Dominik Pawłowski, nr 145289, L16
Bartłomiej Kowalewski, nr 145204, L15
Środa 15:10

Informatyka w medycynie - Projekt I
Symulator tomografu komputerowego

1 Zastosowany model tomografu

W naszym programie zastosowaliśmy równoległy model tomografu. Jego implementacja jest mniej skomplikowana niż w przypadku stożkowego modelu, gdyż mając pozycje emiterów, pozycje detektorów można wyznaczyć w analogiczny sposób.

2 Wykorzystany język programowania

Wykorzystany przez nas język programowania to Python. Wybraliśmy go, ponieważ mamy w nim największe doświadczenie i posiada on przydatne biblioteki, które można było wykorzystać w naszym programie.

3 Dodatkowe biblioteki

W naszym programie skorzystaliśmy z następujących bibliotek:

- Numpy: Biblioteka wykorzystywana do wykonywania operacji i obliczeń na macierzach
- Matplotlib: Biblioteka wykorzystywana do wizualizacji
- OpenCV: Biblioteka wykorzystywana do wczytywania obrazów
- Pydicom: Biblioteka wykorzystywana do zapisu i odczytu plików w formacie DICOM
- Ipywidgets: Biblioteka wykorzystywana do wprowadzenia interakcji

4 Opis głównych funkcji programu

4.1 Pozyskiwanie odczytów dla poszczególnych detektorów

```
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)
    result = rescale(np.array([np.sum(image[tuple(line)]) for line
        in lines]))
    return result
```

W celu pozyskania odczytów dla poszczególnych detektorów najpierw wyznaczamy ich współrzędne oraz współrzędne odpowiadających im emiterów. Następnie, korzystając z algorytmu Bresenhama, wyznaczamy proste przechodzące przez parę emiter-detektor. Po uzyskaniu wszystkich linii dla danego ułożenia sumujemy piksele, przez które przechodzą poszczególne linie i tworzymy jeden wiersz sinogramu. Proces powtarzamy dla każdego położenia do otrzymania pełnego sinogramu.

4.2 Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe

```
def inverse_radon(image, tmp, single_alpha_sinogram, 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)] += single_alpha_sinogram[i]
        num_of_lines[tuple(line)] += 1
```

Aby uzyskać obraz wynikowy na podstawie sinogramu wyznaczamy w pierwszej kolejności pozycje emiterów oraz detektorów i linie przechodzące pomiędzy nimi (algorytm Bresenhama). Następnie, korzystając z wyznaczonych linii, uzupełniamy macierz wynikową obrazu. Normalizację obrazu końcowego przeprowadzamy z wykorzystaniem macierzy `num_of_lines`, która zlicza ile razy dany piksel został ujęty przez parę emiter-detektor. Każda komórka macierzy obraz wynikowego po zakończeniu przetwarzania jest dzielona przez odpowiadającą jej komórkę z macierzy `num_of_lines`.

4.3 Odczyt i zapis pików DICOM

Do odczytu i zapisu plików w formacie DICOM skorzystaliśmy z dostępnej dla języka Python biblioteki Pydicom.

Poniżej zamieszczona została funkcja, która służy do odczytu pliku. Jako argument przyjmuje ona jego ścieżkę. Funkcja wyświetla wszystkie dane oraz dołączony obraz.

```
def dicom_read(file):
    dicom = pd.dcmread(file)
    print(dicom)
    plt.imshow(np.array(dicom.pixel_array).astype(np.uint8), cmap=
        'gray');
```

Do zapisu pliku wykorzystujemy funkcję `dicom_write`, której kod został zamieszczony poniżej:

```
def dicom_write(file, data):
    file_meta = FileMetaDataset()
    file_meta.MediaStorageSOPClassUID = UID('
        1.2.840.10008.5.1.4.1.1.2')
    file_meta.MediaStorageSOPInstanceUID = generate_uid()
    file_meta.ImplementationClassUID = generate_uid()
```

```
file_meta.TransferSyntaxUID = UID('1.2.840.10008.1.2.1')

ds = FileDataset(file, {}, file_meta=file_meta, preamble=b'\0'
    *128)
ds.PatientName = data['name']
ds.PatientID = data['id']
ds.is_little_endian = True
ds.is_implicit_VR = False
ds.StudyDate = data['date'] #.strftime('%Y%m%d')

ds.SeriesInstanceUID = generate_uid()
ds.StudyInstanceUID = generate_uid()
ds.FrameOfReferenceUID = generate_uid()

ds.BitsStored = 8
ds.BitsAllocated = 8
ds.SamplesPerPixel = 1
ds.HighBit = 7

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

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

ds.RescaleIntercept = '0'
ds.RescaleSlope = '1'
```

```
ds.PixelSpacing = r'1\1'
ds.PhotometricInterpretation = 'MONOCHROME2'
ds.PixelRepresentation = 0

ds.ImageComments = data['comment']
ds.PixelData = (data['image']).astype(np.uint8).tobytes()

validate_file_meta(ds.file_meta, enforce_standard=True)

ds.save_as(file, write_like_original=False)
print('File created')
dicom_read(file)
```

Funkcja jako argumenty przyjmuje nazwę pliku, do którego mają zostać zapisane dane oraz słownik z danymi do umieszczenia w pliku.
Pliki zapisywane są poprawnie co sprawdziliśmy w jednej z podanych przeglądarek plików DICOM (`https://www.imaios.com/en/Imaios-Dicom-Viewer`).