

Symulator tomografu komputerowego

Marcin Pastwa 136779
Piotr Tomaszewski 136821

1 Zastosowany model tomografu

Aplikacja symuluje działanie tomografu komputerowego o równoległym modelu układu emiter/detektor.

2 Zastosowany język programowania oraz dodatkowe biblioteki

Aplikacja została napisana w języku Python, wykorzystane zostały następujące biblioteki:

numpy Obliczenia macierzowe i wektorowe.

scikit-image Odczyt i zapis plików graficznych.

pydicom Obsługa plików DICOM.

tkinter Interfejs użytkownika.

pillow Wyświetlanie obrazów w interfejsie użytkownika.

3 Opis głównych funkcji programu

3.1 Pozyskiwanie odczytów dla poszczególnych detektorów

Generowanie sinogramu odbywa się iteracyjnie. W każdej iteracji zmienia się aktualna wartość kąta α , zaczynając od wartości 0° , kończąc na wartości mniejszej od 180° , z krokiem definiowanym przez parametr $\Delta\alpha$. Każda iteracja zaczyna się od wyznaczenia aktualnego położenia emiterów i detektorów. Wartości te są zapisywane, gdyż aplikacja oferuje wizualizację aktualnego stanu skanera.

```
1 def radon_transform_step(self):
2     # Wyznaczenie aktualnego położenia emiterów i detektorów
3     self.current_emitters, self.current_detectors = \
4         self._calculate_emitters_detectors_position()
5     # Wyznaczenie linii łączących odpowiadające emitery i detektory
6     self.current_scan_lines = \
7         self._calculate_scan_lines(self.current_emitters, self.current_detectors)
8     self.scan_lines.append(self.current_scan_lines)
9     # Wyznaczenie całki po każdej z linii skanera
10    for line_id, line in enumerate(self.current_scan_lines):
11        line_integral = np.sum([self.input_image[point] for point
12                                in self.current_scan_lines[line_id]])
13        self.radon_result[line_id, self.current_radon_iteration] += line_integral
14    self.current_alpha += self.delta_alpha
15    self.current_radon_iteration += 1
```

Listing 1: Krok transformaty Radona

```

1 def _calculate_emitters_detectors_position(self):
2     emitters = []
3     detectors = []
4     # Promień obrotu jest połową średnicy okręgu opisanego na obrazie wejściowym
5     r = self.circumcircle_diameter // 2
6     # Poniżej wyznaczone są kąty  $\alpha'$  dla poszczególnych emiterów i detektorów. Gdy wartość kąta  $\alpha'$  emitery
7     # zmienimy o pewną wartość  $x$ , a detektora o wartość  $-x$ , uzyskamy efekt przesunięcia łączącej je linii względem
8     # środka okręgu, jednocześnie nie zmieniając nachylenia tej linii
9     angles = np.linspace(start=self.current_alpha - self.em_det_spread / 2,
10                          stop=self.current_alpha + self.em_det_spread / 2, num=self.em_det_no)
11     angles_rad = [math.radians(x) for x in angles]
12     for i in range(self.em_det_no):
13         angle_rad = angles_rad[i]
14         emitter_a = int(self.input_image_center + r * math.cos(angle_rad))
15         emitter_b = int(self.input_image_center - r * math.sin(angle_rad))
16         emitters.append((emitter_a, emitter_b))
17     # Detektory są umieszczane w odwrotnej kolejności. Dzięki temu wszystkie linie łączące odpowiadające sobie
18     # emitery i detektory będą (w danej iteracji) tak samo nachylone
19     for i in range(self.em_det_no - 1, -1, -1):
20         angle_rad = angles_rad[i]
21         detector_a = int(self.input_image_center - r * math.cos(angle_rad))
22         detector_b = int(self.input_image_center + r * math.sin(angle_rad))
23         detectors.append((detector_a, detector_b))
24     return emitters, detectors

```

Listing 2: Wyznaczanie położenia emiterów i detektorów

```

1 def _bresenham(x1, y1, x2, y2):
2     delta_x = x2 - x1
3     delta_y = y2 - y1
4     j = y1
5     error = delta_y - delta_x
6     points = []
7     for i in range(x1, x2 + 1):
8         points.append((i, j))
9         if error >= 0:
10             j += 1
11             error -= delta_x
12         error += delta_y
13     return points
14
15 # Poniższa funkcja ma na celu obejście ograniczeń algorytmu Bresenham, umożliwiając generowanie dowolnych
16 # linii
17 def generate_line(x1, y1, x2, y2):
18     delta_y = abs(y1 - y2)
19     delta_x = abs(x1 - x2)
20     if delta_y > delta_x:
21         points = generate_line(y1, x1, y2, x2)
22         for i in range(len(points)):
23             points[i] = (points[i][1], points[i][0])
24     elif y2 < y1 and x2 < x1:
25         points = _bresenham(x1, y1, x1 + delta_x, y1 + delta_y)
26         for i in range(len(points)):
27             points[i] = (2 * x1 - points[i][0], 2 * y1 - points[i][1])
28     elif y2 < y1:
29         points = _bresenham(x1, y1, x2, y1 + delta_y)
30         for i in range(len(points)):
31             points[i] = (points[i][0], 2 * y1 - points[i][1])
32     elif x2 < x1:
33         points = _bresenham(x1, y1, x1 + delta_x, y2)
34         for i in range(len(points)):
35             points[i] = (2 * x1 - points[i][0], points[i][1])
36     else:
37         points = _bresenham(x1, y1, x2, y2)
38     return points

```

Listing 3: Algorytm Bresenham

3.2 Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

Odtwarzanie obrazu wygląda bardzo podobnie do generowania sinogramu. Jednak, zamiast wyznaczać całkę po kolejnych liniach skanujących, dodajemy jej wartość do każdego piksela obrazu wynikowego, który do tej linii należy.

```
1 def iradon_step(self):
2     for s, line in enumerate(self.scan_lines[self.current_iradon_iteration]):
3         for point in line:
4             self.iradon_result[point] += self.radon_result[s, self.current_iradon_iteration]
5     self.current_iradon_iteration += 1
```

Listing 4: Krok odtwarzania obrazu

Następnie, wynikowy obraz zostaje poddany normalizacji. Przyjęliśmy, że pracujemy z obrazami 8-bitowymi.

```
1 def visualize_reconstructed_img(self):
2     # Usunięcie poprzedniego obrazu, jeśli taki istnieje
3     self.rec_img = np.zeros((self.iradon_result.shape[0], self.iradon_result.shape[1]), dtype=
4     np.uint8)
5     max_val = np.max(self.iradon_result)
6     min_val = np.min(self.iradon_result)
7     for i in range(len(self.iradon_result)):
8         for j in range(len(self.iradon_result[i])):
9             self.rec_img[i, j] = \
10                int((self.iradon_result[i, j] - min_val) / (max_val - min_val) * 255)
11     return self.rec_img
```

Listing 5: Normalizacja odtworzonego obrazu

3.3 Odczyt i zapis plików DICOM

Aplikacja wspiera obsługę plików DICOM. Mimo, że pliki te, z zasady, przechowują wynik badania tomografem, tj. już odtworzony obraz, uznaliśmy że może on posłużyć za dane wejściowe naszego symulatora.

Program umożliwia odczyt i zapis podstawowych informacji o badaniu i pacjencie, komentarza i obrazu w formacie DICOM.

```
1 def dicom_load(path):
2     # zbiór danych DICOM
3     ds = pydicom.dcmread(path)
4     image = ds.pixel_array
5     # Przyjęliśmy, że pracujemy z obrazami 8-bitowymi
6     if image.dtype != np.uint8:
7         img_max = np.max(image)
8         img_min = np.min(image)
9         image = 255 * (image - img_min) / (img_max - img_min)
10    return ds, image.astype(np.uint8)
```

Listing 6: Wczytywanie plików DICOM

Aby ułatwić dalszą pracę i zwiększyć czytelność danych pozyskanych z pliku DICOM, interesujące nas pola zostają odpowiednio sformatowane i umieszczone w słowniku `data`.

Zbiór danych `dataset` jest obiektem dostarczonym przez bibliotekę `pydicom`, który przechowuje w swojej strukturze poszczególne pola zdefiniowane w standardzie DICOM.

```
1 def dicom_read_dataset(dataset):
2     data = {}
3     if dataset.get('StudyDate'):
4         data['StudyDate'] = dicom_date_dataset_to_display(dataset.get('StudyDate'))
5     else:
6         data['StudyDate'] = ''
7     if dataset.get('StudyTime'):
8         data['StudyTime'] = dicom_time_dataset_to_display(dataset.get('StudyTime'))
9     else:
10        data['StudyTime'] = ''
11    data['PatientID'] = str(dataset.get('PatientID') or '')
12    if dataset.get('PatientName'):
13        patient_name = dataset.get('PatientName')
14        try:
15            data['PatientGivenName'] = patient_name.given_name
16        except AttributeError:
17            data['PatientGivenName'] = ''
18        try:
19            data['PatientFamilyName'] = patient_name.family_name
```

```

20         except AttributeError:
21             data['PatientFamilyName'] = ''
22     else:
23         data['PatientGivenName'] = ''
24         data['PatientFamilyName'] = ''
25     sex = dataset.get('PatientSex')
26     if sex:
27         if sex == 'F':
28             data['PatientSex'] = 'Kobieta'
29         elif sex == 'M':
30             data['PatientSex'] = 'Mezyczna'
31     else:
32         data['PatientSex'] = 'Nieznana'
33     bday = dataset.get('PatientBirthDate')
34     if bday:
35         data['PatientBirthDate'] = dicom_date_dataset_to_display(bday)
36     else:
37         data['PatientBirthDate'] = ''
38     data['ImageComments'] = dataset.get('ImageComments') or ''
39     return data

```

Listing 7: Odczyt interesujących wartości ze zbioru danych DICOM

Jeżeli użytkownik otworzy obraz wejściowy, który nie jest plikiem DICOM, zostaje utworzony i zainicjalizowany nowy zbiór danych, który jest następnie umieszczany w pliku tymczasowym.

```

1 def dicom_create_new_dataset():
2     suffix = '.dcm'
3     file_name = tempfile.NamedTemporaryFile(suffix=suffix).name
4     file_meta = Dataset()
5     # Ustawienie niektórych wymaganych wartości (pozostałe ustawiane przy zapisie)
6     file_meta.MediaStorageSOPClassUID = '1.2.840.10008.5.1.4.1.1.2'
7     file_meta.MediaStorageSOPInstanceUID = "1.3.6.1.4.1.5962.1.1.1.1.20040119072730.12322"
8     file_meta.ImplementationClassUID = "1.3.6.1.4.1.5962.2"
9     file_meta.TransferSyntaxUID = '1.2.840.10008.1.2'
10    return FileDataset(file_name, {}, file_meta=file_meta, preamble=b"\0" * 128)

```

Listing 8: Tworzenie nowego pliku DICOM

Po zatwierdzeniu przez użytkownika wprowadzenia danych o badaniu i pacjencie oraz komentarza, zostają one sformatowane tak, aby spełniać warunki standardu DICOM, a następnie są zapisywane do przechowywanego w pamięci zbioru danych.

```

1 def dicom_store_data(data, dataset):
2     if data.get('StudyDate'):
3         dataset.StudyDate = dicom_date_display_to_dataset(data.get('StudyDate'))
4     if data.get('StudyTime'):
5         dataset.StudyTime = dicom_time_display_to_dataset(data.get('StudyTime'))
6     if data.get('PatientGivenName') or data.get('PatientFamilyName'):
7         patient_given_name = data.get('PatientGivenName') or ''
8         patient_family_name = data.get('PatientFamilyName') or ''
9         dataset.PatientName = '~'.join((patient_family_name, patient_given_name))
10    if data.get('PatientID'):
11        dataset.PatientID = data.get('PatientID')
12    if data.get('PatientSex'):
13        if data.get('PatientSex') == 'Mezyczna':
14            dataset.PatientSex = 'M'
15        elif data.get('PatientSex') == 'Kobieta':
16            dataset.PatientSex = 'F'
17    if data.get('PatientBirthDate'):
18        dataset.PatientBirthDate = dicom_date_display_to_dataset(data.get('PatientBirthDate'))
19    if data.get('ImageComments'):
20        dataset.ImageComments = data.get('ImageComments')
21    return dataset

```

Listing 9: Wpisywanie danych do zbioru danych DICOM

Użytkownik posiada wybór, czy umieścić w pliku DICOM obraz wyjściowy, czy też wejściowy. Ta druga opcja może być przydatna, gdy ma on zamiar jedynie uzupełnić dane w już istniejącym pliku.

```

1 def dicom_save(file_name, dataset, image):
2     # Upewnienie się, że wyjściowy obraz ma właściwy format
3     if image.dtype != np.uint8:
4         img_max = np.max(image)
5         img_min = np.min(image)
6         image = ((image - img_min) / (img_max - img_min) * 255)
7         image = image.astype(np.uint8)
8     dataset.SpecificCharacterSet = 'utf-8'
9     # Konwersja obrazu i ustawienie jego parametrów
10    dataset.PixelData = image.tobytes()
11    dataset.Rows, dataset.Columns = image.shape
12    dataset.BitsStored = 8
13    dataset.BitsAllocated = 8
14    dataset.HighBit = 7
15    dataset.SamplesPerPixel = 1
16    dataset.PhotometricInterpretation = "MONOCHROME2"
17    dataset.PixelRepresentation = 0
18    dataset.save_as(file_name, write_like_original=False)
19

```

Listing 10: Zapis pliku DICOM

4 Wygląd aplikacji

The screenshot displays the application's main interface with several sections:

- Top Bar:** Contains navigation tabs: "Otwórz plik", "Zapisz obraz wejściowy w formacie DICOM", "Zapisz obraz wyjściowy w formacie JPG", and "Zapisz obraz wyjściowy w formacie DICOM".
- Dane wejściowe:** A large empty area for inputting patient data.
- Edytuj DICOM:** A section for editing DICOM metadata, including fields for "Data badania", "Godzina badania", "ID pacjenta", "Imię pacjenta", "Nazwisko pacjenta", "Płeć pacjenta" (set to "Nieznana"), "Data urodzenia pacjenta", and "Komentarz". A "Zatwierdź" button is at the bottom.
- Ustawienia skanera:** A section for scanner settings, including "Krok $\Delta\alpha$ układu emiter/detektor" (set to 10), "Liczba detektorów (n)" (set to 400), and "Rozwartość / rozpiętość układu emiter/detektor (l)" (set to 100). A "Zatwierdź" button is at the bottom.
- Transformata Radona:** A section with a checkbox "Pokazuj kroki pośrednie" and a "Wykonaj" button. Below are progress bars for "Całkowity postęp", "Postęp iteracji", and "Postęp normalizacji".
- Odtwarzanie obrazu wejściowego:** A section with a checkbox "Pokazuj kroki pośrednie" and a "Wykonaj" button. Below are progress bars for "Całkowity postęp", "Postęp iteracji", and "Postęp normalizacji".

Figure 1: Ekran aplikacji

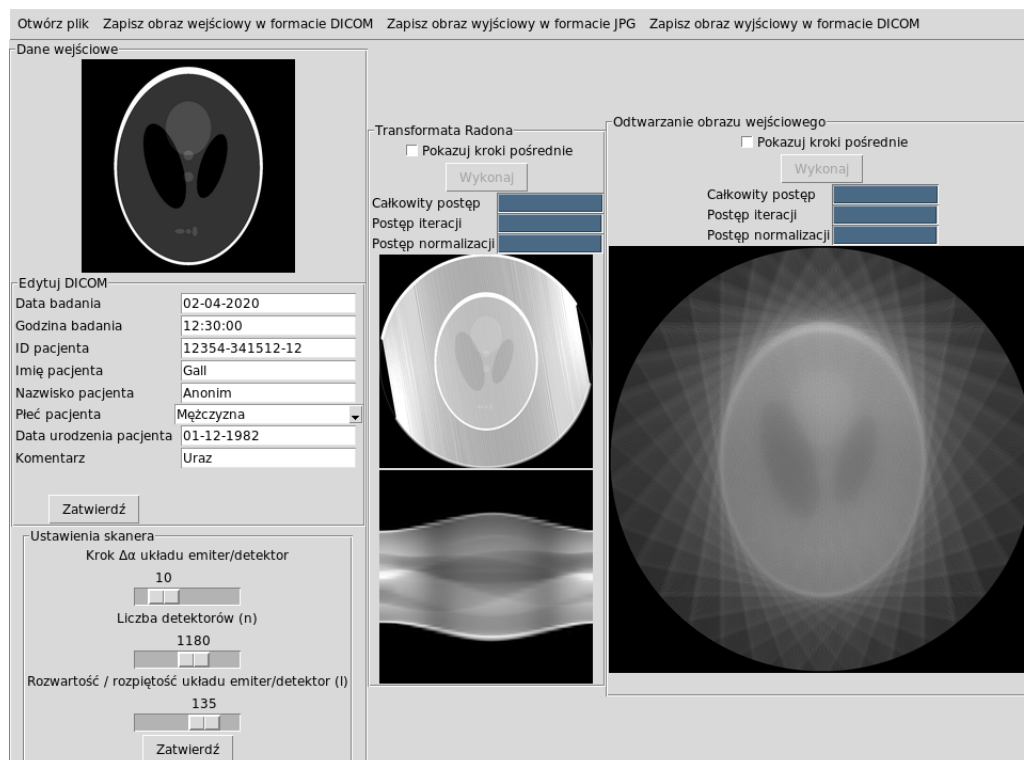


Figure 2: Przykład działania

5 Sprawdzenie poprawności pliku DICOM

Poprawność tworzonych plików DICOM weryfikowana była za pomocą <https://www.ofoct.com/viewer/dicom-viewer-online.html>

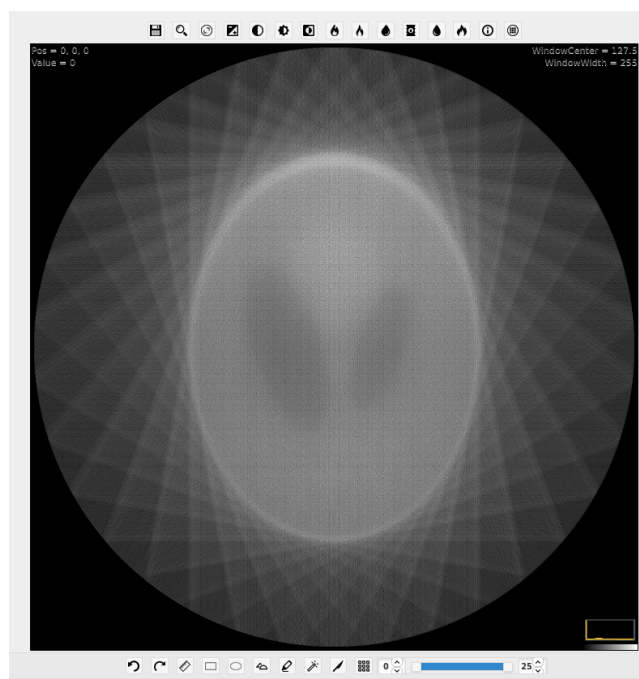


Figure 3: Weryfikacja poprawności obrazu

