Sprawozdanie tomograf

1. Skład grupy.
   Filip Kozłowski 151823
   Wojciech Zysnarski 151872

2. Zastosowany model tomografu.
   Równoległy

3. Zastosowany język programowania oraz dodatkowe biblioteki.
   język programowania: Python 3.11.2
   dodatkowe biblioteki: matplotlib, tkinter, os, numpy, PIL, pydicom

4. Opis głównych funkcji programu (ilustracja za pomocą fragmentów kodu źródłowego)
   a. pozyskiwanie odczytów dla poszczególnych detektorów

```
for iteration, main_angle in enumerate(main_angles):
   sinogram_iteration = []
   for offset_angle in offset_angles:
       emitter_coords = (calculate_emitter_coords
                         (main_angle, offset_angle, img_size / 2))
       receiver_coords = (calculate_receiver_coords
                          (main_angle, offset_angle, img_size / 2))
       beam_points = bresenham_algorithm(emitter_coords, receiver_coords)
       color = []
       for x, y in beam_points:
           if x < img_size and y < img_size:
               color.append(image[x, y])
       sinogram_iteration.append(np.mean(color) if color else 0)
```

b. filtrowanie sinogramu, zastosowany rozmiar maski
Zastosowaliśmy maskę o rozmiarze 21

```
def generate_kernel(size: int) -> np.array:
    if size % 2 == 0 or size < 1:
        return np.array([1])
    middle = size // 2
    filter_kernel = np.ones(size)
    for k in range(1, size // 2 + 1):
        if k % 2 == 0:
            filter_kernel[middle - k] = filter_kernel[middle + k] = 0
        else:
            filter_kernel[middle - k] = filter_kernel[middle + k] = (
                    (-4 / (np.pi ** 2)) / (k ** 2))
    return filter_kernel


def convolution_filter(sinogram: np.ndarray,
                       filter_kernel: np.array) -> np.array:
    filtered_sinogram = np.zeros_like(sinogram)
    for i, row in enumerate(sinogram):
        filtered_row = np.convolve(sinogram[i, :],
                                   filter_kernel, mode='same')
        filtered_sinogram[i, :] = filtered_row
    return filtered_sinogram
```

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

```
beam_points = bresenham_algorithm(emitter_coords,
                                  receiver_coords)
color = sinogram[x_sinogram][y_sinogram]
for x, y in beam_points:
    if x < img_size and y < img_size:
        images[x_sinogram][x, y] += color

# LATER IN CODE
for i in range(len(images)):
    min_val = np.min(images[i][images[i] != 0])
    max_val = np.max(images[i])
    images[i][images[i] == 0] = min_val
    images[i] = ((images[i] - min_val)
                 / (max_val - min_val)) * 255
```

d. wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego

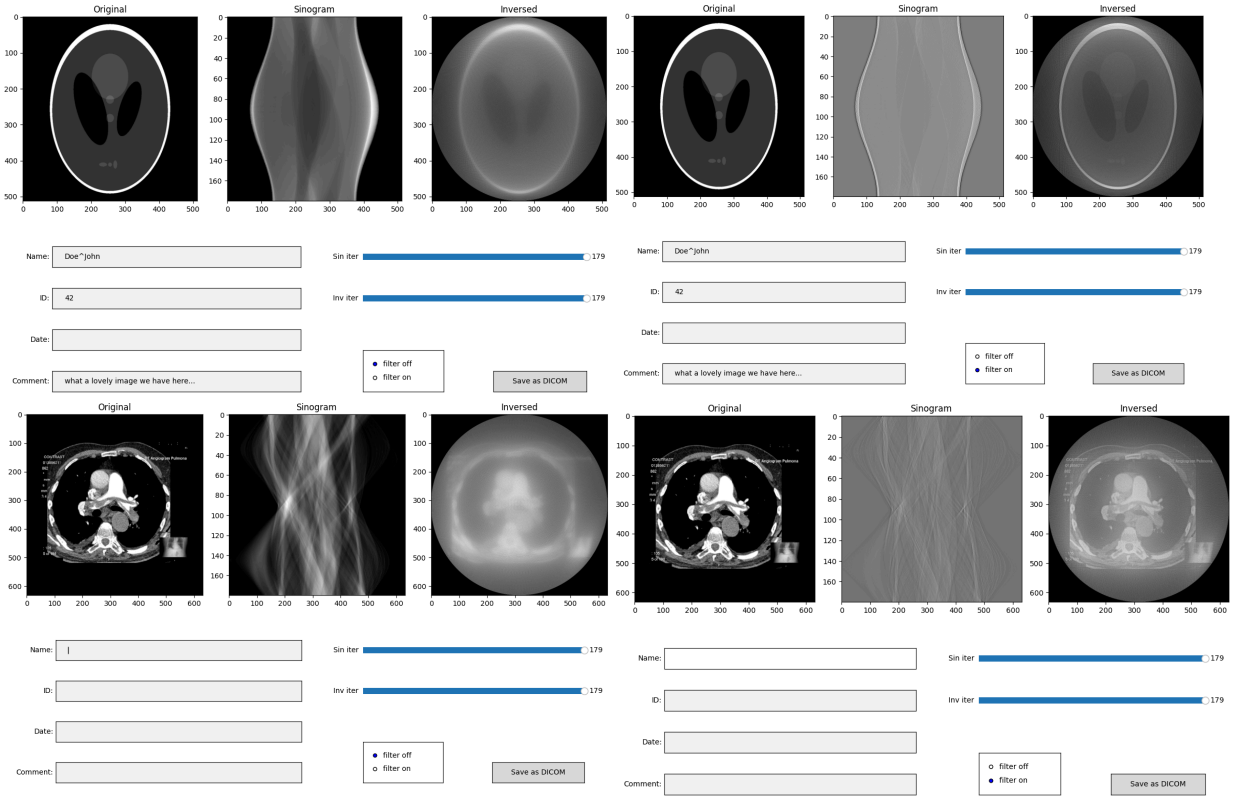
```
def rmse(image1: np.ndarray, image2: np.ndarray) -> float:
    return np.sqrt(np.mean((image1.astype(np.float32) -
                            image2.astype(np.float32)) ** 2)) / 255
```

e. odczyt i zapis plików DICOM

```
def load_file(image_path: str, res: float = 1.0) \
        -> tuple[dict, np.ndarray]:
    _, extension = os.path.splitext(image_path)
    if extension.lower() == '.dcm':
        ds = dcmread(image_path)
        meta = {attr: getattr(ds, attr) for attr in dir(ds)
                if not callable(getattr(ds, attr))}
        img = Image.fromarray(ds.pixel_array.astype(np.uint8))
```

```
def save_dicom(img: np.ndarray, meta: MetaData) -> None:
    file_path = filedialog.asksaveasfilename(
        filetypes=[("DICOM files", "*.dcm")], defaultextension=".dcm")
    if not file_path:
        return

    img_min = np.min(img)
    img_max = np.max(img)
    out_min, out_max = (0.0, 1.0)
    img_converted = (((img - img_min) / (img_max - img_min) *
                      (out_max - out_min) + out_min) *
255).astype(np.uint8)

    # Create FileMetaDataset
    fmd = FileMetaDataset()
    fmd.MediaStorageSOPClassUID = CTImageStorage
    fmd.MediaStorageSOPInstanceUID = generate_uid()
    fmd.TransferSyntaxUID = ExplicitVRLittleEndian

    # Create Dataset
    ds = Dataset()
    ds.PatientName = meta.patientName
    ds.PatientID = meta.patientID
    ds.ImageComments = meta.imageComments

    ds.StudyDate = meta.studyDate
    ds.file_meta = fmd
    ds.is_little_endian = True
    ds.is_implicit_VR = False
    ds.SOPClassUID = CTImageStorage
    ds.SOPInstanceUID = fmd.MediaStorageSOPInstanceUID
    ds.Modality = "CT"
    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.InstanceNumber = 1
    ds.Rows, ds.Columns = img_converted.shape
    ds.ImageType = r"ORIGINAL\PRIMARY\AXIAL"
    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.PixelRepresentation = 0
    ds.PixelData = img_converted.tobytes()

    ds.save_as(file_path, write_like_original=False)
    plt.close()
```

5. Przykład działania programu

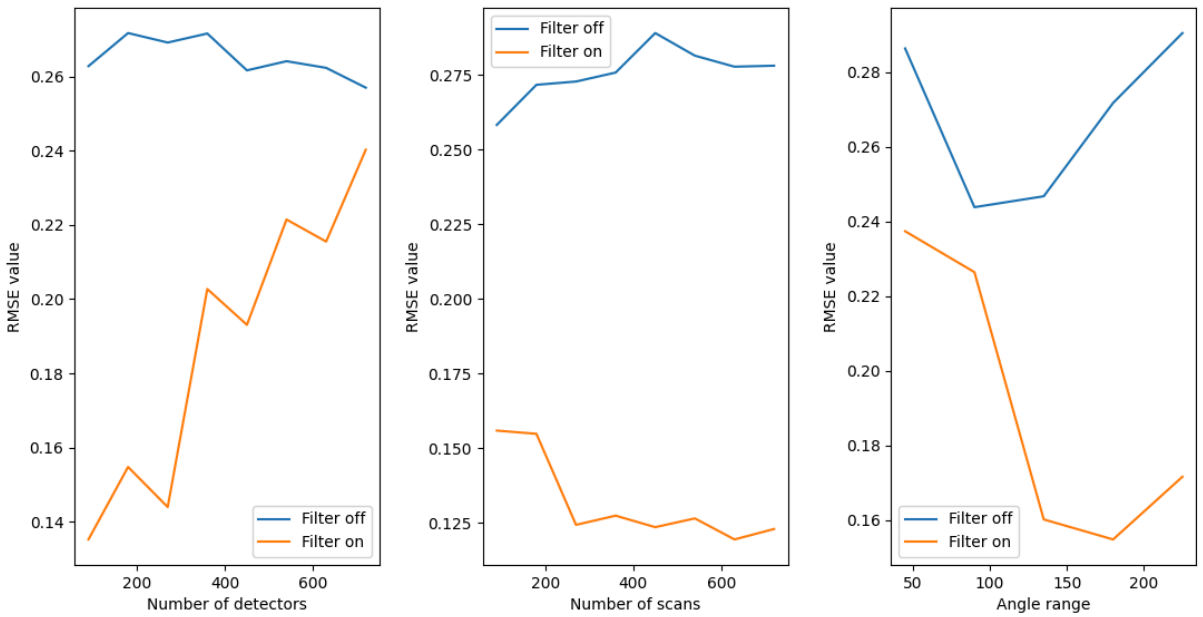


6. Wynik eksperymentu dla Shepp_logan.jpg 256x256

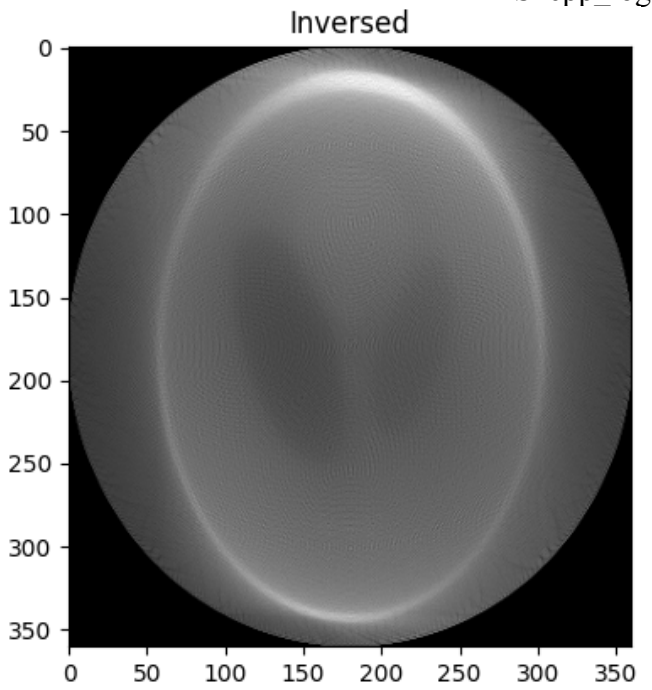
i. Zmienna liczba detektorów - Przy wyłączonym filtrowaniu RMSE maleje, ale w bardzo małym stopniu. Przy włączonym filtrze RMSE znacząco rośnie. Jest to efekt którego się spodziewaliśmy i staraliśmy zwalczyć, niestety nieskutecznie. Wynika on z tego, że niektóre piksele (szczególnie przy emiterach i detektorach) mają bardzo niskie wartości, przez co przy normalizacji całość wychodzi jaśniejsza. RMSE rośnie z powodu słabego kontrastu a nie słabej ostrości.

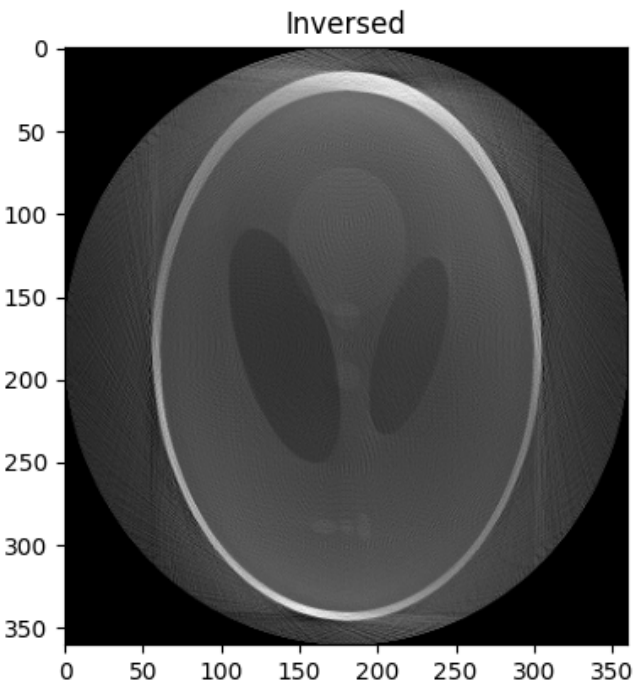
ii. Liczba skanów - Przy wyłączonym filtrowaniu RMSE rośnie a potem maleje. Przy włączonym filtrowaniu RMSE maleje.

iii. Rozpiętość wachlarza - Niezależnie od filtra RMSE maleje po czym rośnie. Wzrost spowodowany jest tym samym problemem wspomnianym w punkcie i.



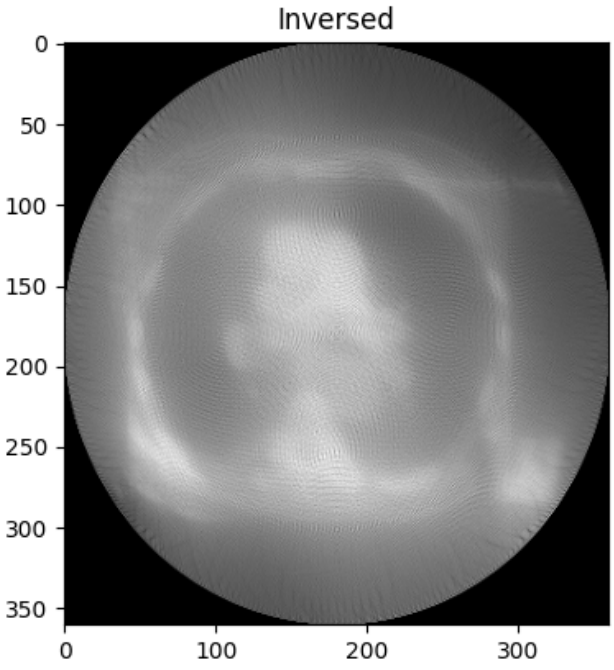
Shepp_logan 1024x1024

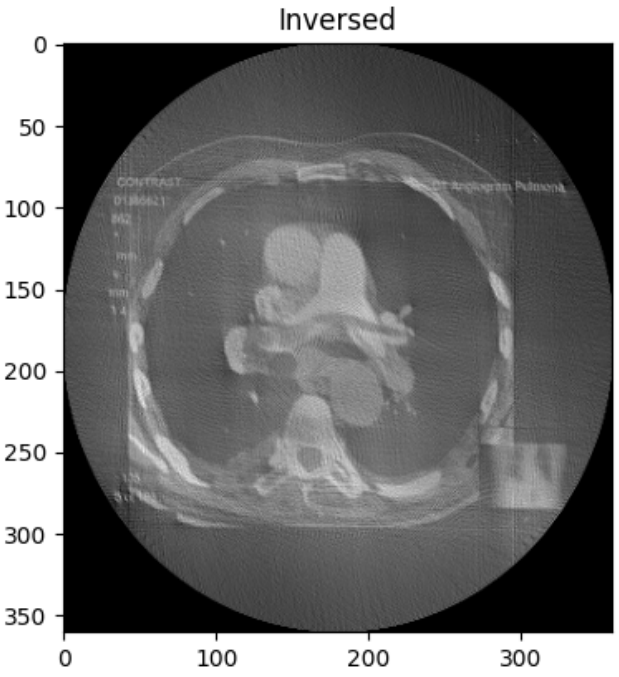




RMSE 0.269 0.182

SADDLE_PE-large.jpg 1264x1264 (scaled)





RMSE 0.369 0.266

Filtr zdecydowanie zwiększa ostrość obrazów.