Symulacja Tomografu Komputerowego

Informatyka w Medycynie – projekt

Wykonanie: Paulina Pacura, 142179

1. Przedstawienie problemu oraz wybranych metod rozwiązania

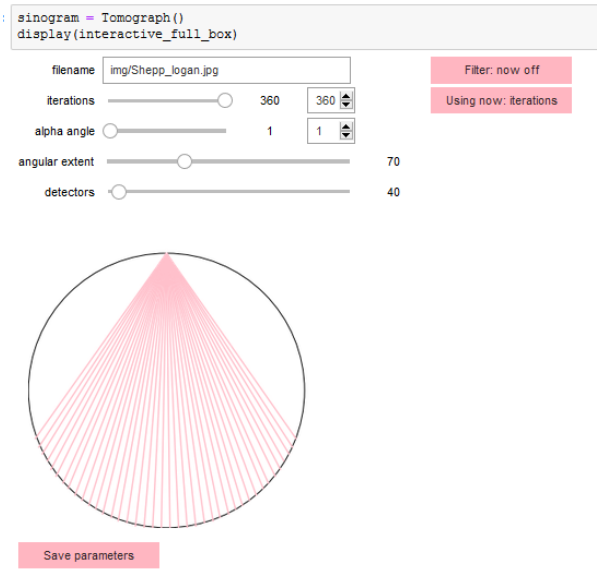
Stworzony program to implementacja **transformaty Radona** oraz odwrotności tego procesu. W wyniku działania transformaty program generuje sinogram, z którego następnie odtwarzany jest obraz wejściowy. Do implementacji zastosowany został **model stożkowy**.

Jako interfejsu aplikacji użyty został **Jupyter Notebook** z wykorzystaniem interaktywnych suwaków dla umożliwienia łatwego modyfikowania parametrów przez użytkownika. Program napisany został w języku **Python**.

Użyte biblioteki:

- numpy
- matplotlib
- math
- pillow
- pydicom
- scipy
- sklearn

Interaktywna obsługa programu:



2. Opis głównych funkcji programu

Tomograf zaimplementowany został jako samodzielna klasa, której pola to odpowiednie parametry symulacji. Początkowo w tomografie ustawiany jest obraz, który konwertowany jest do rozmiaru kwadratu oraz dla tak pozyskanego rozmiaru obliczany jest promień koła, po którym krążyć będą emiter i detektory. Użytkownik ustala również ilość detektorów, rozpiętość kątową między skrajnymi detektorami, a także ilość iteracji lub kąt dla pojedynczej iteracji.

W trakcie tworzenia sinogramu dla każdego kąta w kole obliczane są pozycje emitera i detektorów oraz dzięki algorytmowi Bresenhama piksele, które wchodzą w skład każdej linii promienia. Piksele z każdej linii są sumowane; tak skonstruowana lista wartości staje się sinogramem, z którego dalej odtwarzany jest obraz wyjściowy poprzez sumowanie wartości sinogramu dla kolejnych pikseli.

2.1. Pozyskiwanie odczytów dla poszczególnych detektorów

Pozyskiwanie odczytów zaimplementowane zostało zgodnie z podanymi wzorami. Do funkcji przekazany zostaje kąt akualnej rotacji: na jej podstawie obliczana jest pozycja emitera oraz wszystkich detektorów.

```
def find_points_positions_cone(self,current_rotation_angle):

    angle_radians = np.deg2rad(self._angular_extent)
    picture_center = self._size/2
    emitter_x = int(picture_center + self._radius*np.cos(np.deg2rad(current_rotation_angle)))
    emitter_y = int(picture_center + self._radius*np.sin(np.deg2rad(current_rotation_angle)))

    detectors=[]
    for detector in range(self._detectors_number):
        angle_calc_formula = np.deg2rad(current_rotation_angle) + np.pi - angle_radians/2 + detector * angle_radians
        detector_x = int(self._radius*np.cos(angle_calc_formula)+ picture_center)
        detector_y = int(self._radius*np.sin(angle_calc_formula)+ picture_center)
        detectors.append([detector_x,detector_y])
    return [emitter_x, emitter_y], detectors
```

2.2. Filtrowanie sinogramu, zastosowany rozmiar maski

Program daje możliwość zastosowania maski. Po jej wygenerowaniu wykonywany jest splot sinogramu z maską, co poprawia czytelność generowanych obrazów wyjściowych. Wybrałam dynamiczne zmienianie rozmiaru maski na podstawie ilości detektorów (rozmiar maski = ilość detektorów / 2).

```
if self._with_filter:
    for i in range(self._iterations_number):
        sinogram[i] = convolve(sinogram[i], self.get_mask())
```

```
def get_mask(self):
    mask_size = self._detectors_number//2
    mask = np.zeros(( mask_size))

    top = -4/(np.pi**2)
    center = mask_size // 2
    for i in range(0, mask_size):
        mask[i] = ( 0 if (i-center)%2==0 else (top/((i-center)**2)) )
    mask[center] = 1
    return mask
```

2.3. Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe (np. uśrednianie, normalizacja)

Jako przetwarzanie końcowe obrazu wynikowego użyłam wbudowanego zwiększania kontrastu i manipulacji jasności biblioteki Pillow oraz ręcznego przyciemniania pikseli poniżej pewnego pułapu jasności. Dodatkowo wykonywana jest także normalizacja obrazu.

```
def fft_filter(self,image):
    image = gaussian_filter(image, sigma=2)
    fft = fftpack.fft2(image)
    keep_fraction = 0.2
    h,w = fft.shape
    fft[int(h * keep_fraction): int(h*(1 - keep_fraction))] = 0
    fft[:,int(w*keep_fraction):int(w*(1 - keep_fraction))] = 0
    img = fftpack.ifft2(fft).real
    return  img
```

```
def normalize_image_to_one(self,img):
    norm = np.linalg.norm(img)
    output_image = image / norm * 1.0
    return output_image
```

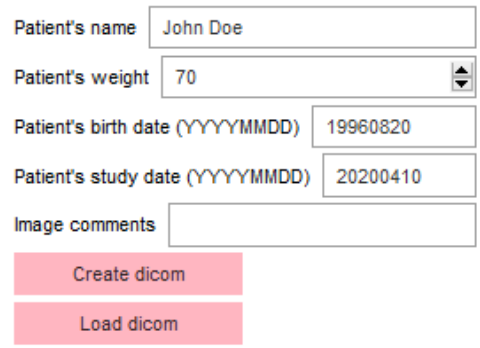
```python
def upgrade_image(self,image):
    new_im = np.round((image - np.amin(image)) / (np.amax(image) - np.amin(image)) * 255)

    new_im = self.fft_filter(new_im)
    new_im = Image.fromarray(np.uint8(new_im))
    new_im = ImageOps.autocontrast(new_im, cutoff=0.7)
    enhancer = ImageEnhance.Brightness(new_im)
    factor = 0.3
    new_im = enhancer.enhance(factor)

    image = np.asarray(new_im,dtype="float64")
    if self._with_filter:

        image=gaussian(image,sigma=3)

        #gentle dark everything except the brightest part
        mean = np.mean(image)
        bright_level = mean*1.5
        coords = np.column_stack(np.where(image < bright_level))
        mask = image < bright_level
        image[mask] -= mean*0.7

        #totally dark the darkest part:
        mean = np.mean(image)
        darkest_threshold_level = mean*1.2
        darkest_coords = np.column_stack(np.where(image < darkest_threshold_level))
        darkest_mask = image < darkest_threshold_level
        image[darkest_mask] = 0

    image = self.normalize_image_to_one(image)
    return image
```

2.4. Odczyt i zapis plików DICOM

Program zaopatrzony jest także w funkcję odczytu i generowania plików DICOM.

```python
def save_dicom_file(self, patient_data, filename="files/test.dcm"):
    filename_little_endian = tempfile.NamedTemporaryFile(suffix=".dcm").name
    file_meta = Dataset()
    file_meta.MediaStorageSOPClassUID = '1.2.840.10008.5.1.4.1.1.2'
    file_meta.MediaStorageSOPInstanceUID = "1.2.3"
    file_meta.ImplementationClassUID = "1.2.3.4"

    ds = FileDataset(filename_little_endian, {},
                     file_meta=file_meta, preamble=b"\0" * 128)
    ds.PatientName = patient_data["PatientName"]
    ds.PatientWeight = patient_data["PatientWeight"]
    ds.ImageComments = patient_data["ImageComments"]
    ds.StudyDate = patient_data["StudyDate"]
    ds.PatientBirthDate = patient_data["BirthDate"]

    ds.is_little_endian = True
    ds.is_implicit_VR = True

    array = self._output_image
    ds.PixelData = np.array(array*255, dtype=np.int8).tobytes()
    ds.Rows, ds.Columns = array.shape
    ds.file_meta.TransferSyntaxUID = pydicom.uid.ImplicitVRLittleEndian

    ds.PixelRepresentation = 0
    ds.BitsAllocated = 8
    ds.SamplesPerPixel = 1
    ds.NumberOfFrames = 1
    ds.PhotometricInterpretation = "MONOCHROME"
    ds.PlanarConfiguration = 0

    ds.save_as(filename)
    print("File saved.")
```

```
display(dicom_box)
```

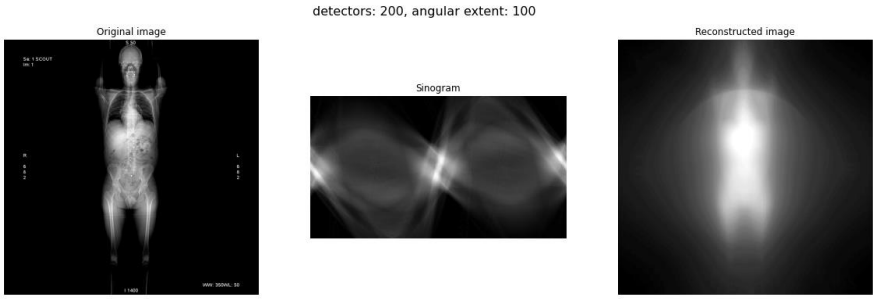


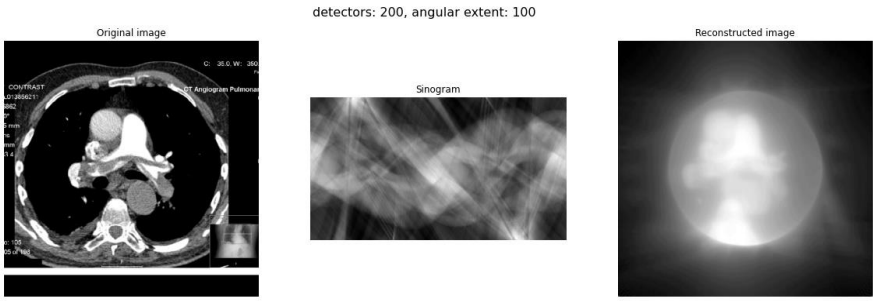

```python
def load_dicom_file(self, filename="files/test.dcm"):
    ds = dcmread(filename)
    image_size = (int(ds.Rows), int(ds.Columns))
    array_picture = np.zeros(image_size, dtype=np.int8)
    array_picture = ds.pixel_array
    patient_data = {}

    if "PatientName" in ds:
        patient_data["PatientName"]=ds.PatientName
    if "ImageComments" in ds:
        patient_data["ImageComments"]=ds.ImageComments
    if "StudyDate" in ds:
        patient_data["StudyDate"]=ds.StudyDate
    if "BirtdDate" in ds:
        patient_data["BirthDate"]=ds.BirthDate
    if "PatientWeight" in ds:
        patient_data["PatientWeight"]=ds.PatientWeight

    plt.imshow(array_picture, cmap=plt.cm.bone)
    return  array_picture, patient_data
```

3. Przedstawienie wyników działania symulacji

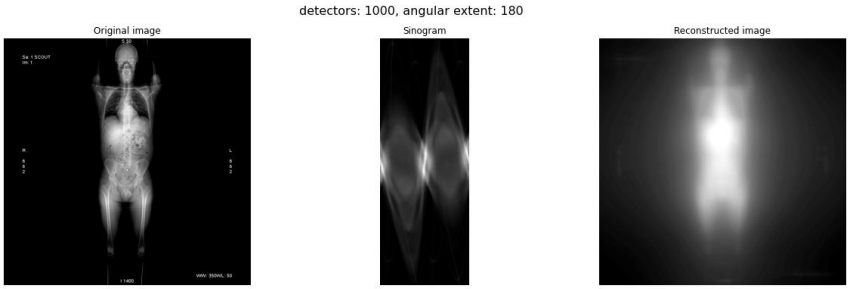
Poniżej przedstawiam przykładowe wyniki działania symulacji. Pozostałe zestawienia znajdują się w katalogu img/image-results.

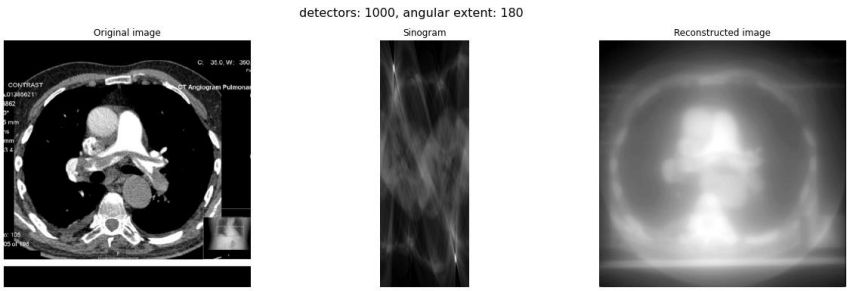
3.1. Brak filtra, niskie ustawienia



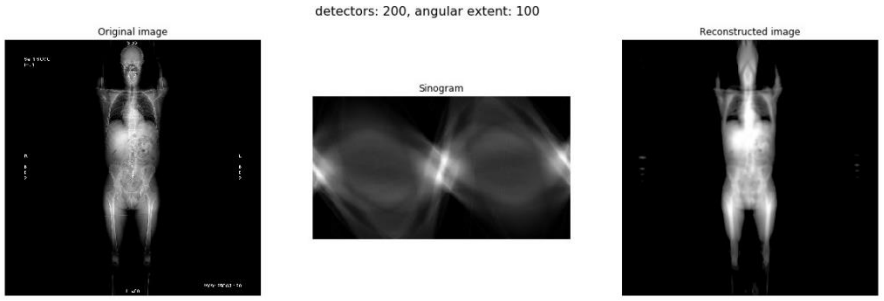


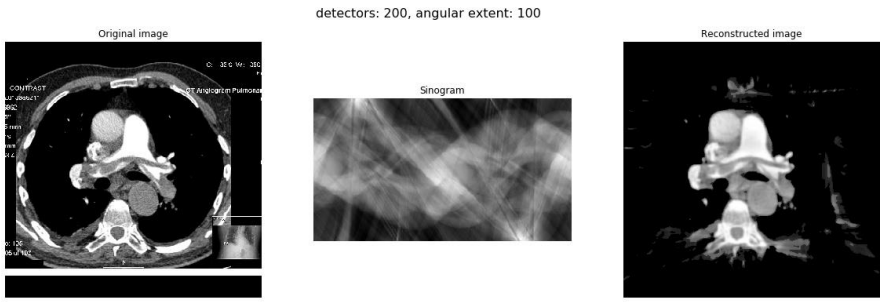
3.2. Brak filtra, wysokie ustawienia



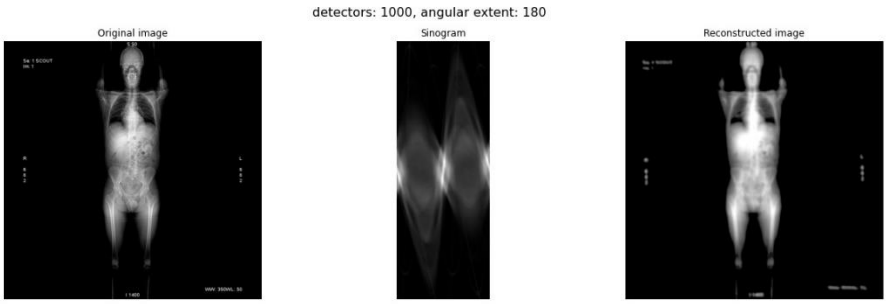


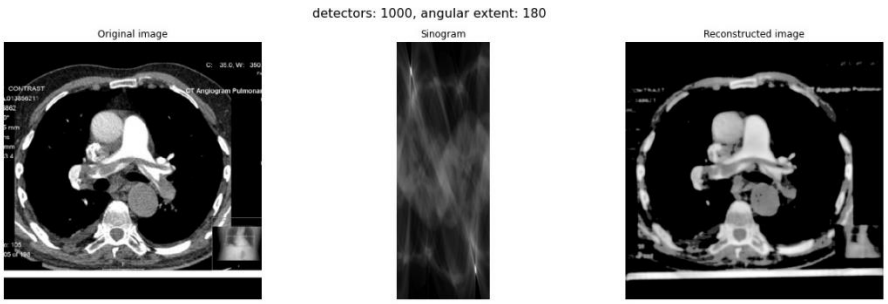
3.3. Filtr, niskie ustawienia





3.4. Filtr, wysokie ustawienia





4. Wnioski

4.1. Dla regularnych kształtów odtwarzanie obrazu oryginalnego wychodziło stosunkowo dokładne. Na jakość obrazu wynikowego wpływ miały dobrane parametry – wraz z liczbą detektorów znacząco zwiększała się jakość odtworzenia. W wyniku odwrotnej transformaty pojawiały się także liczne szumy, wpływające na ostateczne podobieństwo obrazu oryginalnego oraz wyjściowego.

4.2. Zastosowane filtry pozwoliły na wydobycie większej ilości szczegółów bardziej skomplikowanych obrazów. Odpowiednie obrabianie jasności pikseli końcowego obrazu pozwoliło na redukcję szumów i uzyskanie dość dokładnych rezultatów.