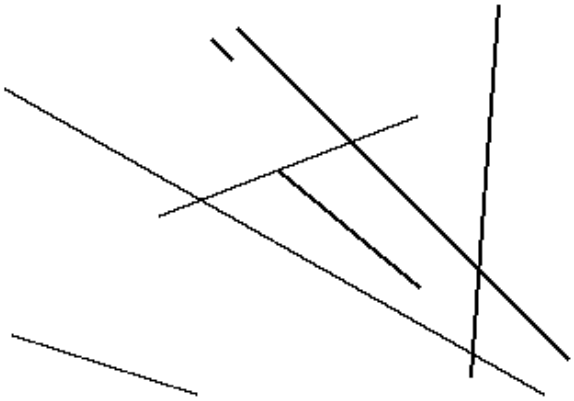
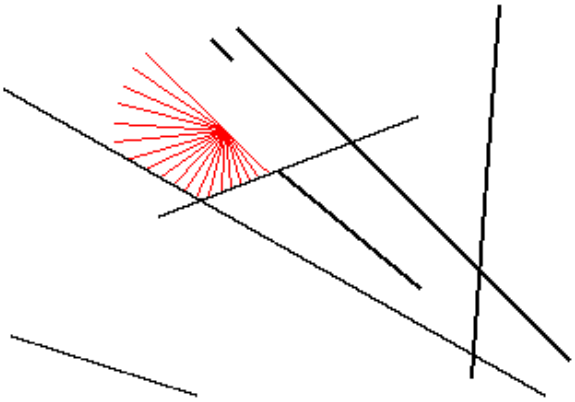
1. CEL I OPIS PROJEKTU

Celem projektu było zrealizowanie symulacji laserowego skanera robota, służącego do pomiaru odległości. Założenie projektowe było takie, że otrzymujemy otoczenie składające się z pustej przestrzeni (białe pole) oraz ścian (kreski w kolorze czarnym) [rys. 1.]. Następnie w pewnym miejscu tego otoczenia stawiamy robota „patrzącego się" w zadanym kierunku. Robot ten wysyła wiązki w zakresie swojego pola widzenia, które zatrzymują się albo na ścianie, albo po przekroczeniu jego zasięgu wzroku [rys. 2.]. Finalnie należało wypisać długość każdej wiązki do pliku tekstowego (w przypadku, gdy wiązka nie trafiła na ścianę miała mieć długość 255).



Rys. 1.



Rys. 2.

2. PODZIAŁ PROGRAMU NA KLASY I INNE PLIKI

Point:
Bardzo prosta klasa służąca do przechowywania współrzędnych x i y w formie punktu oraz obliczania odległości pomiędzy nimi.

Table:
Odpowiada za przechowywanie danych o pikselach w formacie RGB. Jeden piksel to jedna komórka tablicy (macierzy). Klasa ta ma również funkcję rysowania linii z punktu A do punktu B algorytmem Bresenhama.

Img:
Służy do pobrania danych o pikselach ze zdjęcia w formacie .png i stworzenia tablicy klasy Table na ich podstawie, aby można było na nich operować. Finalnie służy również odwrotnemu zabiegowi (wyeksportowanie danych z tablicy klasy Table do pliku .png).

Program:
Główna logika działania programu znajduje się w tej klasie. Klasa Program odpowiada za obliczenie punktów, w których kierunku robot wysyła wiązki oraz następnie narysowanie tych wiązek poprzez wywołanie algorytmu Bresenhama z klasy Table.

Plik interface.py:
Plik zawiera interfejs dołączony do programu i dzieli się na 2 klasy:

GUI – odpowiada za cały interfejs graficzny
ConsoleInterface – odpowiada za cały interfejs konsolowy

Oprócz powyższych klas w programie znajdują się jeszcze:

Plik main.py:
Plik, z poziomu którego uruchamiany jest cały program oraz zawierający mechanizm wyboru interfejsu.

Plik config.py:
Plik zawierający konfigurację działania programu (np. możemy zmienić kolor wiązek lasera robota, jego pole widzenia itd.)

Plik io_data.py:
Plik zawierający funkcje wczytujące oraz zapisujące dane o robocie.

Folder zawierający testy sprawdzające działanie klas i całego programu.

3. INSTRUKCJA UŻYTKOWANIA

Program jest bardzo prosty w obsłudze. Po uruchomieniu terminal spyta się nas o wybór interfejsu. Wpisanie „1" będzie oznaczało interfejs konsolowy, a „2" interfejs graficzny.

Interfejs konsolowy:
Interfejs zapyta nas o nazwę pliku z otoczeniem. Wpisujemy więc np. „otoczenie.png".
Jeśli taki plik istnieje to interfejs zapyta nas teraz o nazwę pliku z danymi robota. Wpisujemy więc np. „parametry.txt"
To tyle!
Jeśli plik z danymi istnieje to program zakończy swe działanie, a równocześnie pojawią się dwa pliki: „wyniki.txt" zawierający długości poszczególnych wiązek lasera robota, a także plik „symulacja.png", który obrazuje działanie programu jak na rys. 2. na poprzedniej stronie.

(UWAGA: Ważne, aby plik otoczeniem oraz plik z danymi znajdował się w tym samym folderze, z którego wykonywany jest program)

Interfejs graficzny:
Na początku ukaże nam się okno wyboru pliku. Musimy wybrać plik o formacie .png, który jest naszym otoczeniem. Po wyborze pliku, otoczenie otworzy się w programie. Teraz mamy możliwość by w dowolnym miejscu, które wybierzemy myszką, postawić robota LPM. Stawiając robota wyświetli się okno do wpisania kąta o jaki obrócony jest robot w stopniach (orientacyjnie: 0 – w górę, 90 – w lewo, 180 – w dół, 270 – w prawo). Robot we wskazanym miejscu od razu wyśle wiązki we wskazanym kierunku, a dane wynikowe oraz symulacja zapiszą się do pliku. Proces ten możemy powtarzać do znudzenia.

4. REFLEKSJE KOŃCOWE

Projekt udało się zrealizować dość sprawnie, a największą przeszkodą była odpowiednia dekompozycja problemu, do której nadal mam pewne wątpliwości, np. klasy Table oraz Img są bardzo podobne w zamyśle – możliwe, że optymalniej byłoby zrealizować ich zadania w formie jednej klasy, a działania Table zastąpić biblioteką NumPy. Łatwiej jednak było mi wyobrazić sobie wartości pikseli w zwykłej tabeli.
Kolejną trudnością było narysowanie linii (wiązek lasera). Algorytm Bresenhama zaimplementowany przeze mnie rysuje linie od punktu A do punktu B, a nie w zadanym kierunku (kącie). Koniecznym więc było:

a) Zmienić zaimplementowany algorytm, by obliczał następne piksele do pokolorowania na podstawie kąta.
b) Obliczyć punkt do jakiego ma dążyć wiązka i użyć tego punktu jako punkt B w algorytmie Bresenhama.

Zdecydowałem się na rozwiązanie b), ponieważ stwierdziłem, że będzie po prostu łatwiejsze. W efekcie powstała klasa Program obliczająca ten punkt oraz wywołująca algorytm Bresenhama z obliczonym punktem jako argument.
Zastanawiałem się też nad możliwością nazywania plików eksportowanych z programu, ale pominąłem to zagadnienie i zaimplementowałem bazowe nazwy, ponieważ chciałem, żeby program był dość prosty w obsłudze, a dodając coraz to kolejne wyskakujące okienka z kolejnymi danymi do wprowadzenia, program wydawałby się coraz to bardziej skomplikowany.
Reszta operacji (takie jak np. konwersja pikseli na faktyczne wartości RGB) wydawała się w miarę prosta do zaimplementowania i tak się okazało.

Podsumowując, myślę, że program sam w sobie działa dość poprawnie i jest pozbawiony bardziej oczywistych błędów. Uważam, że najlepszą jego funkcją interfejs graficzny, a dokładniej możliwość wyboru myszką punktu, w którym chcemy postawić robota i która przyniosła mi wiele satysfakcji, kiedy okazało się, że działa.