

Sprawozdanie

Kamil Niżnik 145238
Jakub Kwiatkowski 145356



Grudzień 2021

Omówienie





1 Temat

Celem projektu było zaimplementowanie prostego systemu OMR (ang. Optical Music Recognition System) wykrywającego podstawowe symbole stosowane w zapisie nutowym:

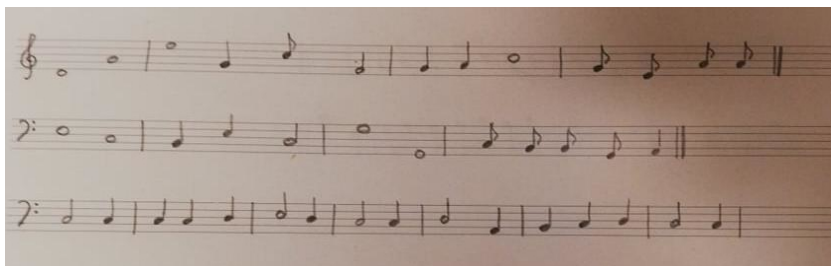
- Klucze

- wiolinowy (*G-clef*) - 
- basowy (*F-clef*) - 

- Nuty

- całą nutę (*semibreve*) - 
- półnutę (*minim*) - 
- ćwierćnutę (*crotchet*) - 
- ósemkę (*quaver*) - 

System został zaimplementowany z myślą o odręcznie pisanych nutach, dlatego może, lecz nie musi, działać z nutami drukowanymi.



Rysunek 1: Obraz oryginalny

2 Opis działania

System został zaprojektowany w sposób umożliwiający potokowe przetwarzanie obrazu.

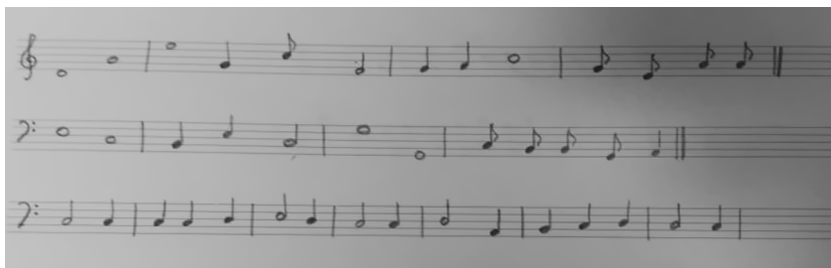
System można podzielić na trzy części:

- Preprocessing 2.1
- Object detection 2.2
- Object recognition 2.3

2.1 Preprocessing

Ze względu na charakter danych wejściowych, tj. zdjęć wykonanych aparatem smartfonu, niezbędny jest etap, który ma za zadanie oddzielić tło od elementów pierwszoplanowych, usunąć szum czy obrócić obraz do prawidłowej pozycji. Etap ten został rozbity na pięć zadań:

1. Desaturacja 2.1.1
2. Binaryzacja i odszumianie 2.1.2
3. Prostowanie 2.1.3
4. Wykrywanie linii 2.1.4
5. Usuwanie pięciolinii 2.1.5



Rysunek 2: Obraz skonwertowany do skali szarości

2.1.1 Desaturacja (*grayscale*)

Pierwszy krok przetwarzania, **niezbędny** do zastosowania pozostałych technik.

Desaturacja usuwa z obrazu informację o kolorze. W przypadku rozpoznawania nut informacja o kolorze jest zbędna, ponieważ standardowy zapis muzyczny używa jedynie czarnych znaków.

Efekt desaturacji rysunku 1 został przedstawiony na rysunku 2

2.1.2 Odszumianie i binaryzacja (*denoising & thresholding*)

Binaryzacja kompresuje obraz do dwóch kolorów: czarnego i białego. Upraszcza ona oddzielenie obiektów pierwszoplanowych od tła dzięki zwiększeniu kontrastu między obiektami.

Ze względu na niejednorodne oświetlenie i inne przeszkody utrudniające klasyczną binaryzację w programie zastosowano algorytm *local adaptive thresholding*, który dobiera wartość progu bazując na jasnościach sąsiadujących pikseli.

Odszumianie służy usunięciu niewielkich, losowych skupisk pikseli wynikających z niedoskonałości zdjęcia i zakłócających zbinaryzowany obraz.

Efekty binaryzacji i odszumiania rysunku 2 zostały przedstawione na rysunku 2



Rysunek 3: Zbinaryzowany i odszumiony obraz 2

2.1.3 Prostowanie

Prostowanie ma na celu obrócenie obrazu tak, aby linie pięciolinii były w położeniu horyzontalnym.

W implementacji wypróbowaliśmy dwie techniki znajdowania kąta obrotu:

Pierwszą metodą jest *Projection Profile* - kąt obrotu wyliczany jest na podstawie różnicy maksimów histogramów generowanych co pewien mały kąt.

Druga metoda natomiast wykorzystuje transformatę Hougha do znalezienia pięciolinii i na podstawie znalezionych kątów wylicza kąt obrotu.

Efekty prostowania rysunku 2 zostały przedstawione na rysunku 2

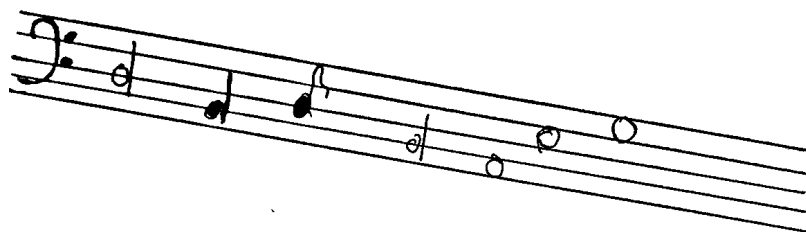
2.1.4 Wykrywanie linii

Po wstępnym przetworzeniu zdjęcia dokonywana jest detekcja, a następnie usunięcie pięciolinii ze zdjęcia. Może to być osiągnięte na wiele sposobów, a dobór odpowiedniej techniki wciąż jest przedmiotem opracowań naukowych.

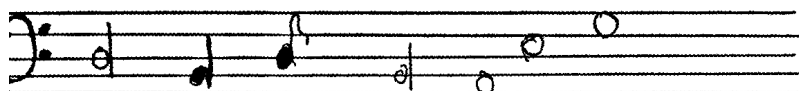
W niniejszej implementacji do wykrywania linii wykorzystano operację morfologiczną otwarcia z elementem strukturalnym odpowiadającym kształtowi linii (tj. prostokąt o dużej szerokości i małej wysokości).

Wykryte linie przedstawione są na rysunku 6

Analogicznie wykrywane są linie pionowe (rysunek 7), które służą do odbudowy



Rysunek 4: Oryginalny obraz



Rysunek 5: Obraz obrócony do pozycji horyzontalnej

nut po operacji usunięcia pięciolinii.

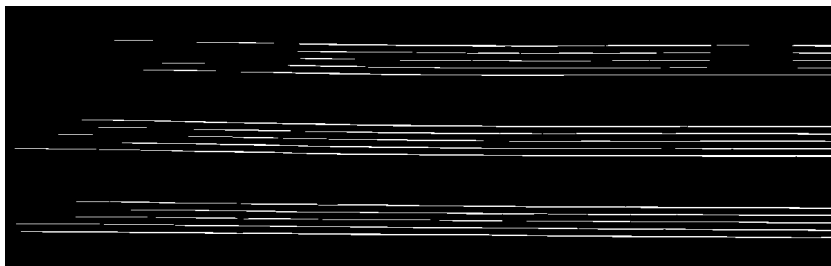
2.1.5 Usuwanie pięciolinii

Wykryte w poprzednim podpunkcie linie poziome są usuwane z obrazu z wykorzystaniem logicznego operatora OR.

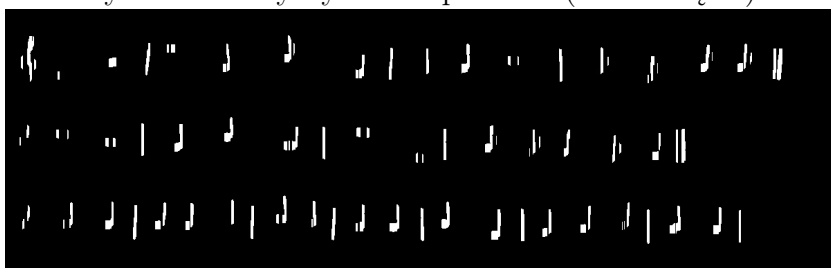
W rezultacie pozostaje jednak wiele niewyeliminowanych, samotnych pikseli. Piksele te usuwane są morfologiczną operacją domknięcia z elementem strukturalnym o kształcie małego prostokąta.

Na końcu tego etapu następuje istotny punkt przetwarzania - odbudowa uszkodzonych nut z użyciem otwarcia i wykrytych uprzednio linii pionowych.

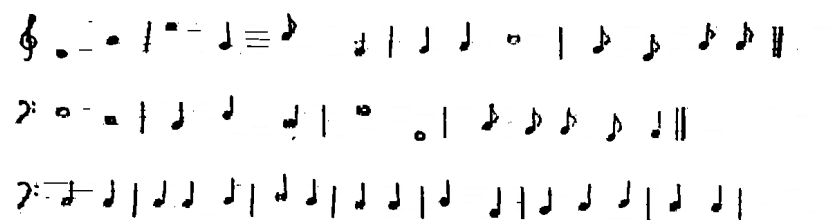
Efekty usunięcia pięciolinii zostały przedstawione na rysunku 8



Rysunek 6: Wykryte linie poziome (do usunięcia)



Rysunek 7: Wykryte linie pionowe (do odbudowy)



Rysunek 8: Obraz z usuniętą pięciolinia

2.2 Object detection

Po usunięciu pięciolinii następuje etap wyodrębniania symboli ze zdjęcia. Wykorzystywany jest do tego sparametryzowany algorytm *Connected-Components Labeling* zaimplementowany w OpenCV. Wykrywane są obiekty, których wielkość mieści się w ustalonym zakresie (rysunek 9).



Tej samej operacji poddawane są linie poziome (obraz źródłowy: rysunek 6), które na tym etapie można już rozumieć jako linie wchodzące w skład pięciolinii. Dodatkowo, ze względu na dość duże poszatkowanie obrazu źródłowego (rysunek 10), algorytm uzupełnia brakujące linie interpolując wartości pochodzące ze zbioru wszystkich wykrytych linii.

Algorytm rozdzielania pięciolinii jest oparty o odległość między liniami - jeśli wykryta zostanie znacząca przestrzeń między liniami to uznawana jest ona za granicę między pięcioliniami (rysunek 11).

2.3 Object recognition

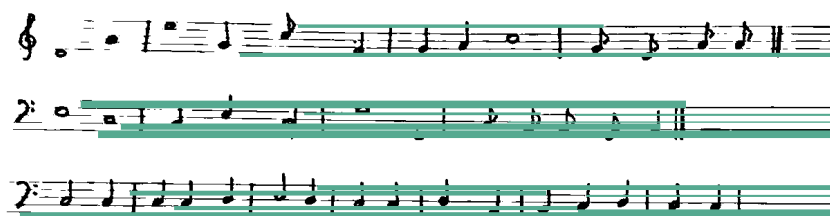
2.3.1 Ograniczenia

Na początku musimy zdefiniować kilka ograniczeń, na których będzie bazować rozpoznawanie wykrytych obiektów.

- Ograniczenia dziedzinowe
 - Klucz wiolinowy () jest zdecydowanie wyższy od wysokości pięciolinii
 - Klucz basowy () jest mniejszy niż wysokość pięciolinii.
 - Główki nut mają wysokość jednego odstępu między liniami pięciolinii
- Dodatkowe założenia
 - Każda pięciolinia zawierająca nuty rozpoczyna się kluczem
 - Główki nut mają horyzontalny aspect ratio ($\geq 1:1$): szerokość jest co najmniej równa wysokości
 - Laseczki nut znajdują się zawsze nad główką, po prawej stronie.



Rysunek 9: Wykryte obiekty (do klasyfikacji)



Rysunek 10: Wykryte linie pięciolinii



Rysunek 11: Wykryte pięciolinie

2.3.2 Rozpoznawanie kluczy

Rozpoznawanie kluczy oparto na naiwnym podejściu wykorzystującym założenia podane powyżej:

Wybieramy pierwsze obiekty z pięciolinii i bazując na założeniach 1 i 2 klasyfikujemy je na podstawie ich wysokości.

2.3.3 Rozpoznawanie nut

Rozpoznawanie rodzaju opiera się na charakterystycznej budowie nuty: każda nuta składa się z główki (*head*) i opcjonalnej laseczki (*stem*). W rozpatrywanych przypadkach mamy dwa rodzaje główek:

- pełną (closed)
- otwartą (open)

oraz trzy rozpatrywane opcje laseczek:

- brak laseczki
- pełną laseczkę (whole)
- laseczkę ósemkową (quaver)

Dzięki temu możemy określić minimalne charakterystyczne kombinacje tych elementów dla każdego rodzaju nut:

cała nuta (*semibreve*) - nie posiada laseczki

półnuta (*minim*) - posiada pełną laseczkę oraz otwartą główkę

ćwierćnuta (*crotchet*) - posiada pełną laseczkę i pełną główkę

ósemka (*quaver*) - posiada ósemkową laseczkę

Rozpoznawanie laseczki Algorytm rozpoznawania laseczki bazuje na analizie sum kolumn i rzędów obrazu. Przebiega on według następującego schematu:

1. Mając dany czarno-biały obraz nuty zaneguj go tak, aby nuta miała kolor biały ('negatyw').
2. Znormalizuj negatyw do postaci zmiennoprzecinkowej z zakresu 0.0 - 1.0.
3. Stwórz wektor sum kolumnowych ('kolumny') po wszystkich kolumnach negatywu.
4. Ustal próg definiujący minimalną wysokość głównej laseczki ('próg').
5. Przefiltruj stabilnie kolumny eliminując te, które nie przekraczają progu.
6. Z pozostałych kolumn wybierz tę z najniższym indeksem - będzie to lewa granica laseczki ('left offset').
7. Jeśli nie istnieje taka kolumna, to **nuta nie posiada laseczki**.
8. Ogranicz negatyw od lewej strony przez left offset.
9. Ustal próg definiujący minimalną wysokość ósemkowej laseczki ('próg ósemkowy').
10. Przefiltruj stabilnie kolumny eliminując te, które nie przekraczają progu ósemkowego.
11. Z pozostałych kolumn wybierz tę z najniższym indeksem - będzie to lewa granica laseczki ('right offset').
12. Analogicznie postępuj dla rzędów, aby ograniczyć laseczkę od góry i od dołu.
13. Wybierz najwęższy i najszerszy odcinek laseczki.
14. Jeśli najszerszy odcinek jest przynajmniej trzykrotnie większy niż najwęższy odcinek, to uznajemy laseczkę za ósemkową, w przeciwnym przypadku uznajemy laseczkę za pełną.

Rozpoznawanie główek Algorytm rozpoznający główkę działa analogicznie do algorytmu rozpoznającego laseczkę. Najpierw odcinamy z obrazu część z laseczką, następnie tworzymy negatyw i znajdujemy lewy, prawy, górny i dolny offset.

Typ główki rozpoznajemy na podstawie punktu centralnego główki: wyznaczamy środek główki, obliczamy sumę w środkowym rzędzie oraz środkowej kolumnie. Uznajemy, że główka jest pełna, jeśli obie sumy są większe od, odpowiednio, połowy długości i połowy szerokości główki.

Przykłady