**Raport z realizacji projektu Tomograf Komputerowy**

1. **Skład grupy –** Jakub Raczyński (136787), Julia Tadej (136820)
2. **Zastosowany model tomografu –** równoległy
3. **Zastosowany język programowania oraz dodatkowe biblioteki –** Projekt został w całości zrealizowany w języku programowania Python. Do zaprojektowania GUI wykorzystaliśmy bibliotekę Qt. Poza tym, w programie wykorzystaliśmy biblioteki takie jak:

**-** Numpy

**-** Scikit Image

**-** Pydicom

1. **Opis głównych funkcji programu**
2. Pozyskiwanie odczytów dla poszczególnych detektorów – Zaczynamy od wyliczenia współrzędnych emitera i detektora w danej iteracji, mając dany kąt położenia oraz promień okręgu, na którym układ się znajduje:

Wynik musi być zaokrąglony do liczby całkowitej, gdyż pracujemy na dyskretnym obrazie składającym się z pikseli.

Następnie iterujemy się wzdłuż pikseli należących do prostej przechodzącej przez emiter i detektor. Działanie to symuluje promień przechodzący przez obrazek. Do liniowego przejścia po kolejnych pikselach obrazu dyskretnego wykorzystaliśmy algorytm Bresenhama. Zdecydowaliśmy się na addytywne modelowanie pochłaniania promieniowania, więc promień na początku jest zerowy, a wraz z napotkaniem jasnych pikseli obrazu (przeszkody) staje się on coraz silniejszy. Jako że algorytm Bresenhama działa minimalnie inaczej dla czterech różnych przedziałów nachyleń prostej po której przechodzimy, poniżej załączamy kod jedynie dla prostych o współczynniku kierunkowym z przedziału (0,1).

 Powyższy kod to po prostu implementacja algorytmu Bresenhama. Jako że dla współczynników kierunkowych z tego przedziału współrzędna x zmienia się szybciej niż y, iterujemy po x-ach. Warto zwrócić uwagę, że cały obrazek jest „przesunięty” o połowę długości i szerokości tak, aby jego środek znajdował się w punkcie o współrzędnych (0,0). W ostateczności po wykonaniu tego algorytmu, w pikselu odpowiadającym danemu sinogramowi w danej pozycji otrzymujemy sumę jasności pikseli, przez które przeszedł nasz promień.

1. Filtrowanie sinogramu, zastosowany rozmiar maski – Maska wykorzystywana w naszym algorytmie ma długość 41 elementów. Nie było sensu wydłużać jej bardziej, gdyż wartości skrajne były bardzo małe i nie miały wpływ na wynik przetwarzania. Sinogram przefiltrowaliśmy wykonując na nim operację splotu z maską. Poniżej zamieszczamy fragment kodu, który odpowiadał za splot:



Przy wykonywaniu splotu trzeba było mieć na uwadze, żeby przesunąć maskę o połowę jej długości, gdyż jest ona symetryczna i piksel aktualnie przetwarzany powinien być przemnażany przez jej środkowy element.

1. Ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe – Jasność poszczególnych punktów obrazu wynikowego jest równa wynikowi odwrotnej transformaty Radona dla danego piksela. Odwrotna transformata Radona przebiegała analogicznie do transformaty pierwotnej. Jedyna różnica względem transformaty pierwotnej, to dodawanie do piksela obrazu wynikowego (przez który przechodził promień, który został pochłonięty przez dany detektor) wartości odpowiadającej mocy promienia odczytanej przez dany detektor w danej pozycji zamiast dodawania do piksela sinogramu jasności piksela obrazu wejściowego. Poniżej

zamieszczam kod dla prostych o współczynnikach kierunkowych z przedziału (-1,0):



1. Wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego – TODO
2. odczyt i zapis plików DICOM – BABA OPOWIE
3. **Wynik eksperymentu sprawdzającego wpływ poszczególnych parametrów (liczba detektorów, liczba skanów, rozpiętość stożka/wachlarza z detektorami) na jakość obrazu wynikowego wyrażoną za pomocą miary RMSE -** TODO