Tłumacz alfabetu Braille’a

Zuzanna Adamiuk i Piotr Kielak

Wstęp i cele

Aplikacja umożliwia tłumaczenie treści zapisanej angielskim alfabetem Braille’a na tradycyjny alfabet łaciński. Oryginalny tekst jest wprowadzany w formie obrazu, a tłumaczenie jest wyświetlane w oknie oraz, po zaznaczeniu, zapisywane w pliku tekstowym.

Założenia ogólne

Aplikacja została napisana w języku Java z wykorzystaniem biblioteki OpenCV 4.5.0 umożliwiającej przetwarzanie obrazów. Umożliwia:

- wczytywanie obrazów w formacie .jpg/.png z interfejsu graficznego
- tłumaczenie tekstu zapisanego angielskim alfabetem Braille’a
- zapisywanie wyników do pliku .txt

Cele jakościowe

Za cel postawione zostało osiągnięcie jakości która pozwoli na bezawaryjne działanie programu w taki sposób, aby dostęp i użytkowanie było możliwie jak najwygodniejsze dla użytkownika.

Projekt aplikacji

Program jest aplikacją z interfejsem graficznym działająca w kilku krokach, które można podzielić na:

- wczytanie obrazu przez użytkownika
- przetworzenie obrazu (część wykonywana jest automatycznie przez wykorzystaną metodę HoughCircles)
- tłumaczenie na podstawie danych z obrazu
- wyświetlenie oraz zapisanie przetłumaczonego tekstu w pliku .txt

Biblioteki

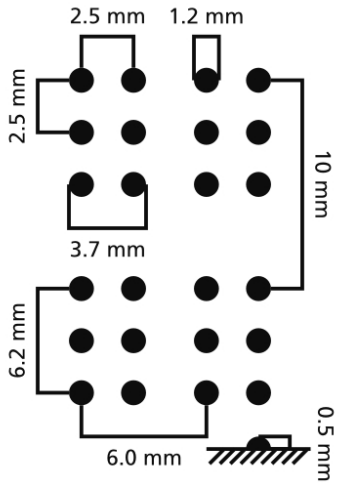
Do stworzenia programu zostały użyte dwie biblioteki. Pierwszą jest java swing która jest rozwinięciem starszej biblioteki ATW. Została ona użyta do stworzenia interfejsu graficznego. Jest to najstabilniejsza i najszerzej używana biblioteka która zapewnia „lekkie" komponenty tworzące interfejs graficzny.

Drugą wykorzystaną biblioteką jest OpenCV 4.5.0 która umożliwia przetwarzanie obrazów w czasie rzeczywistym. Jest ona wieloplatformowa oraz darmowa na licencji typu open-source.

Wymagania

Do korzystania z aplikacji potrzebny jest obraz spełniający następujące warunki:

- tekst musi być dobrze widoczny (zdjęcie musi być ostre)
- w kadrze może znajdować się jedynie tekst
- tekst musi być równy względem krawędzi obrazu
- tekst musi spełniać normy przewidziane dla tekstu w języku Braille'a (zgodność ze schematem z Rysunku 5)



Rysunek 5. Schemat proporcji wykorzystanych do stworzenia algorytmu tłumaczącego

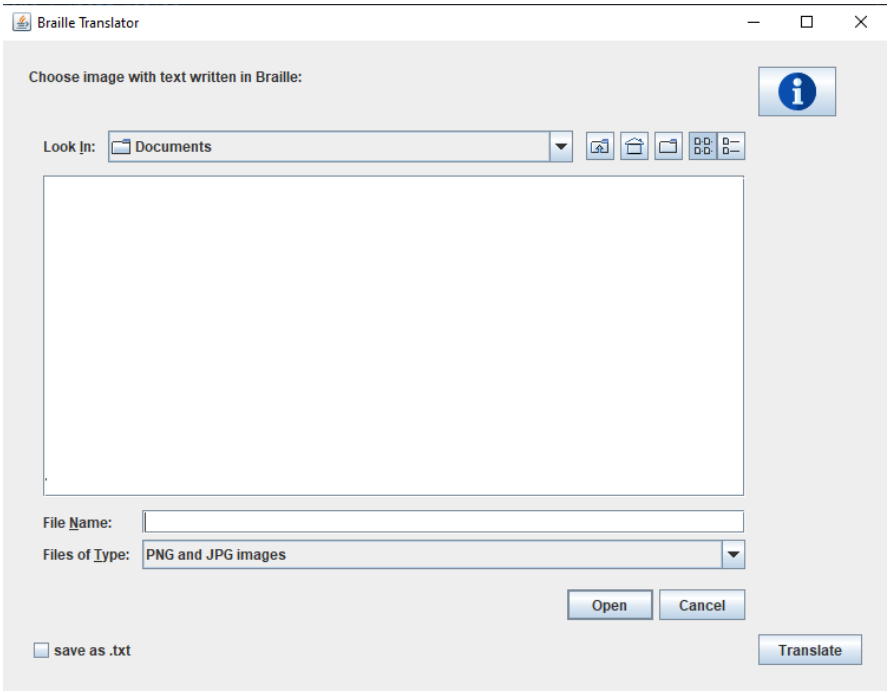
Klasy

Nazwa klasy	Opis	Komentarz
mainBraille	Główna klasa aplikacji	Łączy GUI z funkcjami programu
App_interface	Klasa przechowująca informacje o GUI	Interfejs graficzny umożliwiający korzystanie z aplikacji

Letters	Klasa przechowująca słownik alfabetu Braille'a	Zawiera 2 tablice – ze znakami w alfabecie Braille'a oraz z odpowiadającymi im tradycyjnymi literami

Interfejs graficzny

Interfejs graficzny został napisany przy użyciu biblioteki Swing. Kierując się wcześniejszymi założeniami aplikacji została zaprojektowana w prosty i możliwie najwygodniejszy dla użytkownika sposób, stąd na głównym ekranie pojawiły się tylko 4 elementy, z którymi użytkownik może wejść w interakcje.

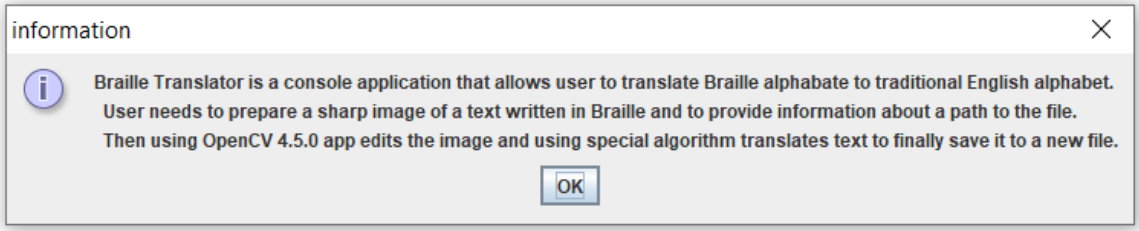


Rysunek 2. Interfejs użytkownika

Program został przygotowany na wszelkiego typu błędy, o których użytkownik jest informowany - przycisk „Translate" wyświetli stosowny komunikat jeżeli użytkownik użyje go przed wybraniem pliku z kolei sam plik może być tylko typu jpg lub png.

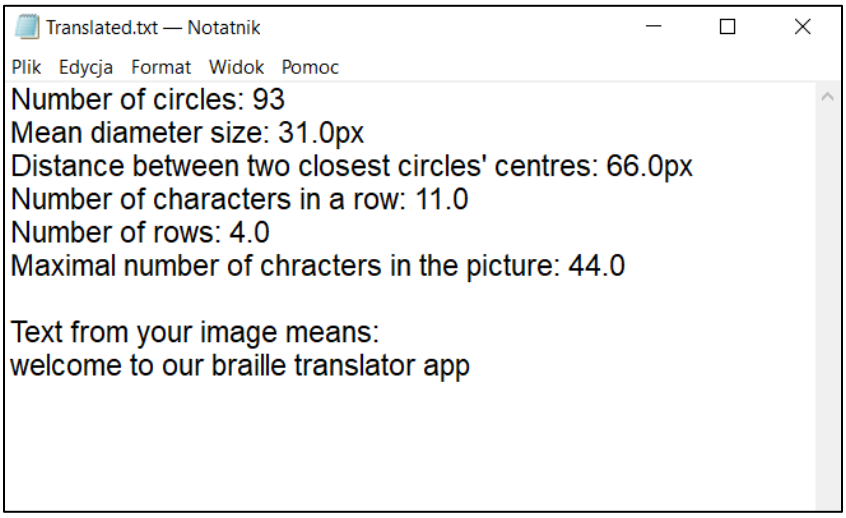
Dodatkową funkcjonalnością, która została dodana jest możliwość wyboru czy program ma zapisać przetłumaczony tekst do pliku o rozszerzeniu txt – wyboru dokonuje się poprzez zaznaczenie pola przy „save as .txt", następnie należy wybrać opcję „Translate".

W celu wyświetlenia dodatkowych informacji na temat aplikacji, należy wybrać przycisk info.

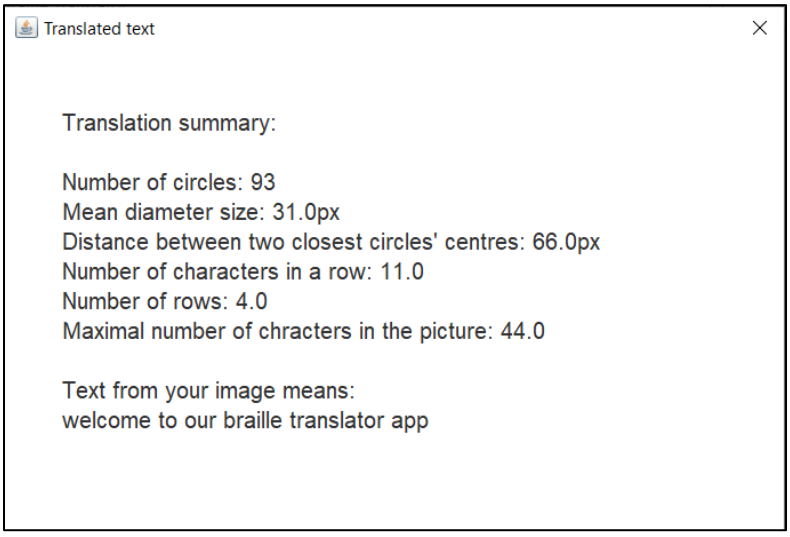


Rysunek 3. Podgląd informacji dot. aplikacji

Ostatecznie, po przetłumaczeniu, pojawi się nowe okno z gotowym podsumowaniem przeprowadzonego tłumaczenia, które można edytować. Zawartość poniższego pola tekstowego (Rysunek 4) zostaję wyeksportowana do pliku .txt na życzenie użytkownika (Rysunek 5).



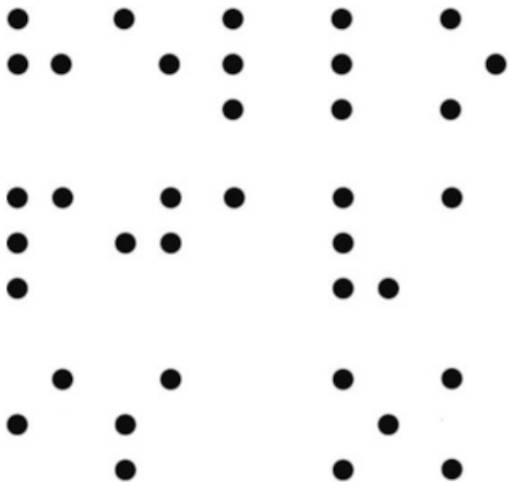
Rysunek 4. Podsumowanie wykonanych operacji w oknie aplikacji



Rysunek 5. Podgląd zapisanego pliku .txt

Przetwarzanie obrazu

Do przetworzenia obrazu użyto biblioteki Open CV 4.5.0, która zapewnia takie funkcje jak przekształcenie macierzy obrazu na skalę szarości, redukcje szumów, wyodrębnienie krawędzi oraz binaryzacje obrazu. Po wykonaniu przez program tych czynności, otrzymujemy obraz w dużo bardziej czytelny, a to z kolei zapewnia, że program nie odczyta błędnie któregokolwiek ze znaków.



Rysunek 6. Przykładowy przetworzony obraz

Tłumaczenie tekstu Braille'a

Do finalnego tłumaczenia tekstu wykorzystana została metoda HoughCircle z biblioteki OpenCV 4.5.0. Sama funkcja korzysta z „circle hough transform" która jest podstawową techniką wykorzystywaną w cyfrowym przetwarzaniu obrazu do wykrywania okręgów. HoughCircle ma zaimplementowaną obróbkę obrazu (redukcja szumu, wykrywanie krawędzi), dlatego w kodzie oddzielnie została tylko zaimplementowana zmiana na skalę szarości (tego nie ma zaimplementowanego wspomniana metoda). Po obróbce funkcja wykorzystuje kołową transformatę Hough'a, aby określić, gdzie występują okręgi. Wiedząc, gdzie znajdują się środki okręgów, program określa ich położenie w przestrzeni, wielkość oraz umożliwia obliczenie odległości pomiędzy najbliższymi sobie okręgami, która jest głównym parametrem będącym punktem odniesienia dla dalszych obliczeń. Na podstawie najmniejszej odległości między środkami okręgów określana jest ilość rzędów oraz wierszy, a następnie punkty ustawiane są w „szóstki", ponieważ każdy znak w języku Braille'a składa się z maksymalnie 6 punktów. Znając pełne znaki, są one porównywane z bazą znaków z klasy Dictionary, po czym każdemu znakowi zostaje przypisana litera z alfabetu łacińskiego.

Algorytm tłumaczący

```
public static  String letter_translation_by_minDist(double w, double h, Mat imgEdited, Mat image) throws
IOException{
        […] – fragment kodu został pominięty
        for (int currRow = 1; currRow < numberOfRows+1; currRow++) {
                    for (int currCol = 1; currCol < numberOfColumns+1; currCol++){

                        for (int i = 0; i < 6; i++){
                                currCharacter[i] = "0";
                        }

                        for (int j = 0; j < circleCounter; j++){

                            // Algorithm translating a single character
                            if (pointArray[j].x < (currCol-1)*oneCharacterWithSpace + minDist &&
                                pointArray[j].x > (currCol-1)*oneCharacterWithSpace &&
                                pointArray[j].y < (currRow-1)*oneCharacterWithNewLine + minDist &&
                                pointArray[j].y > (currRow-1)*oneCharacterWithNewLine){
                                currCharacter[0] = "1";
                            }
                            else if (pointArray[j].x < currCol*oneCharacterWithSpace &&
                                     pointArray[j].x > ((currCol-1)*oneCharacterWithSpace) + minDist &&
                                     pointArray[j].y < (currRow-1)*oneCharacterWithNewLine + minDist &&
                                     pointArray[j].y > (currRow-1)*oneCharacterWithNewLine){
                                currCharacter[1] = "1";
                            }
                            else if (pointArray[j].x < (currCol-1)*oneCharacterWithSpace + minDist &&
                                     pointArray[j].x > ((currCol-1)*oneCharacterWithSpace) &&
                                     pointArray[j].y < (currRow-1)*oneCharacterWithNewLine + 2*minDist &&
                                     pointArray[j].y > (currRow-1)*oneCharacterWithNewLine + minDist){
                                currCharacter[2] = "1";
                            }
                            else if (pointArray[j].x < currCol*oneCharacterWithSpace &&
                                     pointArray[j].x > ((currCol-1)*oneCharacterWithSpace) + minDist &&
                                     pointArray[j].y < (currRow-1)*oneCharacterWithNewLine + 2* (minDist) &&
                                     pointArray[j].y > (currRow-1)*oneCharacterWithNewLine + minDist){
                                currCharacter[3] = "1";
                            }
                            else if (pointArray[j].x < (currCol-1)*oneCharacterWithSpace + minDist &&
                                     pointArray[j].x > ((currCol-1)*oneCharacterWithSpace) &&
                                     pointArray[j].y < (currRow*oneCharacterWithNewLine - rowSpace)&&
                                     pointArray[j].y > (currRow-1)*oneCharacterWithNewLine + 2* (minDist)){
                                currCharacter[4] = "1";
                            }
                            else if (pointArray[j].x < currCol*oneCharacterWithSpace &&
                                     pointArray[j].x > ((currCol-1)*oneCharacterWithSpace) + minDist &&
                                     pointArray[j].y < (currRow*oneCharacterWithNewLine - rowSpace) &&
                                     pointArray[j].y > (currRow-1)*oneCharacterWithNewLine + 2* (minDist)){
                                currCharacter[5] = "1";
                            }
                        }

                        currCode = String.join("", currCharacter);

                        // Displaying current character code - just to check whether it's translated correctly
                        //System.out.println(currCode);

                        for (int p = 0; p < Letters.numofletters; p++){
                            if (currCode.equals(Letters.idletters[p])){
                                translated_text =translated_text + Letters.trueletters[p];
                                break;
                            }
                        }
                    }
                    if (!currCode.equals("000000"))
                        translated_text =translated_text + " ";
                }
                return translated_text;
            }
        }
```