# Laserowy skaner robota

## Cel i opis projektu

Projekt ten ma na celu pomiar odległości z dowolnego punktu na obrazie w odstępie co 10stopni do najbliższej czarnej linii. Następnie koloruje linię, którą „szedł” do tego punktu na kolor czerwony. Do realizacji projektu wykorzystałem biblioteki:

Pillow – biblioteka posłużyła mi do obsługi obrazu, działania na poszczególnych pikselach.

io – biblioteka, a konkretnie część StringIO posłużyła mi do testowania różnych plików tekstowych bez konieczności ich tworzenia.

pytest – bibioteka ta pomogła mi także przeprowadzać testy – sprawdzanie wyjątków

copy – użyłem jej w celu zrobienia deepcopy obrazu.

sys, argparse oraz configparse – możliwość wywołania programu z parametrami danymi lub domyślnymi

math – obliczanie współczynnika kierunkowego prostej – funkcja tangens oraz pi.

## Wejście i wyjście

Program przyjmuje 3 pliki wejściowe. Obraz o wymiarach 320x240p (.png), plik tekstowy (.txt) w postaci:

x = 130

y = 88

α = 226

oraz plik wczytujący konfigurację(.txt) tj. pożądane wymiary obrazu, ilość obrotów i kąt obrotu:

[DEFAULT]

image_width = 320

image_height = 240

rotation_angle = 10

total_angle = 180

Plikami wyjściowymi są pliki analogiczne do plików wejściowych – o tym samym formacie, określane przez użytkownika bądź o nazwach zawartych w treści polecenia.

Użytkownik poprzez „command line” może wywołać program z parametrami określającymi pliki, z których chcemy skorzystać, pliki, w których chcemy zapisać wyniki programu oraz określić charakterystyczne dla programu cechy konfiguracyjne. Gdy nie podamy jakiegoś parametru lub wywołamy program, na przykład, uruchamiając go w VSC, brakujące parametry będą uzupełnione o domyślne, które podane były w treści zadania (konfiguracja to config.txt).

## Podział projektu i opis klas

RobotIO.py - klasa i plik służący do wczytywania i zapisywania danych oraz sprawdzania ich poprawności.

Robot.py – zawiera klasę Robot zajmującą się rysowaniem linii, obliczaniem współczynnika kierunkowego prostej i komunikacją z funkcją bresenham zawartą w pliku bresenham.py, oraz plikiem RobotIO.py

bresenham.py – zawiera funkcję bresenham z zaimplementowanym algorytmem bresenahama dla funkcji liniowej o różnym kącie nachylenia która prowadzi obliczenia kluczowe do działania programu.

# Część refleksyjna

Moja praca nad projektem w dużej mierze opierała się na zrozumieniu operacji w bibliotece Pillow i nauczeniu się operacji na pikselach. Ważnym elementem było także zaimplementowanie funkcji bresenham, którą można zastąpić inną obliczającą po kolei piksele którymi „chodzimy" po obrazie. Pozostałe użyte przeze mnie biblioteki i operacje na różnych rodzajach zmiennych przedstawiane były na laboratoriach.

Z czym pojawił się problem:

- problem z testowaniem innych obrazów – tutaj brak możliwości użycia np. StringIO – 320x240p jest bardzo dużą liczbą krotek opisujących piksele. Sprawdziłem zatem działanie programu na innym pliku png.

- problem z mockowaniem w testach – problem ten wynika z mojego niezrozumienia mockowania, nie byłem w stanie go odpowiednio zrobić. Stąd występują u mnie testy integracyjne.

Planowane rozwiązanie wyszło tak, jak było zadane w poleceniu, jedyna różnica jest taka, że wnioskując po przykładach – moje rozwiązanie działa w kolejności $[\alpha - 90, \alpha + 90]$, zaś w przedstawianych przykładach program działa odwrotnie, tj. od wyższego kąta do niższego, co w mojej opinii nie jest błędem, lecz jedynie własną interpretacją programu (kwestia zmiany znaku, poza tym uznaję, że program operujący na współrzędnych powinien działać od mniejszego do większego kąta, nie zgodnie z ruchem wskazówek zegara), także jest kwestia przybliżenia, która nie została sprecyzowana, przez co może pojawić się drobna różnica w wyniku np. w wyniku przyjęcia innej wartości liczby pi lub użycia innej metody na obliczenie nachylenia prostych.

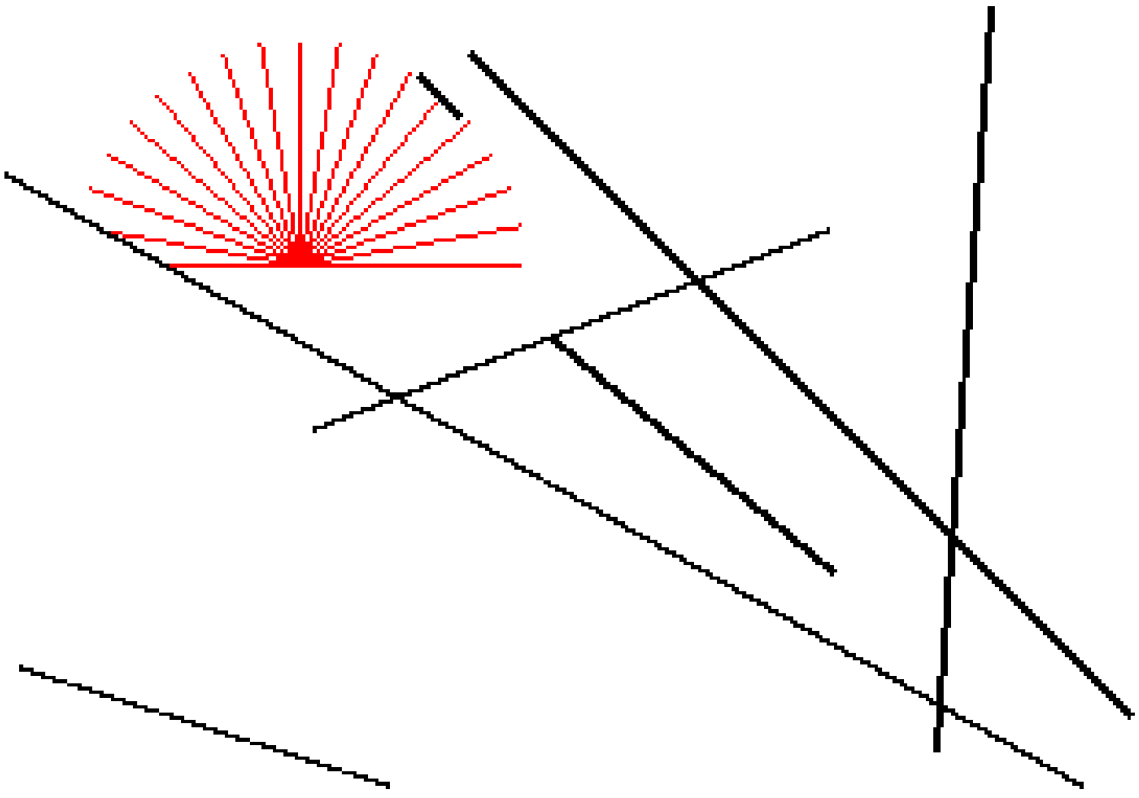
W mojej opinii projekt wykonany jest dobrze, gdyż jego funkcjonalność pokrywa się z założeniami zadania, a kod jest opatrzony dodatkowymi komentarzami w miejscach, które są trudniejsze do zrozumienia (algorytm bresenhama), przy czym jedynymi różnicami są te wynikające z własnej interpretacji.

# Wyniki dla różnych danych wejściowych:

Dla $x = 90$, $y = 90$, $\alpha = 90$

Otrzymujemy:

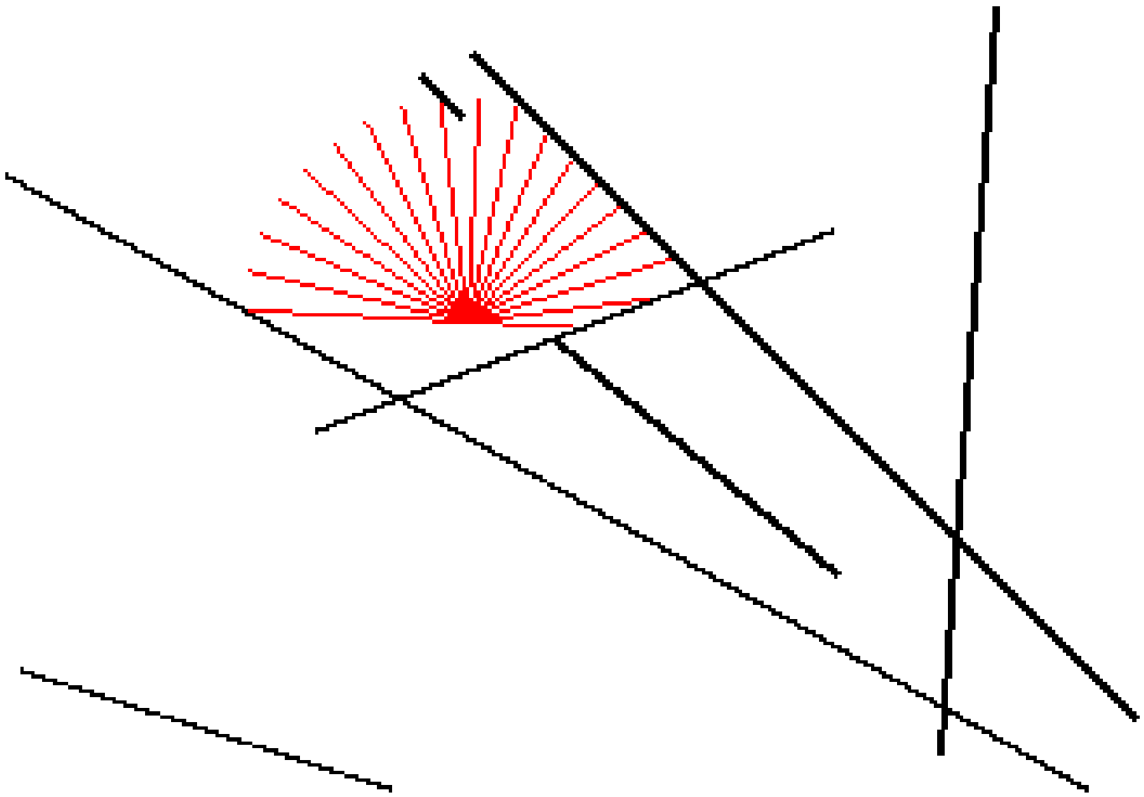
255 255 255 255 255 57 255 255 255 255 255 255 255 255 255 255 255 51 35



Dla x = 135, y = 105, α = 87

Otrzymujemy:

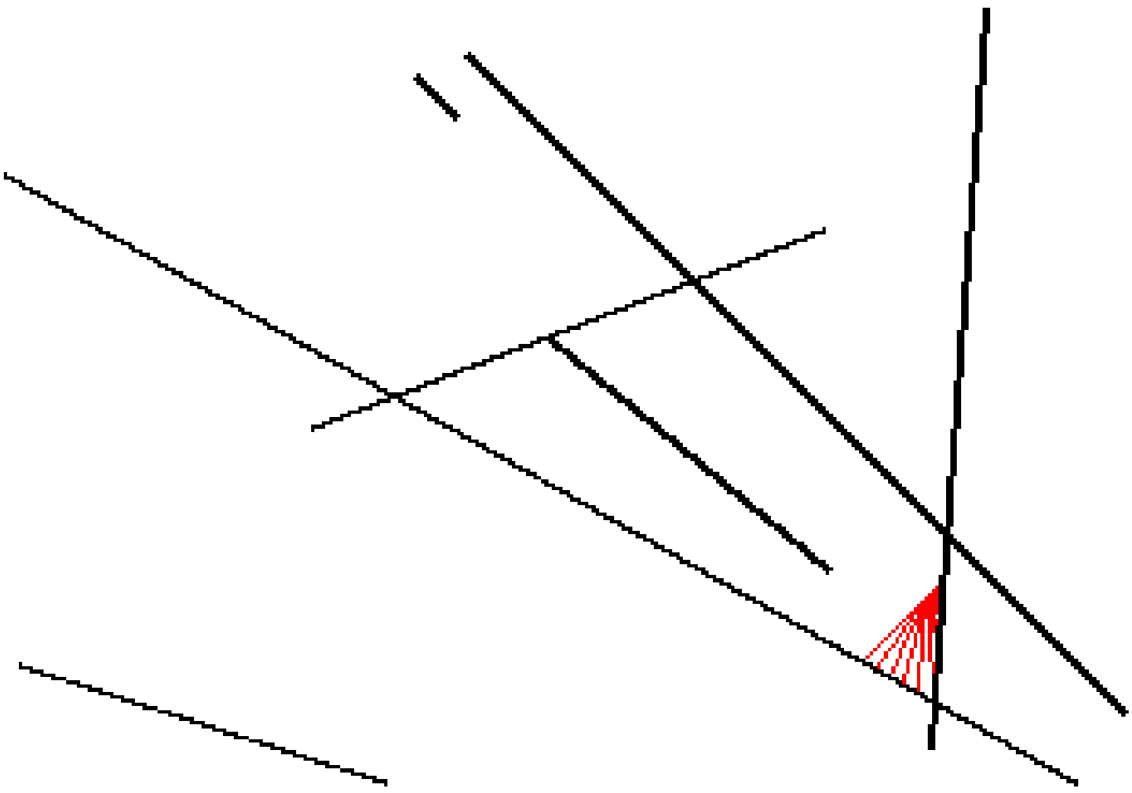
28 49 57 53 51 50 51 54 59 255 59 255 255 255 255 255 255 255 59



Dla x = 262, y = 181, α = 315

Otrzymujemy:

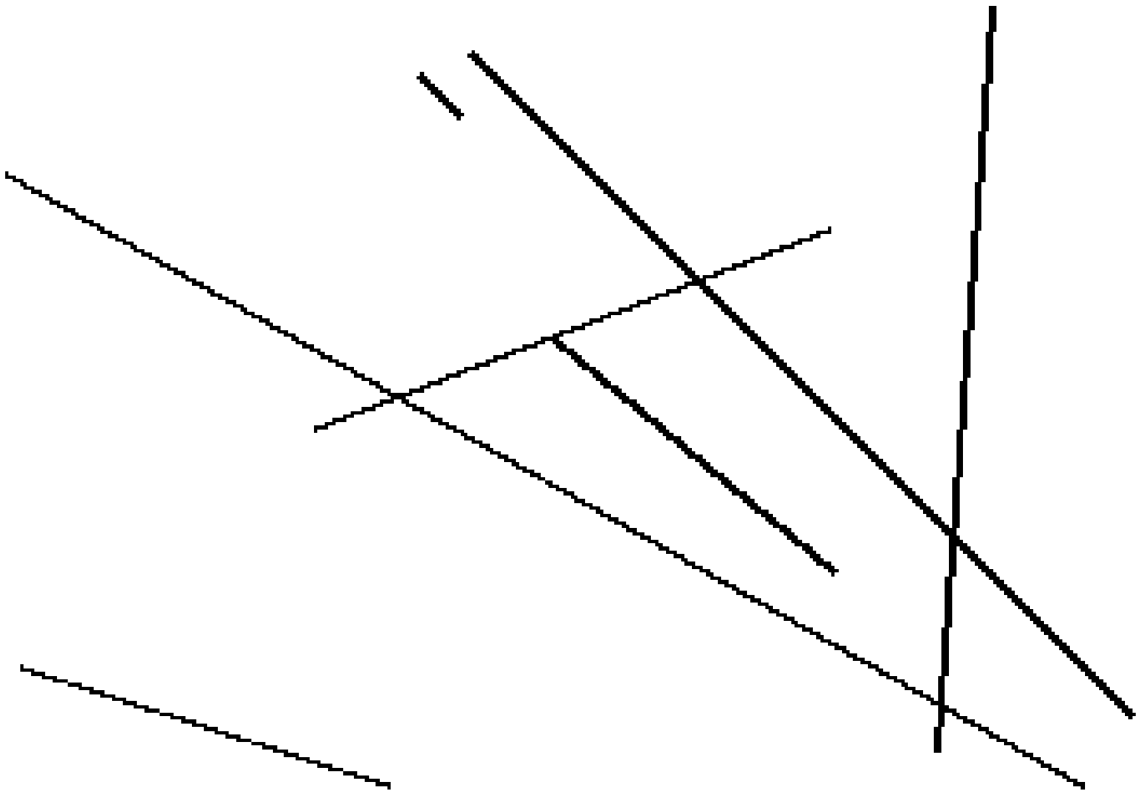
22 23 21 23 25 20 9 7 5 4 3 3 3 3 3 3 3 4



Dla x = 266, y = 181, α = 315

Otrzymujemy:

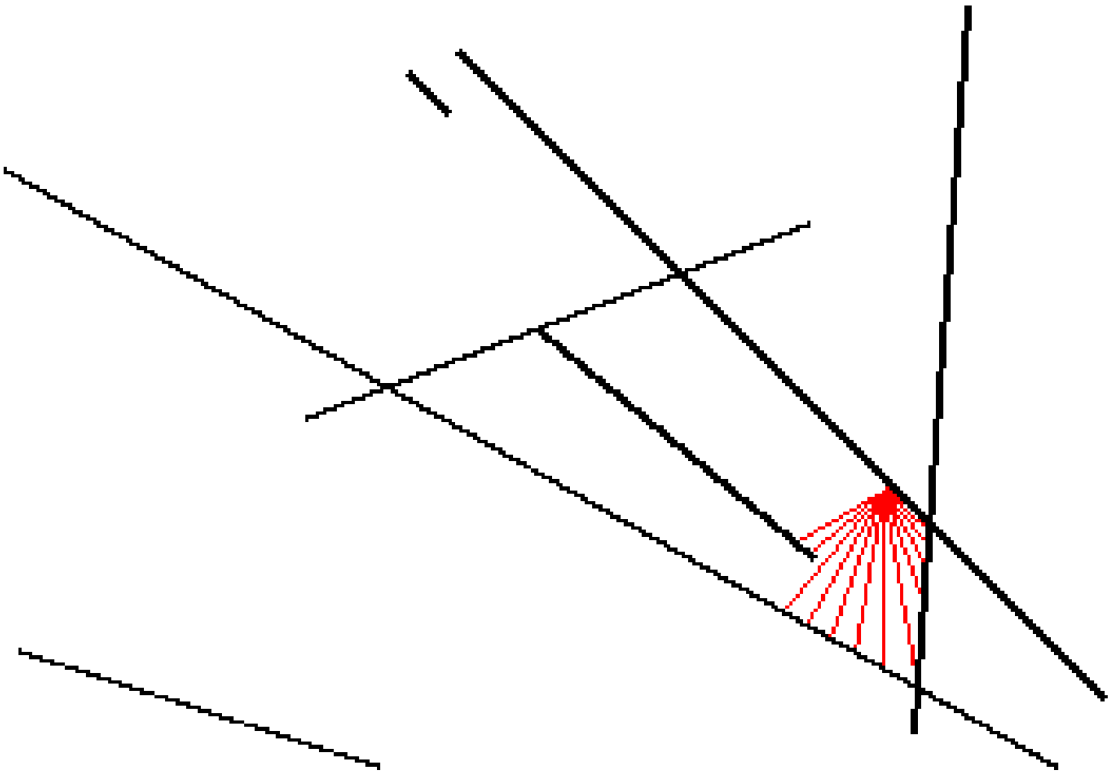
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0



Dla x = 255, y = 155, α = 660

Otrzymujemy:

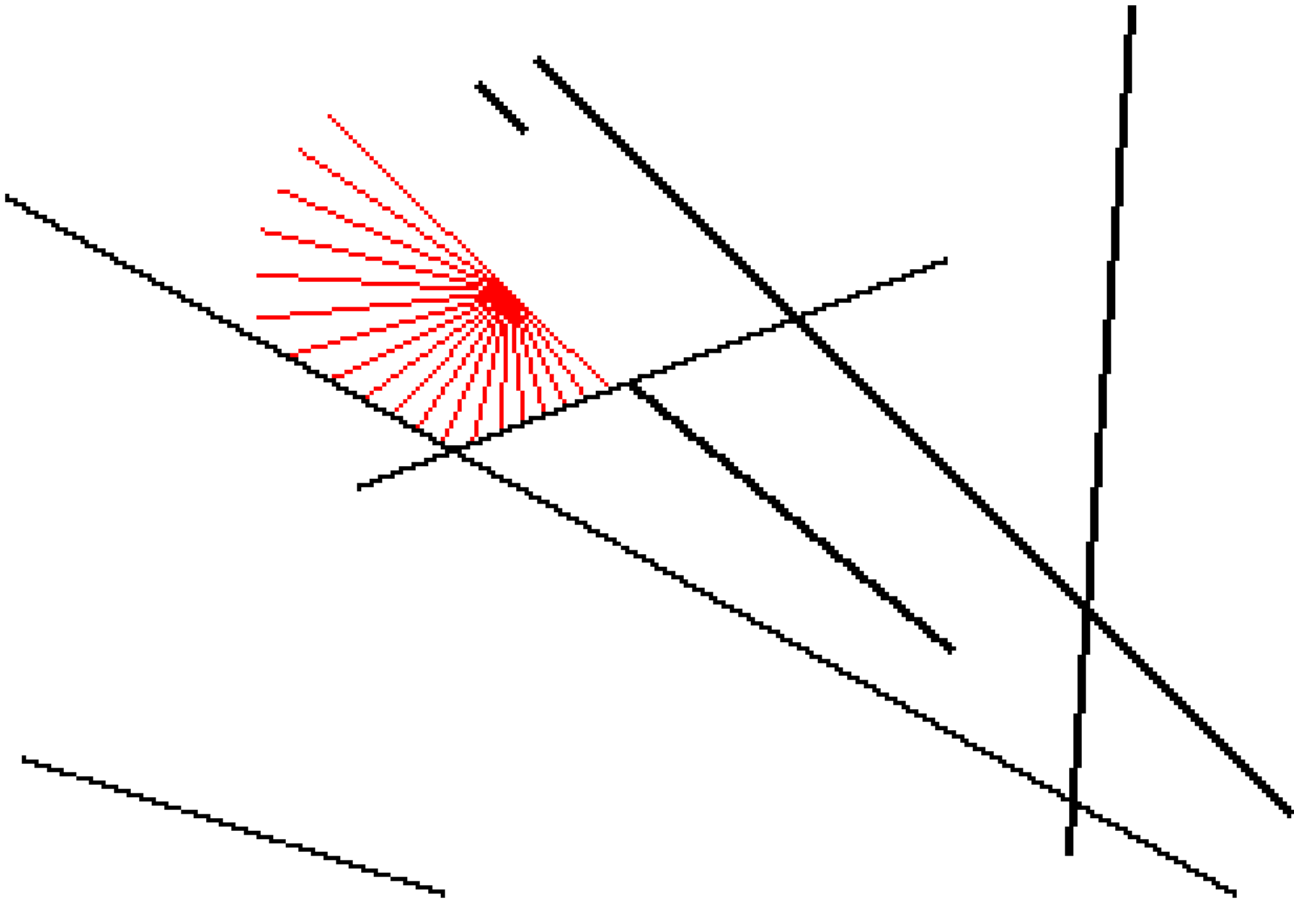
26 24 41 41 42 43 48 48 29 21 17 14 5 3 3 2 2 11



Dla x = 130, y = 88, α = 226

Otrzymujemy:

255 255 255 255 255 255 54 46 42 38 38 38 36 32 30 29 29 29 31



Dla x = 130, y = 88, α = 0 i obrazka „kreski.png” z image_width = 540 i image_height = 320

Otrzymujemy:

255 255 255 255 255 255 255 255 255 255 255 255 56 40 32 28 25 23 23