Symulator tomografu komputerowego

Adrian Kokot (148165) i Wiktor Szymański (148084)

---

1. **Skład grupy**: Adrian Kokot (148165), Wiktor Szymański (148084)
2. **Zastosowany model tomografu**: stożkowy
3. **Zastosowany język programowania oraz dodatkowe biblioteki**:

   Wykorzystaliśmy język python wraz z bibliotekami: os, datetime, numpy, matplotlib, PIL, skimage, pydicom, streamlit

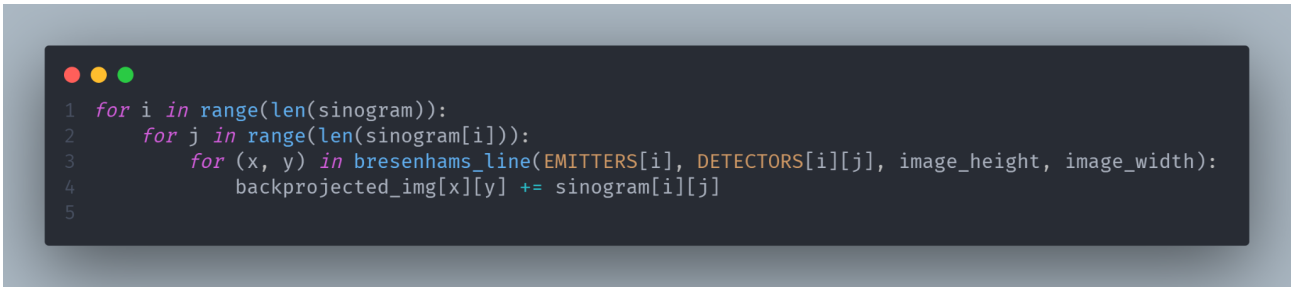
4. Opis głównych funkcji programu
   a. **pozyskiwanie odczytów dla poszczególnych detektorów**

```python
sinogram = np.zeros((angles_num, detectors_num))

for angle_num in range(angles_num):
    alpha_rad = np.radians(angles[angle_num])
    fi_rad = np.radians(fi)

    E_x = int(radius * np.cos(alpha_rad)) + center_x
    E_y = int(radius * np.sin(alpha_rad)) + center_y
    emitter = (E_x, E_y)

    EMITTERS.append(emitter)
    DETECTORS.append([])

    for i in range(detectors_num):
        D_x = int(radius * np.cos(alpha_rad + np.pi - fi_rad / 2 +
                                  i * (fi_rad / (detectors_num - 1)))) + center_x
        D_y = int(radius * np.sin(alpha_rad + np.pi - fi_rad / 2 +
                                  i * (fi_rad / (detectors_num - 1)))) + center_y

        detector = (D_x, D_y)

        DETECTORS[-1].append(detector)

        line = bresenhams_line(
            emitter, detector,  image_height, image_width)

        sinogram_value = 0
        points = 0

        for (x, y) in line:
            sinogram_value += image[x][y]
            points += 1

        if points > 0:
            sinogram_value = sinogram_value / points

        sinogram[angle_num][i] = sinogram_value
```

Rys. 1. Ustalanie pozycji detektorów oraz obliczanie sinogramu

```python
def bresenhams_line(start_point, end_point, height, width):
    points = []

    def get_pixel(i, j):
        if i >= 0 and i < width and j >= 0 and j < height:
            return (i, j)
        return None

    x1, y1 = start_point
    x2, y2 = end_point

    rise = y2 - y1
    run = x2 - x1

    if run == 0:
        if y2 < y1:
            y1, y2 = (y2, y1)
        for y in range(y1, y2 + 1):
            pixel = get_pixel(x1, y)

            if (pixel):
                points.append(pixel)
    else:
        m = float(rise) / run
        adjust = 1 if m >= 0 else -1
        offset = 0

        if m <= 1 and m >= -1:
            delta = abs(rise) * 2
            threshold = abs(run)
            threshold_inc = abs(run) * 2
            y = y1
            if x2 < x1:
                x1, x2 = (x2, x1)
                y = y2
            for x in range(x1, x2 + 1):
                pixel = get_pixel(x, y)

                if (pixel):
                    points.append(pixel)

                offset += delta
                if offset >= threshold:
                    y += adjust
                    threshold += threshold_inc
        else:
            delta = abs(run) * 2
            threshold = abs(rise)
            threshold_inc = abs(rise) * 2
            x = x1
            if y2 < y1:
                y1, y2 = (y2, y1)
                x = x2
            for y in range(y1, y2 + 1):
                pixel = get_pixel(x, y)

                if (pixel):
                    points.append(pixel)

                offset += delta
                if offset >= threshold:
                    x += adjust
                    threshold += threshold_inc

    return points
```

Rys. 2. Algorytm Bresenhama



```python
for i in range(len(sinogram)):
    for j in range(len(sinogram[i])):
        for (x, y) in bresenhams_line(EMITTERS[i], DETECTORS[i][j], image_height, image_width):
            backprojected_img[x][y] += sinogram[i][j]

```

Rys. 3. Uzyskanie obrazu z sinogramu

b. **filtrowanie sinogramu, zastosowany rozmiar maski** (wymaganie na 5.0)

```python
def create_kernel(kernel_size: int):
    kernel = list()

    for k in range(-(kernel_size // 2), int(np.ceil(kernel_size / 2))):
        if k == 0:
            kernel.append(1)
        else:
            kernel.append(0) if k % 2 == 0 else kernel.append((-4 / np.pi**2)/(k**2))

    return kernel


def image_filtering(sinogram, kernel_size):
    kernel = create_kernel(kernel_size)

    for i in range(len(sinogram)):
        sinogram[i] = np.convolve(sinogram[i], kernel, mode='same')

    return sinogram
```

Rys. 4. Filtrowanie sinogramu

Zastosowaliśmy maskę o wielkości 15.

c. **ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe** (np. uśrednianie, normalizacja),
Ustalanie jasności poszczególnych punktów następuje poprzez sumowanie jasności pixeli i podzielenie tej wartości przez ilość zeskanowanych pixeli (normalizacja). W szerszym kontekście widać to na rysunku 1. w linijce 37.

```
sinogram_value = 0
points = 0

for (x, y) in line:
    sinogram_value += image[x][y]
    points += 1

if points > 0:
    sinogram_value = sinogram_value / points

sinogram[angle_num][i] = sinogram_value
```

Rys. 5. Ustalenie jasności pixela

d. **wyznaczanie wartości miary RMSE** na podstawie obrazu źródłowego oraz wynikowego (wymaganie na 5.0),

```
def calc_RMSE(original_image, backprojected_image):
  return np.sqrt(np.square(np.subtract(original_image, backprojected_image)).mean())

```

Rys. 6. Wyznaczanie wartości miary RMSE

e. **odczyt i zapis plików DICOM** (wymagania na 4.0).

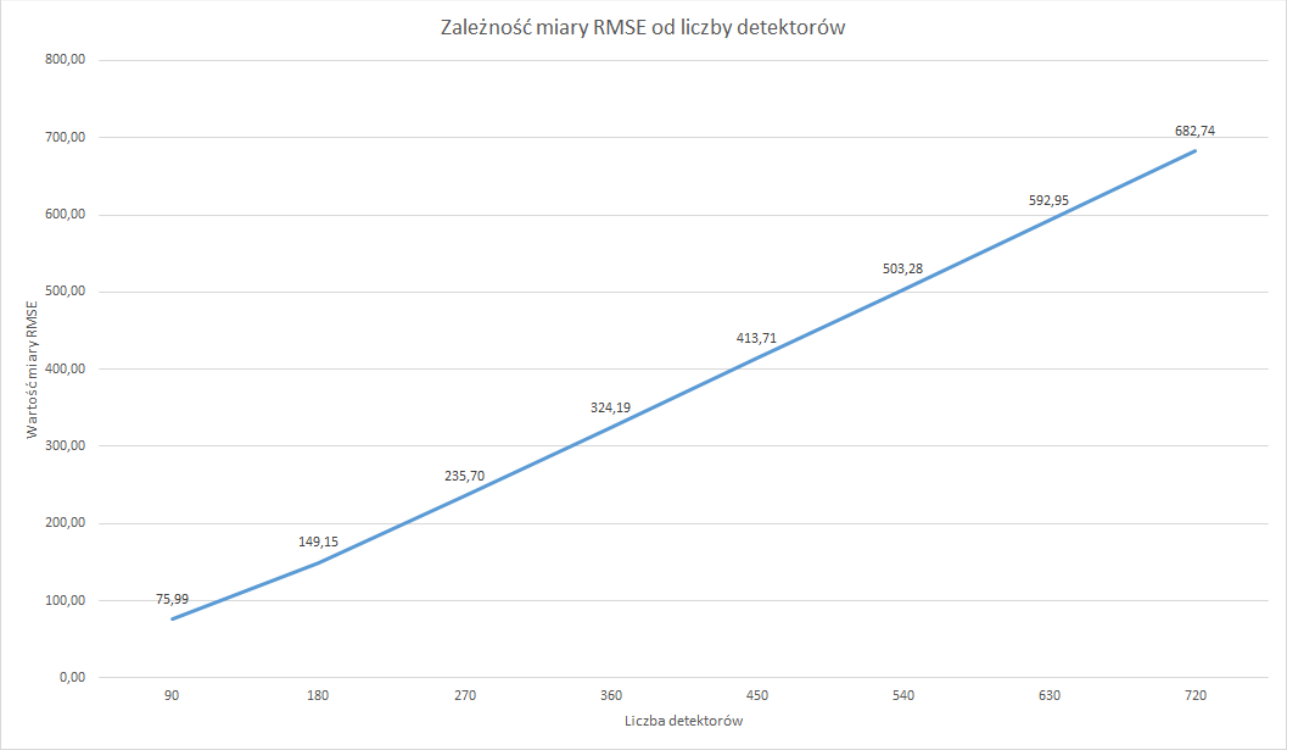
```python

def convert_image_to_ubyte(img):
    return img_as_ubyte(rescale_intensity(img, out_range=(0.0, 1.0)))

def save_as_dicom(file_name, img, patient_data):
    img_converted = convert_image_to_ubyte(img)

    # Populate required values for file meta information
    meta = Dataset()
    meta.MediaStorageSOPClassUID = pydicom._storage_sopclass_uids.CTImageStorage
    meta.MediaStorageSOPInstanceUID = pydicom.uid.generate_uid()
    meta.TransferSyntaxUID = pydicom.uid.ExplicitVRLittleEndian

    ds = FileDataset(None, {}, preamble=b"\0" * 128)
    ds.file_meta = meta

    ds.is_little_endian = True
    ds.is_implicit_VR = False

    ds.SOPClassUID = pydicom._storage_sopclass_uids.CTImageStorage
    ds.SOPInstanceUID = meta.MediaStorageSOPInstanceUID

    ds.PatientName = patient_data["PatientName"]
    ds.PatientBirthDate = patient_data["PatientBirthDate"]
    ds.PatientID = patient_data["PatientID"]
    ds.ImageComments = patient_data["ImageComments"]
    ds.StudyDate = patient_data["StudyDate"]

    ds.Modality = "CT"
    ds.SeriesInstanceUID = pydicom.uid.generate_uid()
    ds.StudyInstanceUID = pydicom.uid.generate_uid()
    ds.FrameOfReferenceUID = pydicom.uid.generate_uid()

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

    ds.ImagesInAcquisition = 1
    ds.InstanceNumber = 1

    ds.Rows, ds.Columns = img_converted.shape

    ds.ImageType = r"ORIGINAL\PRIMARY\AXIAL"

    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.PixelRepresentation = 0

    pydicom.dataset.validate_file_meta(ds.file_meta, enforce_standard=True)

    ds.PixelData = img_converted.tobytes()

    ds.save_as(file_name, write_like_original=False)

```

Rys. 7. Zapis plików dicom (lekka modyfikacja pliku z ekursów)

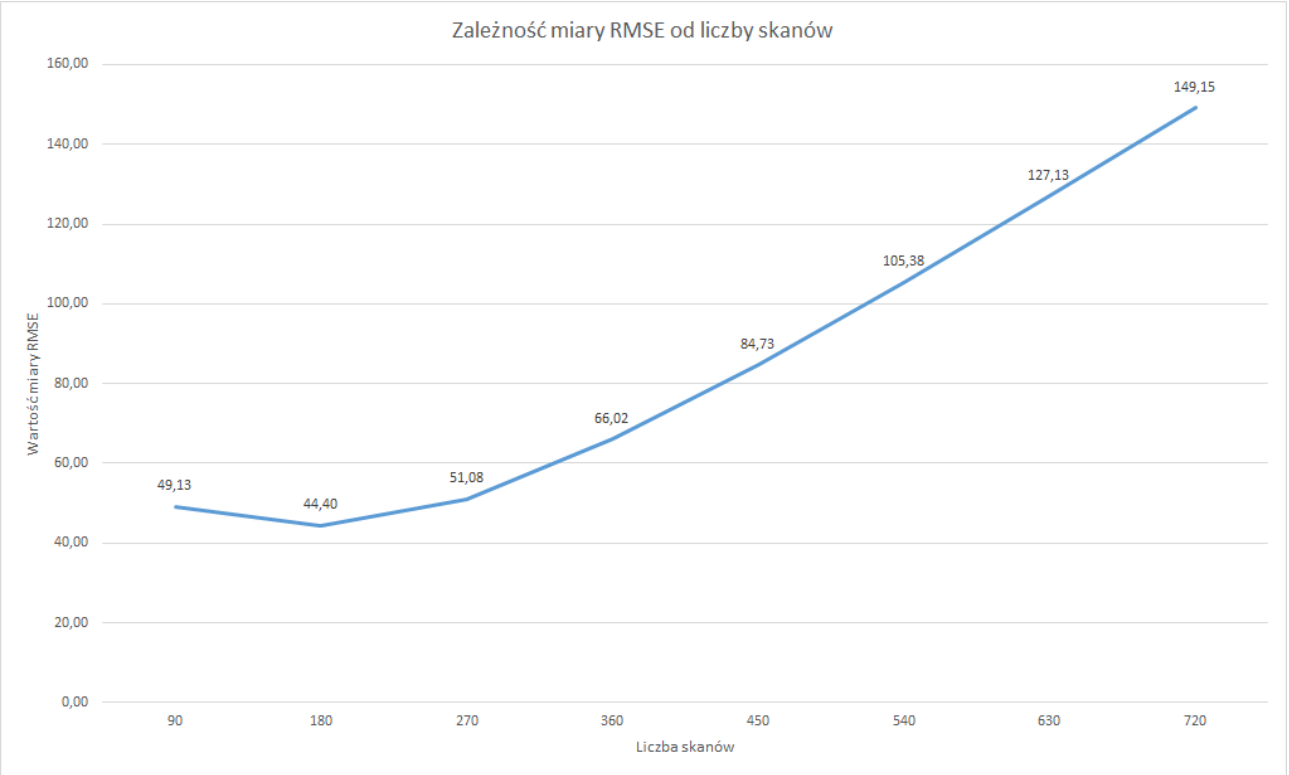
```python
dicom_file_name = dicom_tab.selectbox(
    "Input file",
    os.listdir(DICOM_DIR)
)

dicom_file_path = os.path.join(DICOM_DIR, dicom_file_name)

ds = pydicom.dcmread(dicom_file_path)

dicom_tab.header("Image")

dicom_tab.image(ds.pixel_array)

dicom_tab.subheader("DICOM File data")
dicom_tab.text(ds)
```

Rys. 8. Odczyt plików dicom

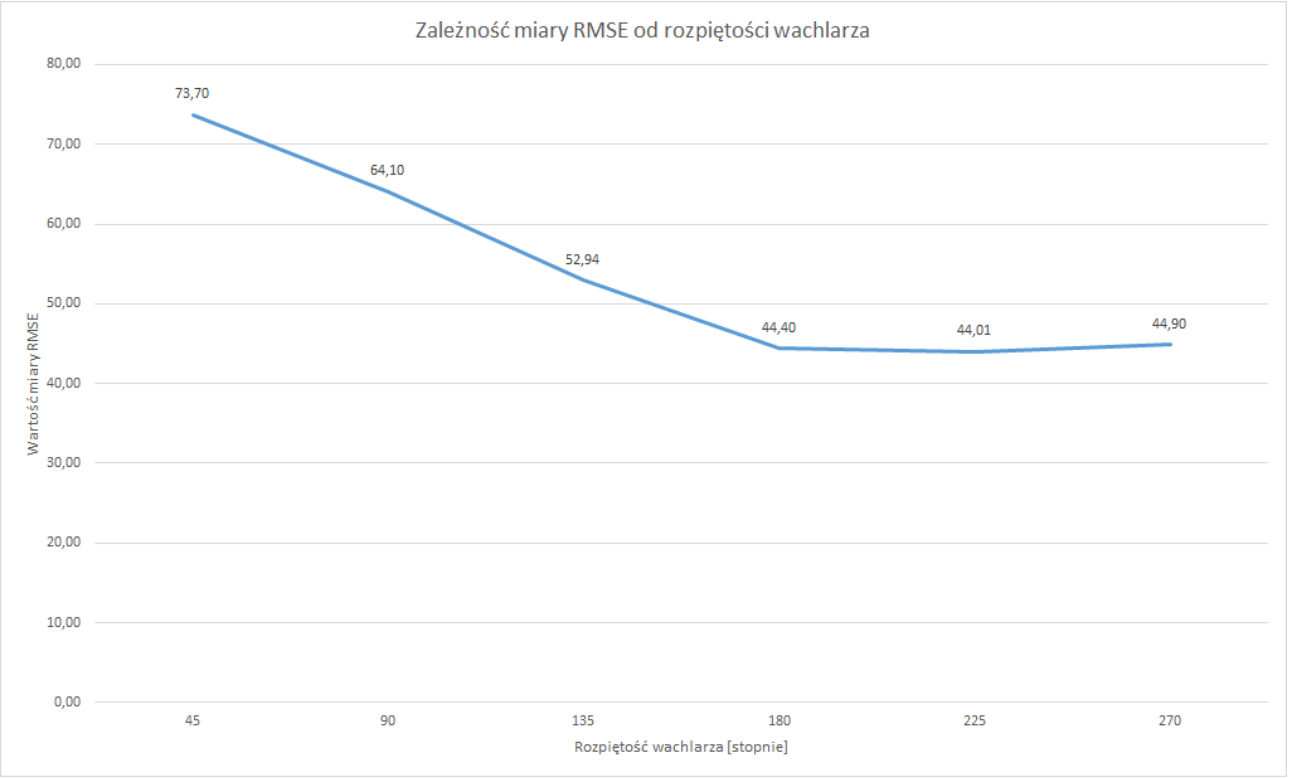
5. Wynik eksperymentu
   a. b.



W przypadku ilości detektorów do RMSE widać niemalże wprost proporcjonalny i liniowy wzrost. Im większa ilość detektorów, tym większe ich zagęszczenie. Z obrazków można zauważyć że wraz ze wzrostem ów zagęszczenia zmniejsza się ilość “luk” w oryginalnym kolorze tła jakim jest kolor czarny i przyjmują te miejsca kolor szary. Z tego faktu mimo iż niektóre miejsca są bardziej zbliżone oryginałowi większość wcześniej czarnych pixeli stała się szara, a mianem tego RMSE zwiększyło się. Patrząc gołym okiem najlepsze wyniki są dla 360 detektorów. Większe ilości sprawiają że obraz jest jaśniejszy przez co jego jasne elementy bardziej się zlewają. Dla mniejszej ilości detektorów cały obraz jest bardziej zszarzały i mniej szczegółowy.



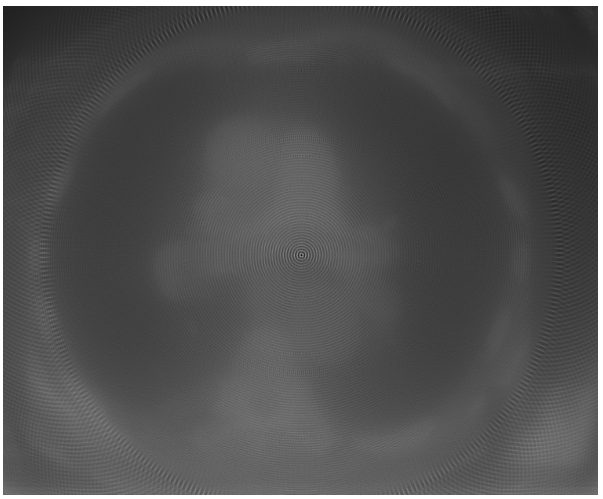
Od 180 skanów na obraz błąd zaczyna rosnąć, gdzie po 270 wzrost ten wygląda na liniowy. Prawdopodobnie występuje tutaj to samo zjawisko co w przypadku liczby detektorów. Dla 180 skanów RMSE jest najmniejsze, ale nie uznałbym że jest to najlepszy obraz. Ze względu na większy kąt pomiędzy kolejnymi skanami na obrazie powstał pewnego rodzaju wzór (kojarzący się z dzianiną). Wydaje się tak przez to, że miejsca gdzie dość dużo wiązek się nie pokryło tworzą spore luki, nasz mózg nie potrafi tego wygładzić w jednolity kolor tak jak ma to miejsce dla obrazów z większą ilością skanów. Wszystkie obrazy powyżej 180 skanów wydają się bardzo podobne z drobną różnicą co do jasności obrazu. Jedne są odrobinę jaśniejsze kolejne ciemniejsze. Nie ma tu jednak zależności tzn. im więcej skanów tym jaśniejszy obraz.



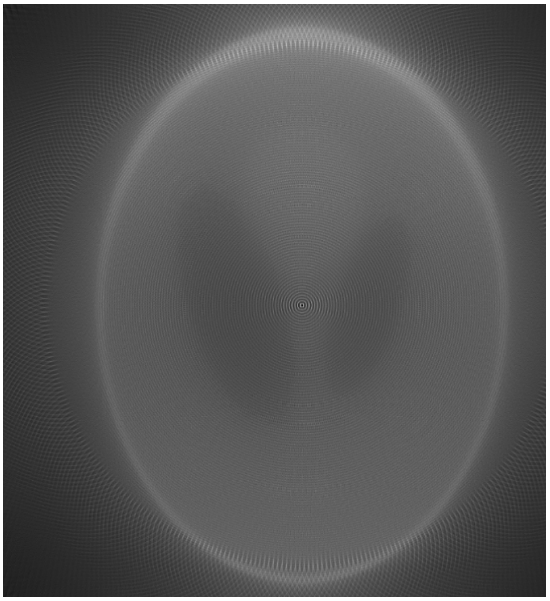
Z wykresu wynika że rozpiętość wachlarza ma znaczenie do kąta 180. Może być to spowodowane faktem że przy takim kącie wiązki są puszczane w każdym kierunku w którego stronę skierowany jest skaner. Przy większej rozpiętości nie miało to większego znaczenia, a mogłoby to wpłynąć negatywnie ze względu na rozrzedzenie gęstości detektorów. Dla obrazów z rozpiętością mniejszą jak 180 na naszym obrazie (Sheep_logan) nie widać pełnego “mózgu”. Jako, że na bardziej zewnętrznych częściach obrazu nie nałożyło się na siebie wystarczająco dużo wiązek, zeskanowany obraz jest tylko widoczny w kółku które powstało na środku obrazu, a jego wielkość jest zależna od wielkości kąta.

c.

SADDLE_PE z filtrowaniem



SADDLE_PE bez filtrowania



Shepp_logan bez filtrowania

Sheep_logan z filtrowaniem

	filtrowanie	bez filtrowania
SADDLE_PE	155.4	4178
Sheep_logan	91.42	2357.44

Obrazy bez zastosowanego filtrowania są bardzo rozmyte i niewyraźne. Ciężko jest zauważyć niektóre elementy a drobniejsze szczegóły praktycznie nie są zauważalne. Po zastosowaniu filtrowania obraz jest znacznie ostrzejszy i dokładniejszy co pozwala nam na zaobserwowanie detali.