Sprawozdanie z projektu symulatora tomografu komputerowego

Jakub Bilski 155865 Szymon Tadych 156102

1 Wprowadzenie

Prezentowany projekt to dwuwymiarowa symulacja tomografii komputerowej, demonstrująca proces pozyskiwania i rekonstrukcji obrazów medycznych. Symulator realizuje:

- dyskretną transformację Radona (sinogram),
- odwrotną transformację Radona (backprojection),
- opcjonalne filtrowanie sinogramu (Ram–Lak),
- ocenę jakości RMSE,
- zapis w formacie DICOM z metadanymi pacjenta.

Projekt odzwierciedla istotne etapy pracy rzeczywistych systemów TK: od akwizycji danych aż po prezentację obrazu gotowego do analizy klinicznej. Dwuwymiarowe podejście upraszcza problem, ale zachowuje wszystkie kluczowe zależności fizyczne i numeryczne. Celem sprawozdania jest nie tylko przedstawienie kodu, lecz także uzasadnienie wyborów algorytmicznych, omówienie czynników wpływających na jakość rekonstrukcji oraz analiza empiryczna zależności pomiędzy parametrami skanowania a dokładnością odwzorowania.

2 Podstawy teoretyczne

2.1 Prawo Beera–Lamberta

Natężenie promieniowania po przejściu przez obiekt opisuje równanie:

$$I = I_0 \exp\Big(-\int_L \mu(s)\, ds\Big),$$

gdzie I_0 to natężenie początkowe, $\mu(s)$ współczynnik pochłaniania w punkcie s.

Równanie Beera–Lamberta implikuje, że każda warstwa materiału absorbuje część promieniowania zgodnie z własnością eksponencjalnego osłabienia. Współczynnik $\mu(s)$ zależy od rodzaju tkanki (gęstość, skład chemiczny) oraz długości fali promieniowania. Dokładność pomiaru osłabienia jest kluczowa, gdyż błąd w wyznaczeniu I/I_0 przekłada się bezpośrednio na artefakty w obrazie.

2.2 Transformata Radona

Transformata Radona $R_\theta(t)$ dla obrazu $f(x, y)$:

$$R_\theta(t) = \iint_{R^2} f(x, y)\, \delta(x \cos\theta + y \sin\theta - t)\, dx\, dy,$$

gdzie δ to delta Diraca, θ kąt projekcji, t przesunięcie od środka.

Dla każdego kąta θ sinogram gromadzi liniowe sumy wartości atenuacji na prostych przecinających obiekt. Interpretacja geometryczna: to projekcja obrazu widziana przez emiter i detektor ustawione pod kątem θ. Zbiór projekcji pod różnymi kątami stanowi komplet informacji niezbędnych do rekonstrukcji.

2.3 Filtrowanie sinogramu

Wykorzystanie prostego filtra Ram-Lak usuwającego rozmycie

$$h[0] = 1, \quad h[k] = 0 \quad \text{dla } k \text{ parzystych}, \quad h[k] = -\frac{4}{\pi^2\, k^2} \quad \text{dla } k \text{ nieparzystych}.$$

W praktyce wystarcza kernel o rozmiarze 21 elementów – uwaga na przesunięcie k, kernel symetryczny $\rightarrow$ wystarczy policzyć połowę maski. Filtr stosowany jest do poszczególnych wierszy sinogramu.

Filtrowanie w dziedzinie częstotliwości ma na celu skompensowanie naturalnego spadku wzmocnienia wyższych częstotliwości w procesie backprojection. Bez filtra otrzymujemy charakterystyczne rozmycie („starburst artifacts"). Filtr Ram–Lak wzmacnia składowe wysokoczęstotliwościowe, co skutkuje ostrzejszymi krawędziami rekonstrukcji, kosztem nieznacznego wzmocnienia szumów.

3 Implementacja głównych algorytmów

3.1 Model emitera i detektorów

Emiter i detektory są rozmieszczone po okręgu wokół obrazu. Funkcje poniżej generują współrzędne punktów w układzie dyskretnym:

```
def circle_coords(angle_shift, angle_range, count, radius=1,
    center=(0,0)):
    angles = np.linspace(0, angle_range, count) +
        angle_shift
    cx, cy = center
    x = radius * np.cos(angles) - cx
    y = radius * np.sin(angles) - cy
    points = np.array(list(zip(x, y)))
    return np.floor(points).astype(int)

def detector_coords(alpha, angle_range, count, radius=1,
    center=(0,0)):
```

```
    return circle_coords(
        np.radians(alpha - angle_range/2),
        np.radians(angle_range),
        count, radius, center
    )

def emitter_coords(alpha, angle_range, count, radius=1,
    center=(0,0)):
    return circle_coords(
        np.radians(alpha - angle_range/2 + 180),
        np.radians(angle_range),
        count, radius, center
    )[::-1]
```

Rzutowanie na siatkę pikseli odbywa się przez zaokrąglenie współrzędnych. Choć w rzeczywistych systemach stosuje się interpolację pod-pikselową, dyskretyzacja `floor()` + typ `int` jest wystarczająca dla dwuwymiarowej symulacji.

3.2 Algorytm Bresenhama

Bresenham służy do generowania punktów najbliższych linii prostej pomiędzy emiterem i detektorem. Dzięki prostym operacjom całkowitym jest to algorytm szybki i pozbawiony błędów zaokrągleń pływających:

```
def bresenham(x0, y0, x1, y1):
    if abs(y1-y0) > abs(x1-x0):
        swapped = True
        x0,y0,x1,y1 = y0,x0,y1,x1
    else:
        swapped = False
    m = (y1-y0)/(x1-x0) if x1!=x0 else 1
    q = y0 - m*x0
    if x0 < x1:
        xs = np.arange(np.floor(x0), np.ceil(x1)+1, dtype=
            int)
    else:
        xs = np.arange(np.ceil(x0), np.floor(x1)-1, -1,
            dtype=int)
    ys = np.round(m*xs + q).astype(int)
    if swapped:
        xs,ys = ys,xs
    return np.vstack((xs, ys))
```

Dyskretny przebieg linii umożliwia akumulację wartości pikseli na ścieżce promienia, co odpowiada całkowaniu $\int_L \mu(s)\,ds$.

3.3 Generowanie sinogramu

```
def radon(detector_count, angle_range, image, radius, center
    , alpha):
    emitters  = emitter_coords(alpha, angle_range,
        detector_count, radius, center)
    detectors = detector_coords(alpha, angle_range,
        detector_count, radius, center)
    lines     = draw_lines(emitters, detectors)
    raw       = np.array([np.sum(image[tuple(line)]) for
        line in lines])
    return rescale(raw)

def radon_all(image, scan_count, detector_count, angle_range
    ):
    image   = image_pad(image)
    center  = np.floor(np.array(image.shape)/2).astype(int)
    radius  = image.shape[0]//2
    alphas  = np.linspace(0, 180, scan_count)
    sino    = np.zeros((detector_count, scan_count))
    for i,a in enumerate(alphas):
        sino[:,i] = radon(detector_count, angle_range, image
            , radius, center, a)
    return sino
```

Sinogram to macierz o wymiarach (liczba detektorów) $\times$ (liczba skanów). Każda kolumna reprezentuje jedną projekcję obrazu przy danym kącie. Interpretacja wizualna sinogramu pozwala zdiagnozować brakujące kąty lub nieprawidłowe kalibracje – regularne wzory oznaczają prawidłowy przebieg skanowania.

3.4 Rekonstrukcja

```
def inverse_radon(image, count, sino_line, alpha,
                  detector_count, angle_range, radius,
                      center):
    emitters  = emitter_coords(alpha, angle_range,
        detector_count, radius, center)
    detectors = detector_coords(alpha, angle_range,
        detector_count, radius, center)
    lines     = draw_lines(emitters, detectors)
    for i,line in enumerate(lines):
        image[tuple(line)] += sino_line[i]
        count[tuple(line)] += 1

def inverse_radon_all(shape, sinogram, angle_range,
                      use_filter=False, original=None, log=
                          None):
    if use_filter:
        sinogram = apply_filter_to_sinogram(sinogram)
```

```
    sino      = np.swapaxes(sinogram,0,1)
    result    = image_pad(np.zeros(shape))
    count     = np.zeros(result.shape)
    center    = np.floor(np.array(result.shape)/2).astype(
        int)
    radius    = result.shape[0]//2
    alphas    = np.linspace(0, 180, sino.shape[0])
    for i,a in enumerate(alphas):
        inverse_radon(result, count, sino[i], a,
                      sino.shape[1], angle_range, radius,
                          center)
        if log is not None and original is not None:
            temp = rescale(result/np.maximum(count,1))
            log.append(calculate_rmse(original, unpad(temp,*
                shape)))
    count[count==0] = 1
    recon = rescale(result/count)
    return unpad(recon, *shape)
```

Proces rekonstrukcji opiera się na backprojection: każda skalarna wartość sinogramowa jest rzutowana z powrotem na obraz i sumowana. Wzmocnienie wartości wysokoczęstotliwościowych przez filtr pozwala uzyskać wyraźniejsze krawędzie, natomiast pominięcie filtra daje bardziej rozmyte, ale mniej zaszumione wyniki.

3.5 Filtrowanie sinogramu

```
def create_filter_kernel(size=21):
    assert size%2==1
    k = np.arange(-(size//2), size//2+1)
    kernel = np.zeros_like(k, dtype=np.float32)
    for i,val in enumerate(k):
        if val==0:      kernel[i]=1
        elif val%2==0:  kernel[i]=0
        else:           kernel[i] = -4/(np.pi**2*val**2)
    return kernel

def apply_filter_to_sinogram(sino, kernel_size=21):
    kernel = create_filter_kernel(kernel_size)
    out    = np.zeros_like(sino)
    for i in range(sino.shape[0]):
        out[i] = np.convolve(sino[i], kernel, mode='same')
    return out
```

Wielkość jądra wpływa na kompromis między rozdzielczością a artefaktami brzegowymi. Mniejszy wpływ objawia się niedostatecznym wzmocnieniem wyższych częstotliwości, większy może zwiększać szum. Dla obrazów medycznych zazwyczaj wybiera się 21–31 próbek.

3.6 Ocena jakości (RMSE)

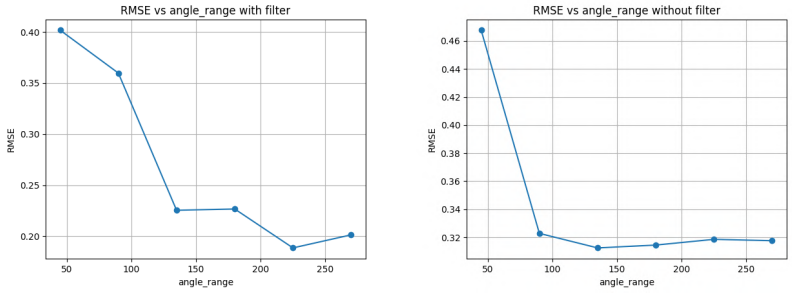
```
def calculate_rmse(orig, rec):
    o = orig/np.max(orig)
    r = rec/np.max(rec)
    return np.sqrt(np.mean((o-r)**2))
```

Metryka RMSE jest czuła na różnice intensywności i pozwala ocenić globalną zgodność rekonstrukcji z oryginałem. Normalizacja do $[0, 1]$ zapewnia porównywalność wyników pomiędzy różnymi obrazami i parametrami.

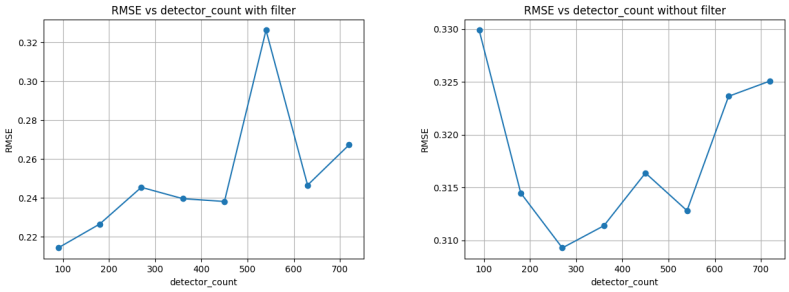
4 Analiza eksperymentów

Analiza przeprowadzona została na obrazie testowym w skali szarości o wymiarach (512×512). Badano wpływ trzech parametrów: liczby detektorów, liczby kątów skanowania oraz rozpiętości kątowej skanu, zarówno z filtrem Ram–Lak, jak i bez niego. Wyniki przedstawiono w formie wykresów RMSE oraz wizualizacji rekonstrukcji.

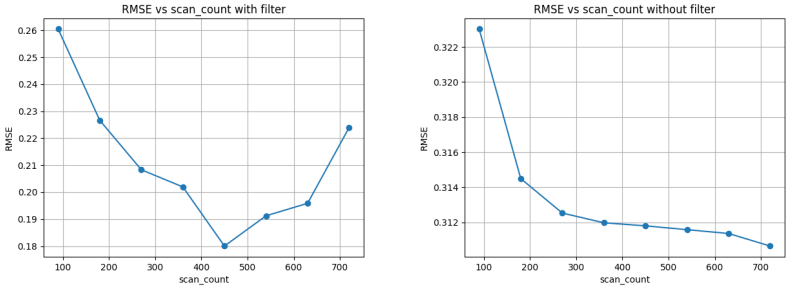
5 Wyniki eksperymentów (RMSE)



Rysunek 1: RMSE vs rozpiętość kątowa: z filtrem (lewy) i bez filtra (prawy). Obserwacje: dla małych kątów ($< 90°$) RMSE jest znacząco wyższe ze względu na brak informacji z pełnego zakresu, filtr poprawia wyniki aż do ok. $180°$, powyżej efekt jest minimalny.



Rysunek 2: RMSE vs liczba detektorów: z filtrem (lewy) i bez filtra (prawy). Obserwacje: zwiększenie liczby detektorów redukuje RMSE niemal wykładniczo do ok. 360, dalej korzyść jest marginalna. Filtr wprowadza dodatkową poprawę, szczególnie przy niższej liczbie detektorów.

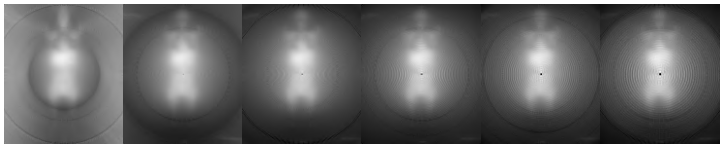


Rysunek 3: RMSE vs liczba skanów: z filtrem (lewy) i bez filtra (prawy). Obserwacje: podobny trend jak dla detektorów – wzrost liczby kątów skanowania znacząco poprawia jakość rekonstrukcji do ok. 360 kątów.

6 Zmiana wyglądu rekonstrukcji przy zmianie parametrów

Poniżej przedstawiono, jak zmienia się wizualny efekt rekonstrukcji wraz ze zmianą parametrów. W każdym podpunkcie pierwsza rząd zdjęć to wersja *bez filtra*, drugi rząd to wersja *z filtrem*. Drobne artefakty promieniowe widoczne przy niższej liczbie kątów lub detektorów wygładzają się po zastosowaniu filtra.

6.1 Rozpiętość kątowa



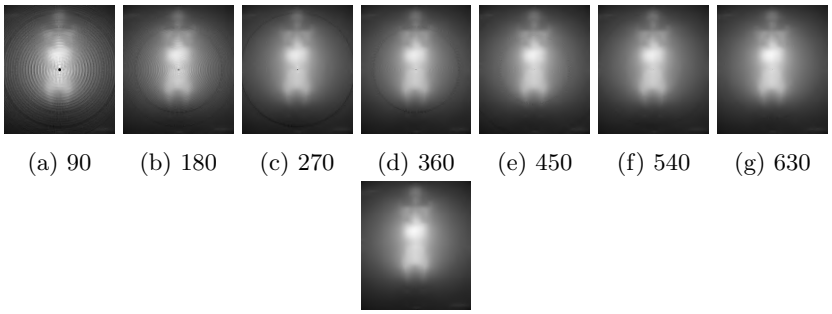
(a) 45° (b) 90° (c) 135° (d) 180° (e) 225° (f) 270°

Rysunek 4: Rekonstrukcja bez filtra dla różnych rozpiętości kątowych

(a) 45° (b) 90° (c) 135° (d) 180° (e) 225° (f) 270°

Rysunek 5: Rekonstrukcja z filtrem dla różnych rozpiętości kątowych

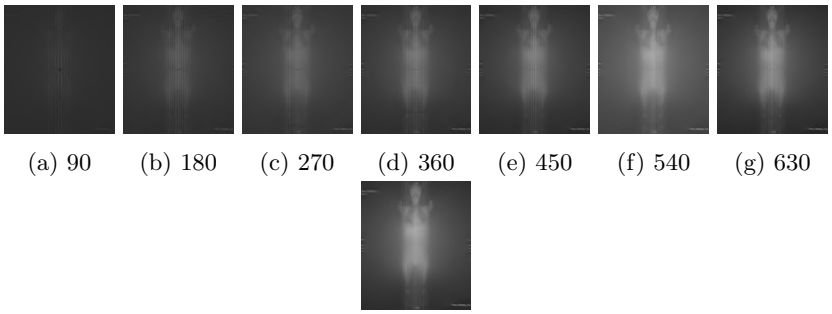
6.2 Liczba detektorów



(a) 90 (b) 180 (c) 270 (d) 360 (e) 450 (f) 540 (g) 630

(h) 720

Rysunek 6: Rekonstrukcja bez filtra dla różnych liczb detektorów

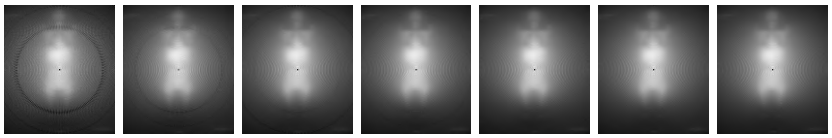


(a) 90 (b) 180 (c) 270 (d) 360 (e) 450 (f) 540 (g) 630

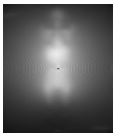
(h) 720

Rysunek 7: Rekonstrukcja z filtrem dla różnych liczb detektorów

6.3 Liczba skanów

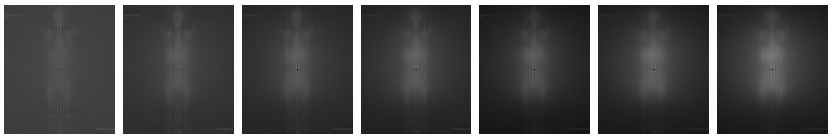


(a) 90 (b) 180 (c) 270 (d) 360 (e) 450 (f) 540 (g) 630



(h) 720

Rysunek 8: Rekonstrukcja bez filtra dla różnych liczby kątów



(a) 90 (b) 180 (c) 270 (d) 360 (e) 450 (f) 540 (g) 630

(h) 720

Rysunek 9: Rekonstrukcja z filtrem dla różnych liczby kątów

7 Obsługa formatu DICOM

```
import pydicom
from pydicom.dataset import Dataset, FileDataset
import datetime
import numpy as np

def save_as_dicom(image, filename, patient_info):
    file_meta = Dataset()
    file_meta.MediaStorageSOPClassUID    = '
        1.2.840.10008.5.1.4.1.1.2'
    file_meta.MediaStorageSOPInstanceUID= pydicom.uid.
        generate_uid()
```

```
    file_meta.TransferSyntaxUID         = pydicom.uid.
        ImplicitVRLittleEndian

    ds = FileDataset(filename, {}, file_meta=file_meta,
        preamble=b"\0"*128)
    ds.PatientName       = patient_info['name']
    ds.PatientID         = patient_info['id']
    ds.PatientBirthDate  = patient_info['birthdate']
    ds.StudyDate         = datetime.datetime.now().strftime(
        '%Y%m%d')
    ds.StudyTime         = datetime.datetime.now().strftime(
        '%H%M%S')
    ds.StudyDescription  = patient_info.get('description','')
        )

    ds.SamplesPerPixel   = 1
    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.Rows, ds.Columns  = image.shape
    ds.BitsAllocated     = 8
    ds.BitsStored        = 8
    ds.HighBit           = 7
    ds.PixelRepresentation=0
    ds.PixelData         = image.astype(np.uint8).tobytes()

    ds.save_as(filename)
    return ds
```

Funkcja tworzy zgodny ze standardem DICOM plik zawierający metadane pacjenta i parametry badania. Standard DICOM zapewnia interoperacyjność z dowolnym oprogramowaniem medycznym, a metadane ułatwiają archiwizację i wyszukiwanie badań.