Sprawozdanie z laboratorium: Informatyka w Medycynie

Projekt 1: Tomograf komputerowy

30 marca 2021

Prowadzący: mr inż. Magdalena Martyn

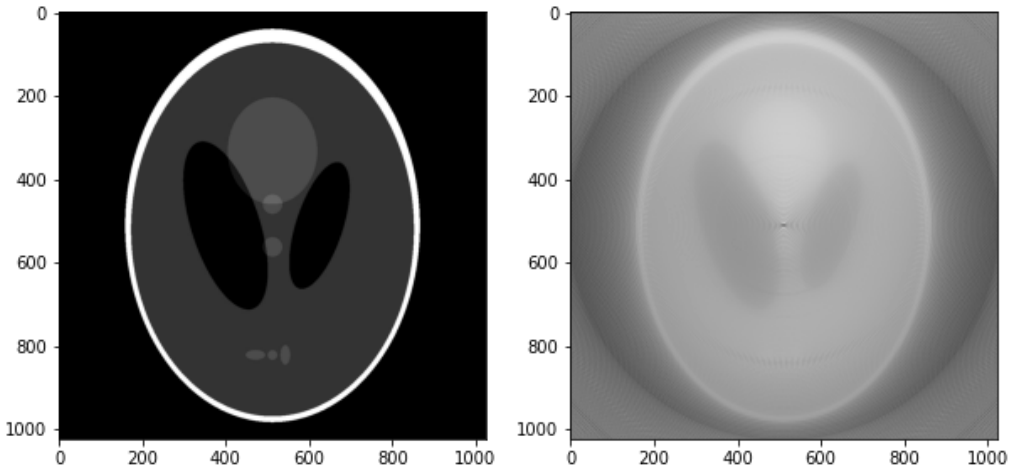
Autorzy: **Jakub Frieske** 141214

Zajęcia wtorkowe, 15:10.

Oświadczam/y, że niniejsze sprawozdanie zostało przygotowane wyłącznie przez powyższych autora/ów, a wszystkie elementy pochodzące z innych źródeł zostały odpowiednio zaznaczone i są cytowane w bibliografii.

Wstęp

Celem projektu jest przygotowanie implementacji aplikacji symulującej działanie tomografu komputerowego z wykorzystaniem transformaty Radona oraz zbadanie jakości odtwarzanego zdjęcia. Na obrazie wejściowy dokonujemy transformaty Radona, w ten sposób otrzymujemy sinogram, na którym wykonujemy odwrotną transformatę, aby otrzymać obraz wyjściowy.



Rysunek 1: Przykładowy efekt działania aplikacji.

1 Model tomografu

W swojej aplikacji zastosowałem model równoległy, ponieważ jest wygodny w implementacji. Po wyznaczaniu współrzędnych punktów odpowiadających emiterom, wyznaczenie współrzędnych odpowiadających detektorom przebiega analogicznie.

2 Oprogramowanie

Wybrałem język programowania Python. Ze względu na jego prostotę i szeroką bazę dodatkowych bibliotek.

Istotne biblioteki i ich wykorzystanie:

- IPython: budowa interfejsu
- Numpy: obliczenia i operacje na n-wymiarowych macierzach
- Matplotlib: wizualizacje obrazów
- cv2: odczyt obrazów
- Pydicom: zapis i odczyt obrazów w standardzie DICOM

3 Opis działania

Symulator tomografu składa się z n detektorów i n emiterów. Rozstawione są one w równych odległościach na dwóch równoległych prostych. Zarówno liczba n jak i krok $\Delta\alpha$, o jaki będziemy obracać układ emiterów/detektorów oraz rozpiętość kątowa l, odległość między pierwszym i ostatnim emiterem, są konfigurowalne.

3.1 pozyskiwanie odczytów dla poszczególnych detektorów

```
def gen_sinogram(n,deltaAlpha,l,img, iter):
    height, width  = img.shape
    cx, cy = height//2, width //2
    iterations = round(360/deltaAlpha)

    em, det = em_det_parallel(n, l, img.shape)
    lines = [bresenham_line(p0,p1) for p0, p1 in zip(em,det)]
    all_lines = []
    sinogram = np.zeros( (n, iterations) )
    all_sinogram = []

    for i in range(iterations):
        rotated_lines = []
        for line in lines:
            rot_line = rotate_line(line, ( cx, cy ), math.pi/180 * deltaAlpha
                                                    * i )
            new_line = [(px,py) for px,py in rot_line if 0<px<height and 0<py<
                                                    width ]
            rotated_lines.append(new_line)

        all_lines.append(rotated_lines)
        for j, line in enumerate(rotated_lines):
            sinogram[j][i] = np.sum([img[tuple(point)] for point in line])

        if (i+1) % iter == 0:
          all_sinogram.append( sinogram/ np.max(sinogram) )

    sinogram = sinogram / np.max(sinogram)
    return all_sinogram, sinogram, all_lines
```

Code Listing 1: Kod generujący sinogram.

Na początku wyznaczamy pozycje emiterów/detektorów za pomocą funkcji em_det_parallel, której podajemy liczbę n emiterów/detektorów, rozpiętość kątową l i rozmiar obrazu. Dla kolejnej iteracji, która zależy od kroku $\Delta\alpha$:

1. Obracamy wszystkie linie o odpowiedni kąt przy pomocy funkcji rotate_line oraz sprawdzamy czy nasz punkt nie wychodzi poza obszar obrazu.

2. Każdemu pikselowi sinogramu przypisujemy sumę wartości pikseli po przejściu po linii między emiterem a detektorem.

3. Następnie normalizujemy otrzymany sinogram dzieląc wartość każdego piksela przez maksymalną wartość w wynikowej tablicy.

Na wyjściu otrzymujemy sinogramy co „n” iteracje, sinogram, a także wszystkie linie, z których stworzyliśmy sinogram.

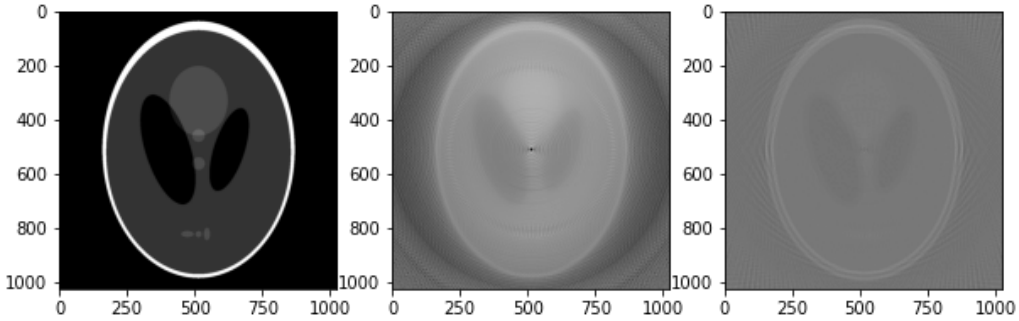
3.2 Filtrowanie sinogramu

Do filtracji sinogramu użyłem wbudowanej funkcji np.convole, z maską zależną od liczby detektorów.

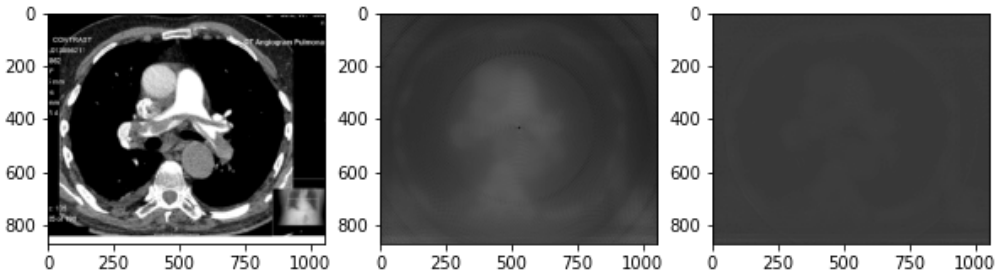
```
def filter_sinogram(sinogram):
    filtered = np.zeros(sinogram.shape)
    for i in range(len(sinogram[0])):
        filtered[:,i] = np.convolve(sinogram[:,i],
                                    gen_kernel( len(sinogram[:,i]) ), mode = 'same')
    return filtered
```

Code Listing 2: Kod filtrujący sinogram.

Poniżej są przedstawione wyniki wpływu filtrowania sinogramu na jakość obrazu, dla liczby detektorów równej 100, kącie $\Delta\alpha$ równej 2 i rozpiętości kątowej równej 90.



Rysunek 2: Wizualizacja efektów filtrowania sinogramu (shepp-logan). Po lewej stronie oryginalny obrazek, na środku efekt odwrotnej transformaty Radona bez filtrowania, a po prawej z filtrowaniem



Rysunek 3: Wizualizacja efektów filtrowania sinogramu (saddle-pe)

Tabela 1: Wyniki eksperymentu

obrazek	rysunek	RMSE (bez filtrowania)	RMSE (z filtrowaniem)
shepp-logan	rysunek 2	0.4107	0.2323 (↓)
saddle-pe	rysunek 3	0.3137	0.4223 (↑)

Na podstawie rysunków można wyciągnąć wnioski, że filtrowanie zdecydowanie poprawia odzwierciedlenie obrazu, w szczególności jasność elementów na obrazie jest bardziej zbliżona do oryginału. Ze względu na niewielką ilość emiterów wynik jest słabo widoczny.

3.3 Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

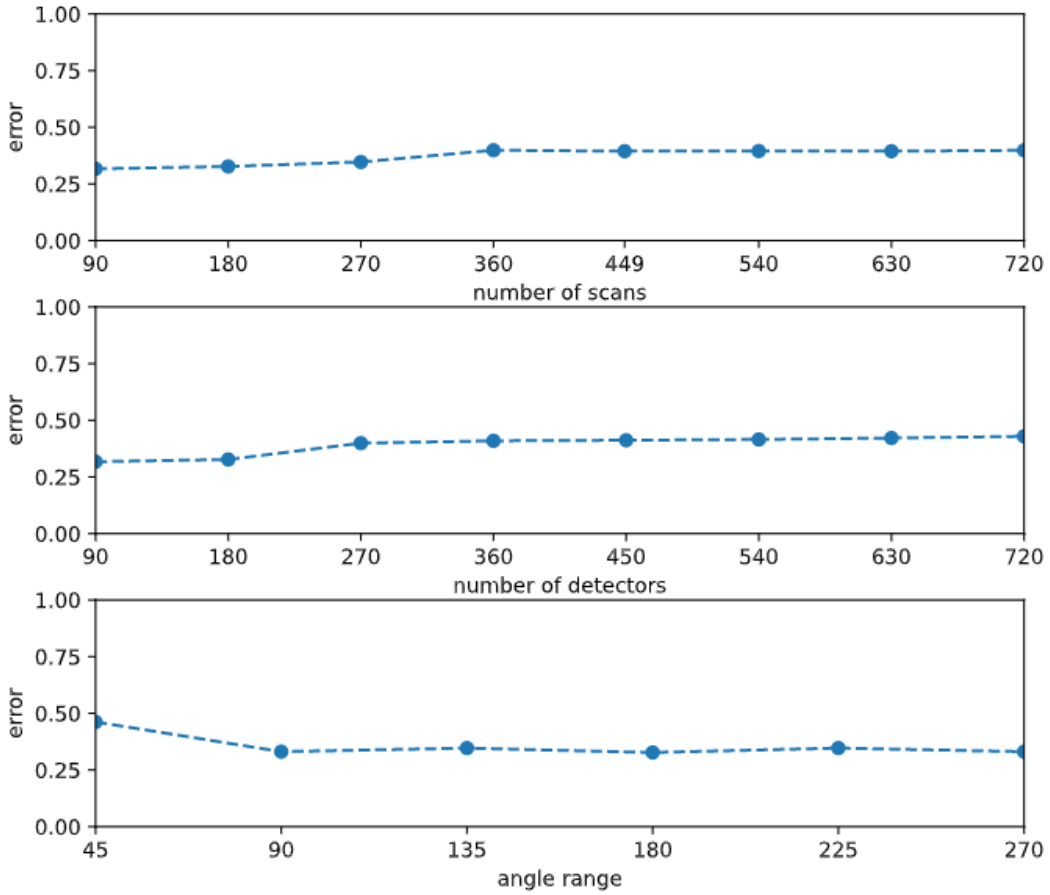
```
def reconstruct_image(sinogram, all_lines, image_size, iter):
    h, w = image_size
    rec_img = np.zeros((h,w))
    count_matrix = np.zeros((h,w))
    all_rec = []

    for x, alpha_line in enumerate(all_lines):
      for y, line in enumerate(alpha_line):
        for px, py in line:
            rec_img[px][py] += sinogram[y][x]
            count_matrix[px][py] += 1
      if (x+1) % iter == 0:
        all_rec.append( rec_img / np.max(rec_img) )

    for i in range(len(count_matrix)):
      for j in range(len(count_matrix[0])):
          if count_matrix[i][j] != 0:
              rec_img[i][j] /= count_matrix[i][j]

    return all_rec, rec_img
```

Code Listing 3: Kod generujący obraz wyjściowy.

Operacja odwrotnej transformaty polega na iteracji po wygenerowanych liniach z funkcji gen_sinogram. Dla każdego punktu z tych linii zwiększamy wartość piksela odtwarzanego obrazu o wartość na odpowiednim pikselu sinogramu. Liczba linii przechodzących przez każdy punkt może być różna, dlatego w celu normalizacji wartość piksela dzielimy przez liczbę linii przez niego przechodzącą.

3.4 Wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego

Na rysunku 4 przedstawiony jest błąd przy zmieniającej się odpowiednio liczbie skanów, detektorów i rozwartości kątowej (wynikowe obrazki widoczne są pod sekcją Eksperyment). Z rysunku 4 można zauważyć, że liczba skanów i detektorów na początku trochę pogarsza jakość obrazu, lecz później już się stabilizuje. Wydaje mi się, że jest to związane ze sposobem normalizacji. Jednak na wykresach RMSE nie widać znaczących zmian, może to być spowodowane, że w obrazach wyjściowych jest duża ilość szumu.



Rysunek 4: Wartość RMSE przy zmienającej się liczbie skanów, detektorów i rozwartości kątowej

3.5 Odczyt i zapis plików DICOM

Do obsługi plików DICOM wykorzystałem bibliotekę pydicom. W programie odczyt jest realizowany przez funkcję read_dicom.

```
def read_dicom(file,all=False):
    try:
      dataset = pydicom.dcmread(file)
      img = dataset.pixel_array
      if all:
        plt.imshow(img, cmap='gray')
        print(dataset)
      else:
        try:
          print(f"Patient's Name...: {dataset.PatientName}")
          print(f"Patient ID.......: {dataset.PatientID}")
          print(f"Patient's Sex....: {dataset.PatientSex}")
          print(f"Patient's Birth..: {dataset.PatientBirthDate}")
          print(f"Study Description: {dataset.StudyDescription}")
          print(f"Study Date.......: {dataset.StudyDate}")
        except:
          pass

    except:
      print("Lack of information")
```

Code Listing 4: Kod do odczytu DICOM'u.

Pliki DICOM zapisuje przy użyciu funkcji save_dicom, która przyjmuje ścieżkę pod którą będzie zapisany plik, imię, id pacjenta, datę badania, komentarz lekarza, zdjęcie z badania, płeć i datę urodzin pacjenta.

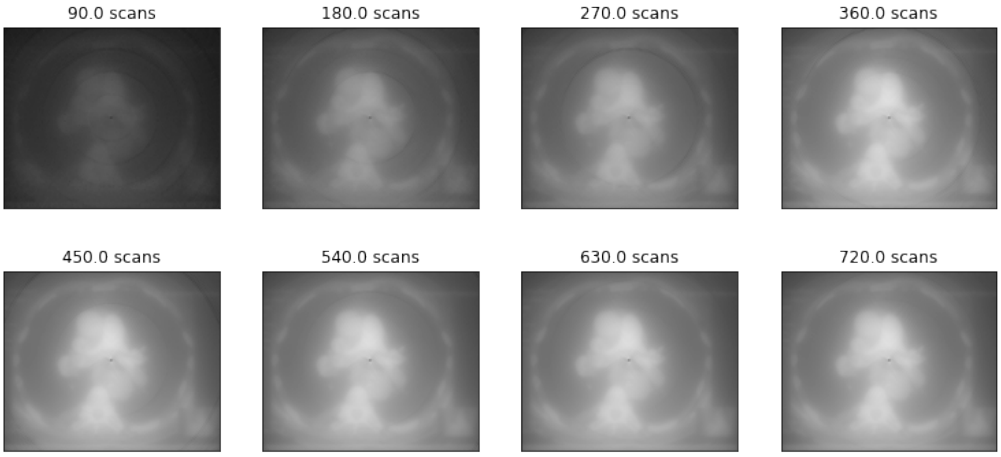
```
def write_to_dicom(arg):
    save_dicom(patient_filename.value, patient_name.value, patient_id.value,
               patient_studydate.value.strftime('%Y%m%d'),
               patient_comment.value, save_image,
               patient_sex.value, patient_birth.value)
    print("File saved.")
```

Code Listing 5: Kod do zapisu DICOM'u.

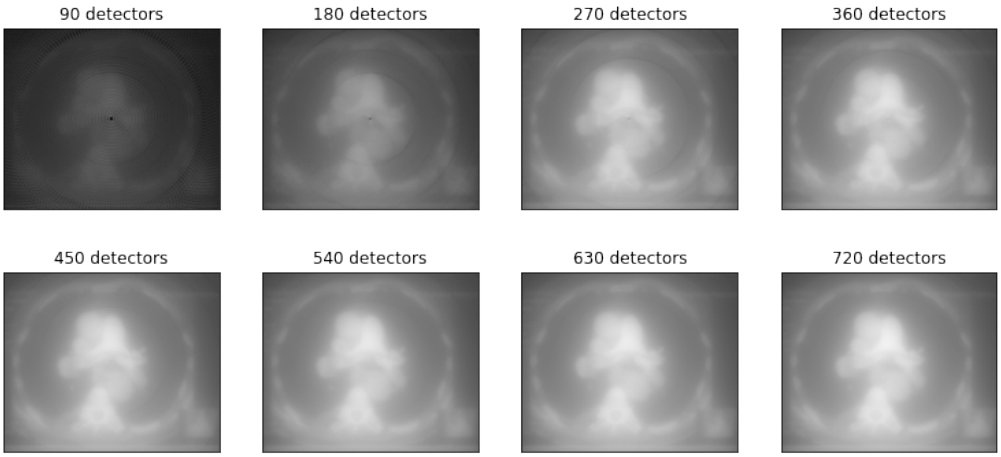
Poprawność zapisu plików DICOM zostały zweryfikowane przy pomocy stron:
`https://www.imaios.com/en/Imaios-Dicom-Viewer`
`https://www.fviewer.com/pl/view-dicom`

4 Eksperyment

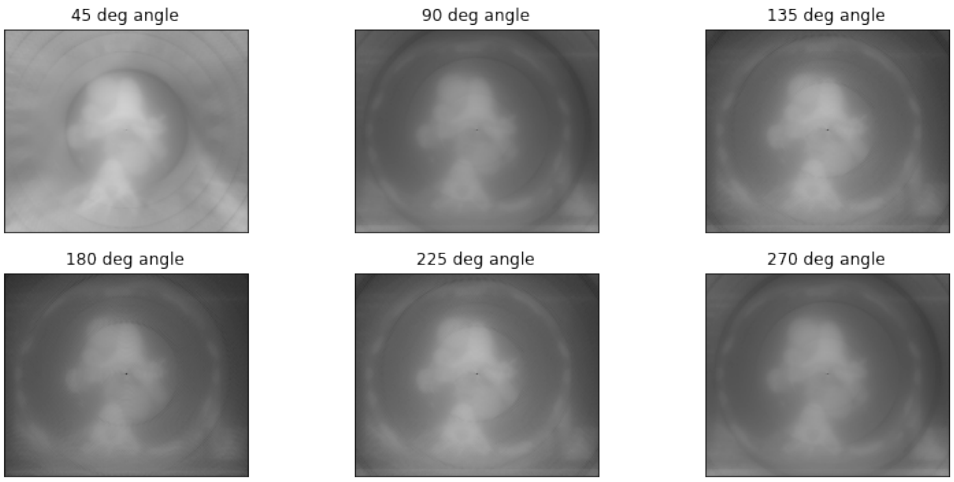
W eksperymencie sprawdzałem wpływ poszczególnych parametrów transformaty na jakość odtworzonego obrazu. Jako miary użyłem błędu średniokwadratowego(RMSE). Zgodnie z poleceniem, jako parametry domyślne przyjąłem wartość 180 dla detektorów, skanów oraz rozpiętości wachlarza. Do testów użyłem obrazu „saddle-pe", widoczny na rysunku 3.



Rysunek 5: Zmiana liczby skanów



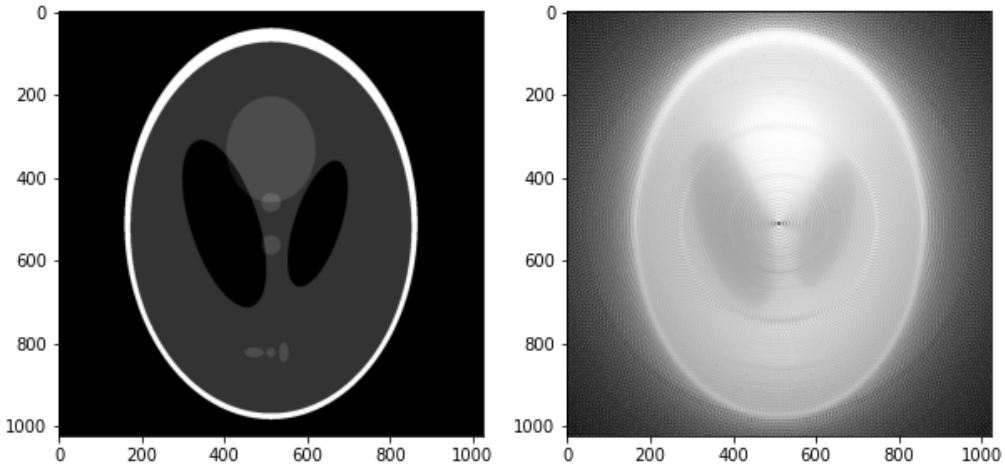
Rysunek 6: Zmiana liczby detektorów



Rysunek 7: Zmiana rozwartości kąta

5 Podsumowanie

Otrzymane wyniki są zadowalające. Ewentualnie można było poprawić normalizacje dla lepszego efektu (Przykład poniżej).



Rysunek 8: Inny sposób normalizacji