

Dokumentacja projektu

Analiza i przetwarzanie obrazów
05 lipca 2023

Cele i założenia projektu

Celem projektu było wykonanie programu pozwalającego na wczytanie mapy, na której drogi są jaśniejsze niż tło i wybrać na niej dwa punkty. Narzędzie ma wyznaczyć najkrótszą drogę między zaznaczonymi punktami. Prędkość z jaką można pokonać dany odcinek trasy jest proporcjonalna do szerokości tego odcinka (dwa razy szersza droga jest pokonywana dwa razy szybciej).

Opis zrealizowanego rozwiązania

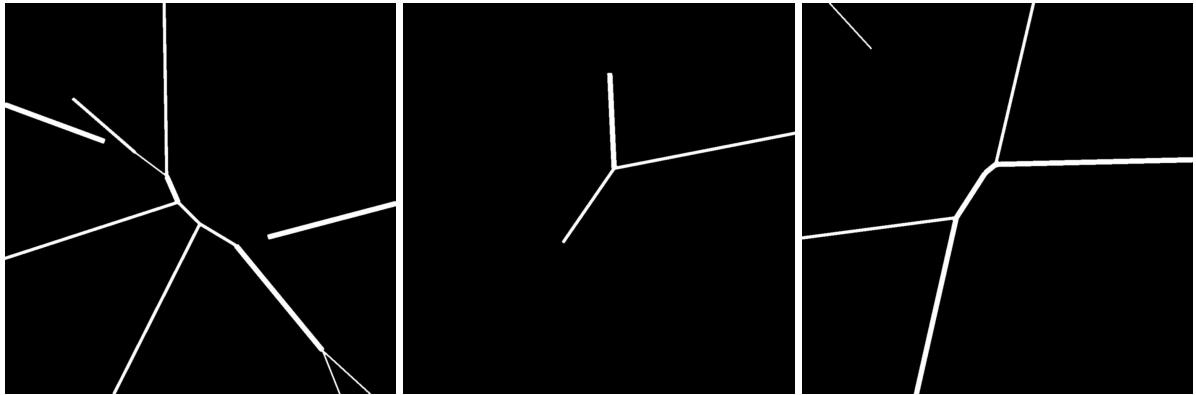
Zrealizowane rozwiązanie działa również na prawdziwych danych (np. zdjęcia satelitarne, jeśli drogi są jaśniejsze niż pozostała część obrazka). Z tego powodu w folderze repozytorium znajdują się nie tylko zdjęcia wygenerowane sztucznie.

Generowanie danych

Nawiązując do wytycznych projektu, dane wejściowe będą składać się z obrazów map, na których "drogi są jaśniejsze od tła" - bazując na tym stwierdzeniu przyjęliśmy założenie, że odpowiednio skonfigurowany blok preprocessingu będzie w stanie doprowadzić te dane do postaci obrazów binarnych, z czarnym tłem i białymi drogami. Na tej podstawie zdecydowaliśmy, że sztuczne dane treningowe będą się składały właśnie z binarnych zdjęć z czarnym tłem i białymi drogami. W celu zróżnicowania generowanych danych, wykorzystujemy w tym celu dwa generatory. Oba generatory zostały zaimplementowane w języku python.

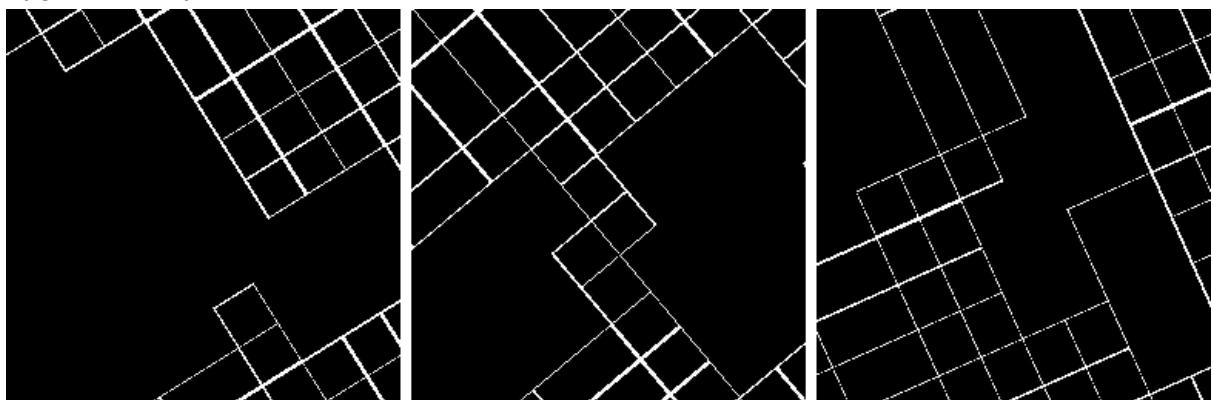
Generator Woronoja

Generator wykorzystuje algorytm do tworzenia diagramów Woronoja. Służy on do wykreowania obrazów ukazujących drogi wiejskie - o mniejszej gęstości i większej nieregularności. Często się zdarza, że ścieżki na obszarach mniej zaludnionych niespodziewanie się urywają, czy łączą się pod bardzo ostrymi kątami, dlatego użyty algorytm musiał zostać urozmaicony, żeby takie funkcjonalności uwzględnić. Zbiór treningowy składa się z 1000 obrazów wylosowanych dla ziarna 0.



Generator cityblock

Generator tworzy obrazy przypominające układ dróg gęsto zaludnionych terenów, w których drogi przecinają się pod kątem prostym. Ilość, szerokość i odległość między drogami są losowane, a dodatkowo po narysowaniu dróg na obrazie część z nich jest losowo usuwana. W zbiorze treningowym znajduje się 1000 obrazów pochodzących z tego generatora, wygenerowanych dla ziarna 0.



Szczegółowy opis algorytmu

Algorytm ma następujący schemat:

1. Preprocessing
 - a. Segmentacja
 - b. Binaryzacja
 - c. Morfologiczne zamknięcie
 - d. Filtr bilateralny
 - e. Szkieletyzacja
 - f. Usuwanie drobnych gałęzi
2. Budowanie grafu
 - a. Szukanie wierzchołków
 - b. Usuwanie nadmiarowych wierzchołków
 - c. Kolorowanie ścieżek
 - d. Zalewanie ścieżek i obliczanie wag
3. Szukanie najszybszej ścieżki
 - a. Algorytm Dijkstry

Preprocessing

Segmentacja

Segmentacja odbywa się za pomocą klasteryzacji k-means z następującymi parametrami:

1. Ilość klastrów (K): 3
2. Kryteria zatrzymania (criteria): max iteracji 100, eps 0.2
3. Ilość prób (attempts): 10
4. Losowe środki startowe (KMEANS_RANDOM_CENTERS)

Dodatkowo obraz po segmentacji jest poddawany rozmyciu Gaussa o rozmiarze macierzy jądra 3x3.

Binaryzacja

Obraz poddawany jest binaryzacji z progiem wykrywanym metodą Otsu.

Morfologiczne zamknięcie

W celu wygładzenia postrzępionych krawędzi ścieżek, stosowana jest operacja morfologicznego zamknięcia, z macierzą jądra o rozmiarze 3x3.

Filtr bilateralny

W celu pozbycia się wystających gałęzi podczas szkieletyzacji ścieżek z nierównymi krawędziami, obraz jest najpierw poddawany filtrowaniu. Wybrany został filtr bilateral ze względum na właściwości wygładzające, z zachowaniem ogólnych konturów obiektu. Użyliśmy rozmiaru filtra 20 oraz wartości obu sigm 75.

Szkieletyzacja

Do szkieletyzacji został użyty algorytm morfologicznego odchudzania (thin), który tworzył najmniejszą ilość błędnych gałęzi dzielących jedną ścieżkę.

Usuwanie drobnych gałęzi

Podczas szkieletyzacji na obrazie powstaje dużo mikroskopijnych gałęzi związanych z niedoskonałościami obrazu. Ponieważ w dość oczywisty sposób użytkownik nie będzie oczekiwał tak małych, odseparowanych od reszty ścieżek, napisany został algorytm do ich usunięcia:

1. Szukanie izolowanych elementów (cv2.connectedComponentsWithStats)
2. Obliczanie długości każdego elementu
3. Sortowanie elementów według długości
4. Tworzenie maski z elementami o długości mniejszej lub równej 5% najdłuższego elementu
5. Usuwanie z obrazka elementów za pomocą maski

Budowanie grafu

Szukanie wierzchołków

Wyszukiwanie wierzchołków odbywa się na podstawie sprawdzania sąsiadów każdego z pikseli – jeśli piksel otoczony jest dokładnie dwoma wypełnionymi pikselami (oznaczającymi ścieżkę) to sam należy do ścieżki i oznaczany jest kolorem niebieskim (255, 0, 0) (obraz przechowywany jest w formacie BGR), w przeciwnym razie jest wierzchołkiem, oznaczanym kolorem zielonym (0, 255, 0).

Usuwanie nadmiarowych wierzchołków

W pewnych kombinacjach pikseli powstają klastry trzech sąsiadujących wierzchołków. Na tym etapie są one usuwane – środkowy piksel pozostaje wierzchołkiem. Na końcu wszystkie wierzchołki są dodawane do grafu.

Kolorowanie ścieżek

W celu odseparowania ścieżek między wierzchołkami, oznaczane są one kolejnymi kolorami BGR (odejmując zarezerowane kolory, daje to $255 \times 255 \times 255 - 1 = 16581374$ możliwych ścieżek na obrazie, kilka rzędów wielkości więcej niż jest to realistycznie wymagane). Przy okazji do krawędzi dodawane są punkty pośrednio, co umożliwia rysowanie na mapie krawędzi bardziej dopasowanych do ścieżek. Do kolorowania stosowany jest następujący algorytm (pomijając obsługę krawędzi obrazu):

1. Dla każdego z wierzchołków, dla każdego z ośmiu kierunków:
 - a. Ustaw wskaźnik na startowy wierzchołek
 - b. Dodaj startowy wierzchołek jako pierwszy punkt pośredni
 - c. Wykonuj dopóki możliwe:
 - i. Jeśli wskazywany piksel jest niebieski (ścieżka):
 1. Pomaluj go na dany kolor
 2. Dodaj 1 do długości ścieżki
 3. Co dziesiąty piksel dodaj punkt pośredni
 - ii. Jeśli wskazywany piksel jest zielony (wierzchołek):
 1. Zakończ ścieżkę
 2. Zapisz krawędź do grafu
 - iii. W pozostałych przypadkach odrzuć ścieżkę
 - iv. Przeszukaj otaczające piksele w poszukiwaniu wierzchołka, jeśli znaleziono:
 1. Zakończ ścieżkę
 2. Zapisz krawędź do grafu
 - v. Przeszukaj otaczające piksele w poszukiwaniu kontynuacji ścieżki, jeśli znaleziono:
 1. Przesuń wskaźnik na nowy piksel

Zalewanie ścieżek i obliczanie wag

W celu obliczenia wag dla krawędzi grafu, obliczana jest łączna liczba pikseli w każdej ścieżce. W tym celu obraz binarny (przed szkieletyzacją) jest “zalewany” kolorami wyznaczonymi w poprzednim punkcie.

W każdym kroku pętli, piksele zawierające kolory (nie-czarne i nie-wierzchołki) kopią swój kolor do sąsiadów. Pętla kończy się gdy w danej iteracji nie zaszły żadne zmiany. W celu optymalizacji, algorytm nie iteruje po całym obrazie, a jedynie po "aktywnych pikselach" – w pierwszej iteracji po wszystkich pikselach należących do ścieżek, w kolejnych po pikselach, które zostały ostatnio pomalowane.

Ostatecznie waga obliczana jest jako iloraz kwadratu długości i liczby pikseli w ścieżce.

Szukanie najszybszej ścieżki

Algorytm Dijkstry

Po stworzeniu struktury grafu zostaje on naniesiony na obrazek. Kolejnym krokiem jest wyznaczenie najkrótszej ścieżki na podstawie wybranych punktów za pomocą algorytmu Dijkstry. Następnie znaleziona ścieżka jest nanoszona na obraz.

Instrukcja obsługi aplikacji

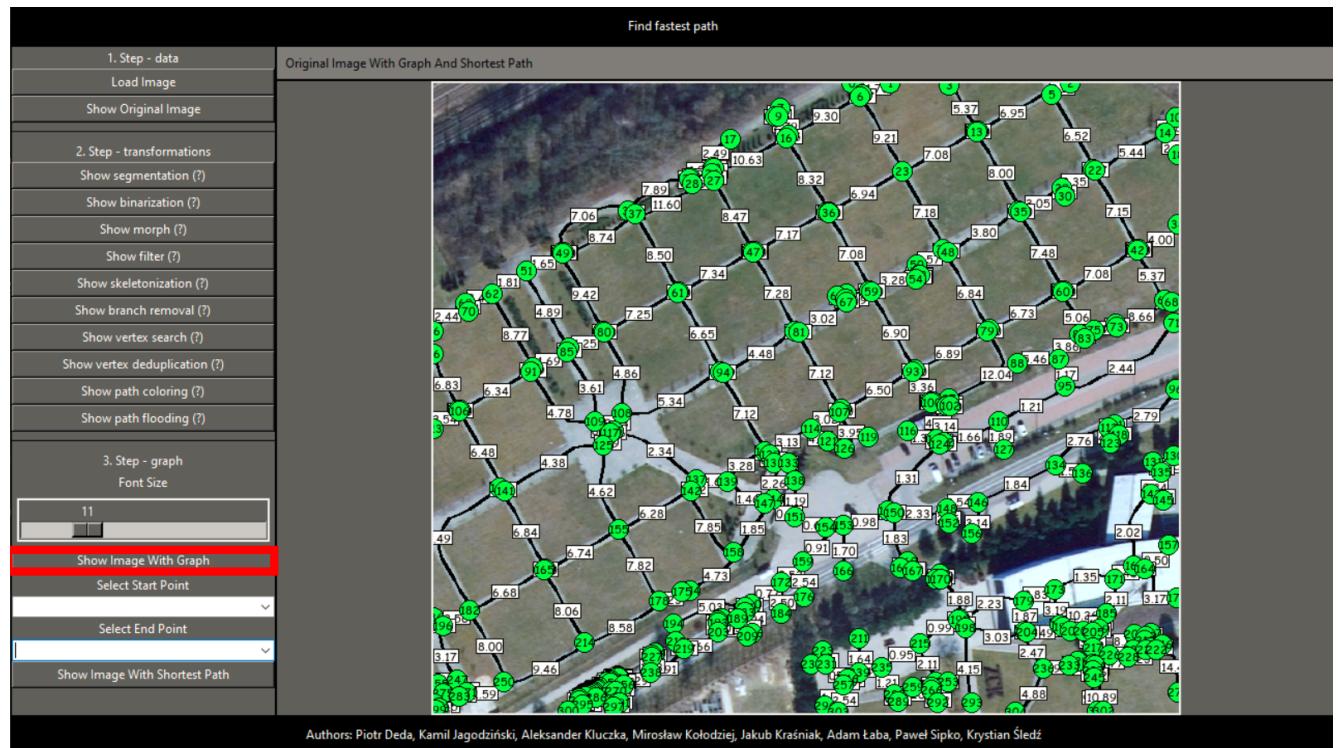
- Załaduj zdjęcie:** Wybierz zdjęcie w formacie JPG lub PNG, naciskając przycisk "Load Image".



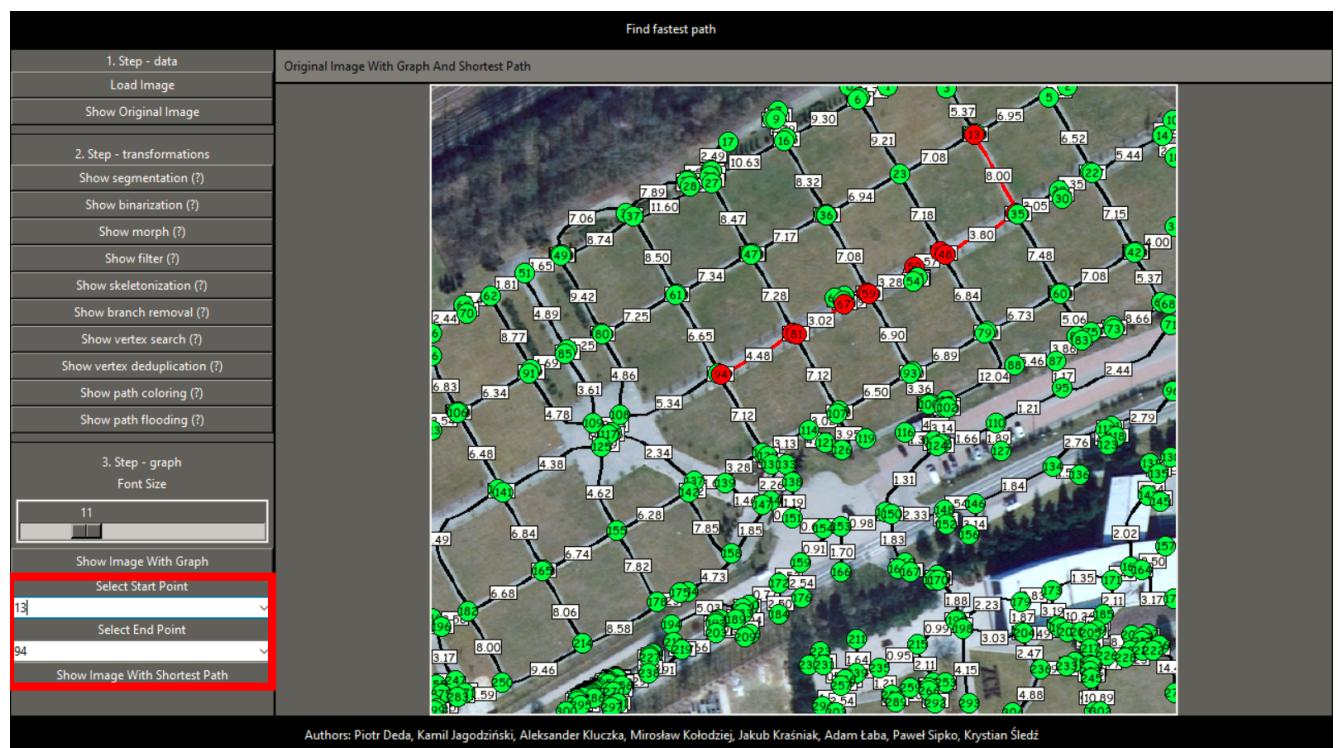
The screenshot shows the application's interface after loading an image. On the left is a sidebar with the same menu structure as the first screenshot. In the center is the "Original image" view, which displays an aerial photograph of a park or industrial area with a grid of paths and some buildings. At the top of this view is the text "Find fastest path". On the right side of the image, there are controls for pathfinding: "Select Start Point" (with a dropdown arrow), "Select End Point" (with a dropdown arrow), and "Show Image With Shortest Path" (which is currently selected, indicated by a blue border).

Authors: Piotr Deda, Kamil Jagodziński, Aleksander Kluczka, Mirosław Kołodziej, Jakub Kraśnian, Adam Łaba, Paweł Sipko, Krystian Śledź

2. **Pokaż zdjęcie z grafem:** Wyświetl swoje zdjęcie z naniesionym na nie grafem, naciskając przycisk "Show Image With Graph".

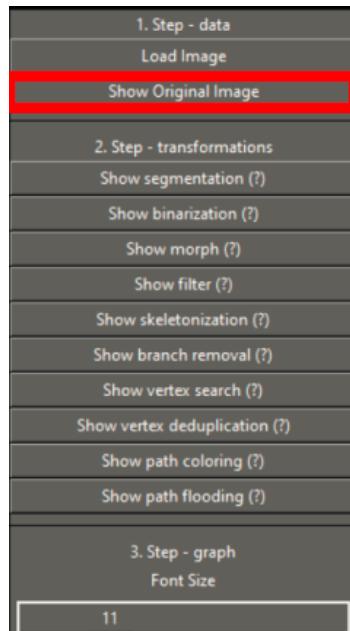


3. **Wybierz punkty początkowy i końcowy:** Użyj rozwijanych menu, aby wybrać punkty początkowy i końcowy na grafie oraz **pokaż zdjęcie z najkrótszą ścieżką:** Wyświetl najkrótszą ścieżkę naniesioną na graf, naciskając przycisk "Show Image With Shortest Path".

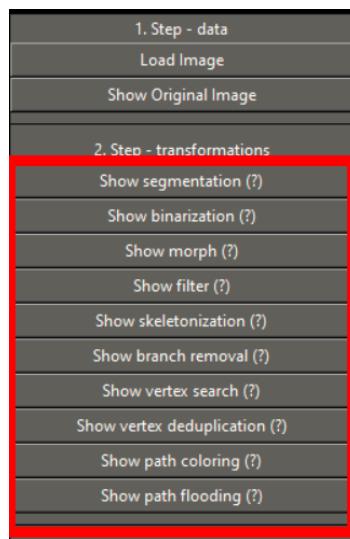


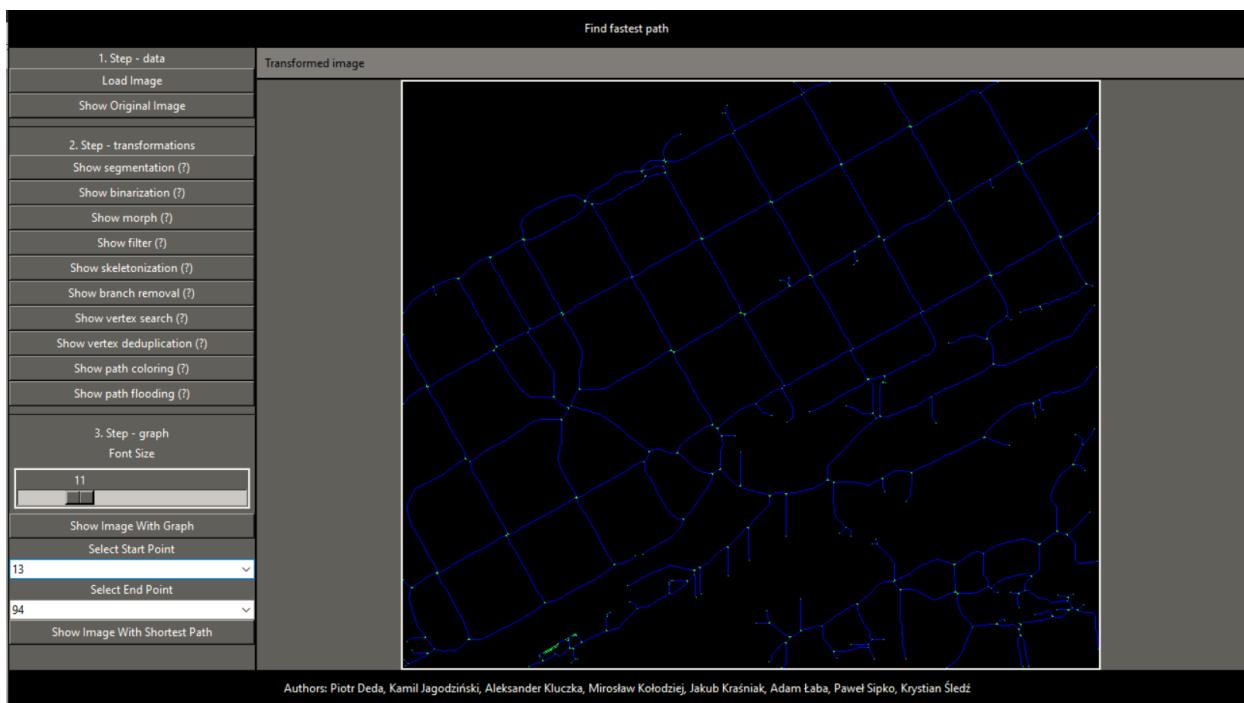
Dodatkowe Funkcje:

- **Pokaż oryginalne zdjęcie:** Jeżeli chcesz zobaczyć, jak wygląda oryginalne zdjęcie, naciśnij przycisk "Show Original Image".

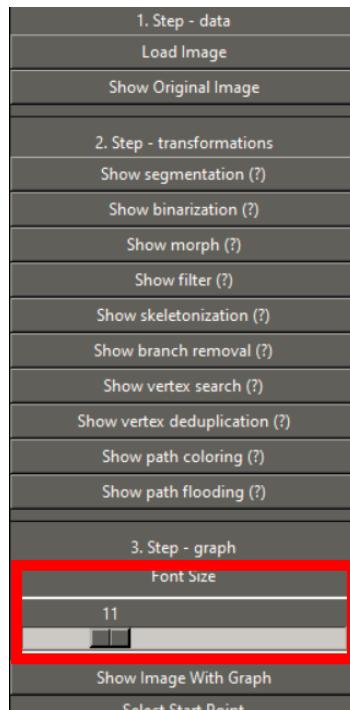


- **Zobacz działanie funkcji transformujących:** Możesz podejrzeć wpływ poszczególnych funkcji transformujących na obrazek, klikając przycisk interesującej Cię funkcji.





- **Zmiana wielkości czcionki:** Ta funkcja pozwala zmienić wielkość czcionki, co wpływa na wielkość generowanego grafu i przestrzeń, którą zajmuje na zdjęciu. Jest to szczególnie przydatne w przypadku zdjęć z wieloma skrzyżowaniami.



Wykorzystane biblioteki

- matplotlib
- Scipy
- numpy
- opencv-python
- scikit-image
- Pillow
- Tkinter

Podział ról w grupie

Generowanie danych wejściowych

Adam Łaba

Aleksander Kluczka

Preprocessing + budowanie grafu

Kamil Jagodziński - preprocessing

Jakub Kraśniak - preprocessing

Piotr Deda - budowanie grafu

Szukanie najszybszej ścieżki + UI

Krystian Śledź - UI

Mirosław Kołodziej - rysowanie grafu + algorytm Dijkstry

Paweł Sipko - rysowanie grafu + algorytm Dijkstry

Źródła

- Program z algorytmem najkrótszej ścieżki:
<https://github.com/PiotrDeda/shortest-path-detection>
- Skrypt do generowania obrazów: https://github.com/vis4rd/iap_project_2023
- Wygenerowany zbiór obrazów (dostęp przez konto uczelniane):
https://aghedupl-my.sharepoint.com/:f/g/personal/qlootchka_student_aqh_edu_pl/Eia_bjSF5IJGUp7yh_L7c0EBJY_KodAZxSBAx0_jVscoTg?e=KBhWol