sinogram = np.zeros((scan_number, detector_number))
    for i, angle in enumerate(angles):
        emitter_coords = calculate_coords(image, angle,
                                            detector_number, span, detector=False)

        detector_coords = calculate_coords(image, angle,
                                            detector_number, span, detector=True)

        bresenham_lines = [bresenham(x1, y1, x2, y2) for (x1, y1), (x2, y2)
                           in zip(emitter_coords, detector_coords)]

        results = []
        for line in bresenham_lines:
            results.append(np.sum(image[tuple(line)]))
        sinogram[i] = normalize(results)

    return sinogram
```

Dla podanego kąta angle (kąt zmienia się o alpha, przekazywane jako parametr) wyliczamy pozycję emiterów i detektorów (emitter_coords, detector_coords). Następnie wytyczamy linię między emiterami, a detektorami, korzystając z algorytmu Bresenhama. Suma danych z linii emiter-detektor jest wartością odczytu tego detektora.

1.2 Filtrowanie sinogramu

```
def create_mask(size):
    mask = np.zeros(size)
    nominator = (-4 / pow(pi, 2))
    center = int(size / 2)
    mask[center] = 1.0
    for i in range(center + 1, len(mask)):
        dist = i - center
        if dist % 2 == 0:
            mask[i] = 0.0
        else:
            mask[i] = nominator / pow(dist, 2)
    mask[center - dist] = mask[i]
```

```

    return mask

if filter:
    mask = create_mask(detector_number)
    for i in range(scan_number):
        sinogram[i, :] = np.convolve(sinogram[i, :], mask, 'same')
    sinogram = tresh(np.real(sinogram), 0, 1)

```

Używamy prostego filtra eliminującego rozmycie, jest on symetryczny, dzięki temu wystarczy, że zostanie wyliczona tylko i wyłącznie połowa maski według poniższego wzory:

$$\begin{aligned}
 h[0] &= 1 \\
 h[k] &= 0 \text{ dla } k \text{ parzystych} \\
 h[k] &= \frac{-4}{k^2\pi^2} \text{ dla } k \text{ nieparzystych}
 \end{aligned}$$

Następnie dokonujemy konwolucji otrzymanej maski wraz z sinogramem w celu eliminacji rozmycia.

1.3 Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

```

def inverse_radon_transform(sinogram, size, span, filter=True):
    if filter:
        sinogram = apply_filter(sinogram)
    scan_number, detector_number = sinogram.shape
    pil_image = Image.new('L', size)
    pil_image = make_square(pil_image)
    image = np.array(pil_image).astype('float64')
    count = image.copy()
    angles = np.linspace(0, 180, scan_number)
    for i, angle in enumerate(angles):
        emitter_coords = calculate_coords(image, angle,
                                            detector_number, span, detector=False)

        detector_coords = calculate_coords(image, angle,
                                             detector_number, span, detector=True)

        bresenham_lines = [np.array(bresenham(x1, y1, x2, y2))
                           for (x1, y1), (x2, y2) in
                           zip(emitter_coords, detector_coords)]

        for j, line in enumerate(bresenham_lines):
            image[tuple(line)] += sinogram[i][j]
            count[tuple(line)] += 1

```

```

count[count == 0] = 1
image = image / count
image = normalize(image)
image = reshape_to_original(image, size)
return image

```

1.4 Wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego

Funkcja wykorzystywana do wyliczania RMSE:

```

def rmsdiff(im1, im2):
    return np.sqrt(np.mean((im1-im2)**2))

```

Dzięki wykorzystaniu numpy możemy z łatwością wykonywać operacje na macierzach. Dla każdego piksela obliczamy kwadrat różnicy między jednym a drugim obrazem, następnie obliczamy pierwiastek z wartości średniej.

1.5 Odczyt i zapis plików DICOM

W celu obsługi plików DICOM zastosowaliśmy bibliotekę PyDicom, która umożliwia nam tworzenie oraz czytanie tego typu plików. Na chwilę obecną umożliwiamy zapis pliku wraz z obrazem, imieniem i nazwiskiem pacjenta, datą wydania oraz komentarzem lekarza. Stworzony plik DICOM został sprawdzony [tutaj](#). Czytanie:

```

if file.name.lower().endswith('.dcm'):
    ds = pydicom.read_file(file, force=True)
    img = Image.fromarray(ds.pixel_array).convert('L')

```

Tworzenie:

```

def create_dicom(path, image, meta):
    ds = Dataset()
    ds.MediaStorageSOPClassUID = MRImageStorage
    ds.MediaStorageSOPInstanceUID = generate_uid()
    ds.TransferSyntaxUID = pydicom.uid.ExplicitVRLittleEndian

    fd = FileDataset(path, {}, file_meta=ds, preamble=b'\0' * 128)
    fd.is_little_endian = True
    fd.is_implicit_VR = False

    fd.SOPClassUID = MRImageStorage
    fd.PatientName = 'Test^Firstname'
    fd.PatientID = '123456'
    now = datetime.datetime.now()
    fd.StudyDate = now.strftime('%Y%m%d')

```

```

fd.Modality = 'MR'
fd.SeriesInstanceUID = generate_uid()
fd.StudyInstanceUID = generate_uid()
fd.FrameOfReferenceUID = generate_uid()

fd.BitsStored = 16
fd.BitsAllocated = 16
fd.SamplesPerPixel = 1
fd.HighBit = 15

fd.ImagesInAcquisition = '1'
fd.Rows = image.shape[0]
fd.Columns = image.shape[1]
fd.InstanceNumber = 1

fd.ImagePositionPatient = r'0\0\1'
fd.ImageOrientationPatient = r'1\0\0\0\1\0'
fd.ImageType = r'ORIGINAL\PRIMARY\AXIAL'

fd.RescaleIntercept = '0'
fd.RescaleSlope = '1'
fd.PixelSpacing = r'1\1'
fd.PhotometricInterpretation = 'MONOCHROME2'
fd.PixelRepresentation = 1

for key, value in meta.items():
    setattr(fd, key, value)

validate_file_meta(fd.file_meta, enforce_standard=True)

fd.PixelData = (image * 255).astype(np.uint16).tobytes()

fd.save_as(path)

```

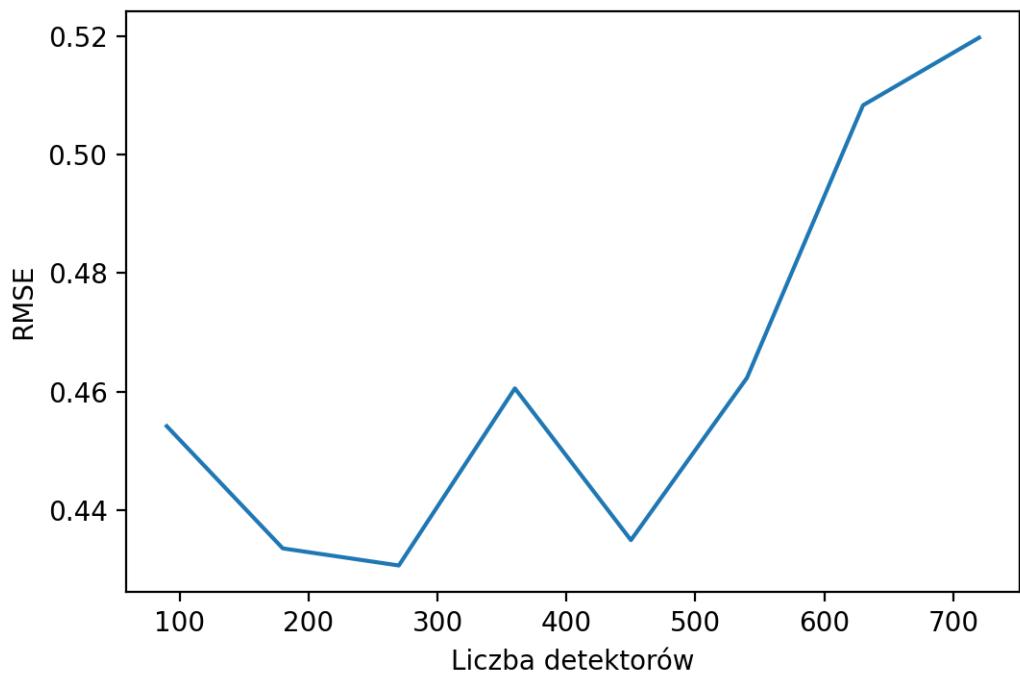
2 Wpływ poszczególnych parametrów (liczba detektorów, liczba skanów, rozpiętość stożka/wachlarza z detektorami) na jakość obrazu wynikowego

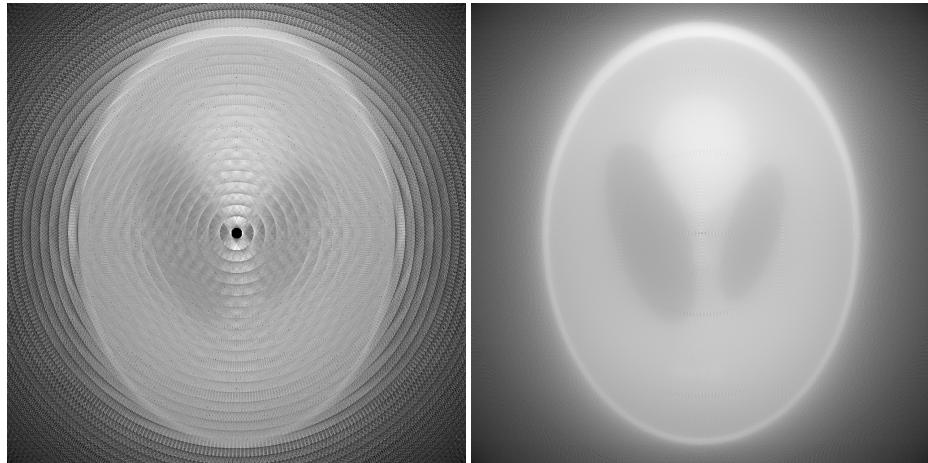
Dla wszystkich parametrów zmiana wartości RMSE jest niewielka, wynika to z niskiej jakości obrazu bez filtrowania. Widzimy jednak kilka tendencji:

- im większa liczba detektorów, tym wyższe RMSE. Wynika to prawdopodobnie z nierregularności obrazu z niższą ich liczbą, gdzie niektóre piksele które powinny być czarne, pozostają czarne.

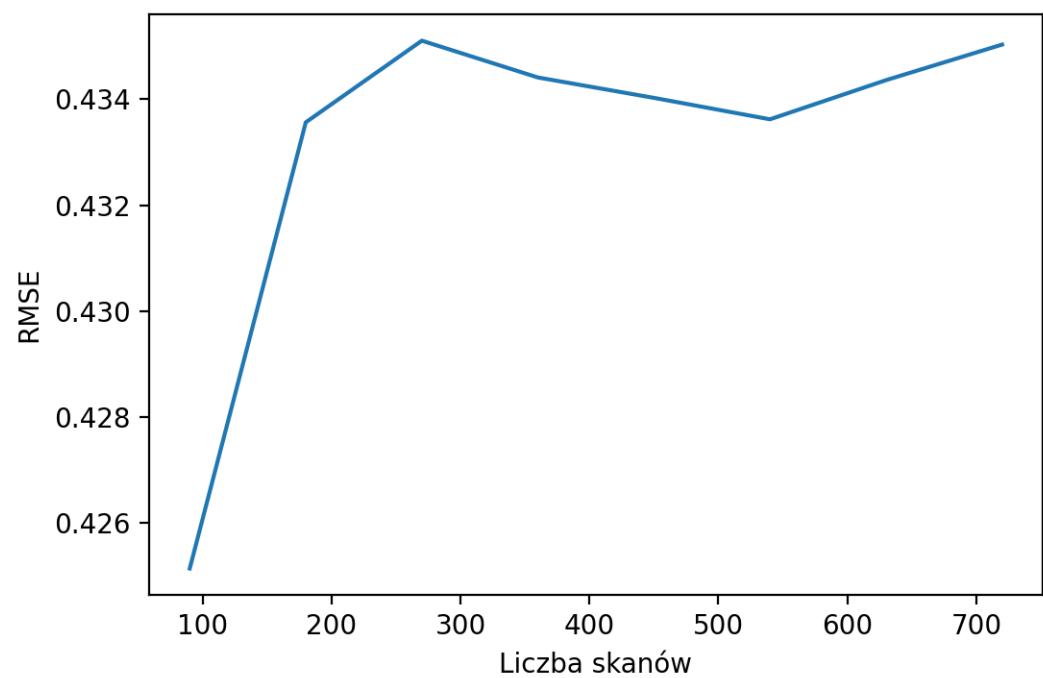
- taką samą zależność widzimy dla ilości skanów, chociaż tutaj różnica wartości jest niższa. Powód wydaje się być ten sam.
- dla zmiany wartości rozpiętości wachlarza, na oko widzimy duże różnice dla różnych wartości parametru, natomiast nie wpływają one mocno na RMSE. Problem znowu polega na braku filtrowania, przez co obraz jest zbyt jasny i bez względu na to jak bardzo przypomina obraz wejściowy, wartość RMSE jest wysoka

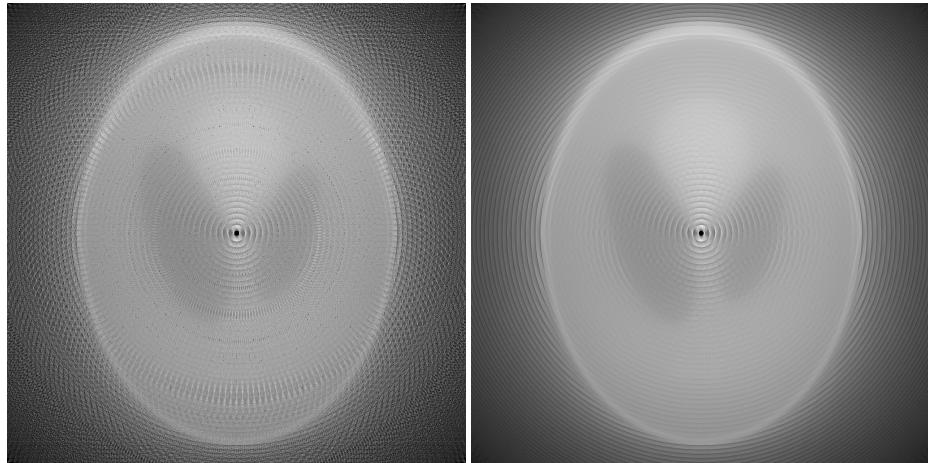
Na pierwszy rzut widać, że jakość obrazu z filtrowaniem jest dużo lepsza. Obraz lepiej oddaje czerń, niż w wersji bez filtra.



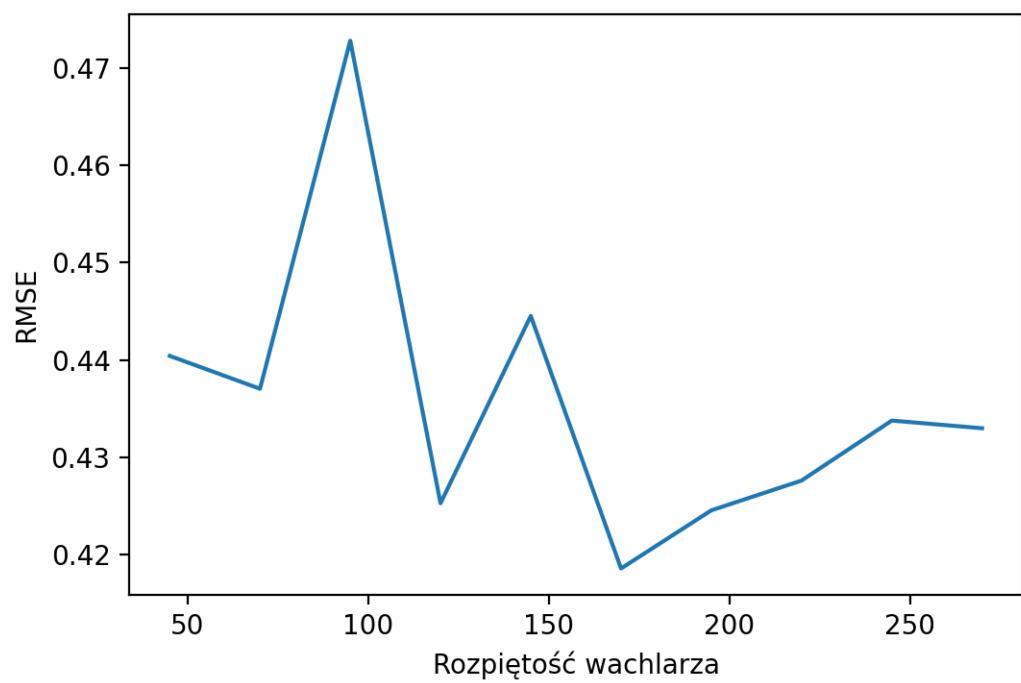


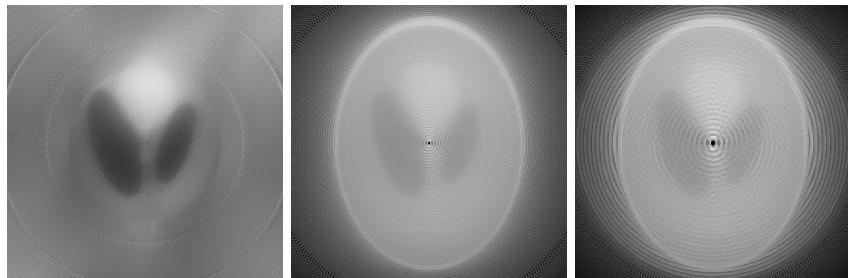
Rysunek 1: Wynik eksperymentu dla odpowiednio 90 i 720 detektorów



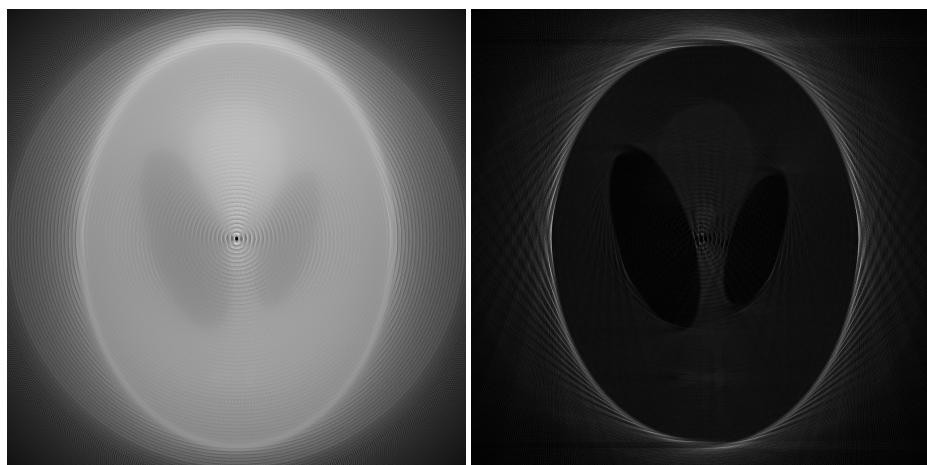


Rysunek 2: Wynik eksperymentu dla odpowiednio 90 i 720 skanów





Rysunek 3: Wynik eksperymentu dla wachlarza o rozpiętości odpowiednio 45, 150 i 270 stopni



Rysunek 4: Wynik eksperymentu dla odpowiednio wersji z filtrem i bez filtra