Sprawozdanie z laboratorium Komunikacja Człowiek – Komputer

Temat : Czytanie nut

Wykonali :

Anna Żurczak 127227
Mikołaj Stankowiak 127234

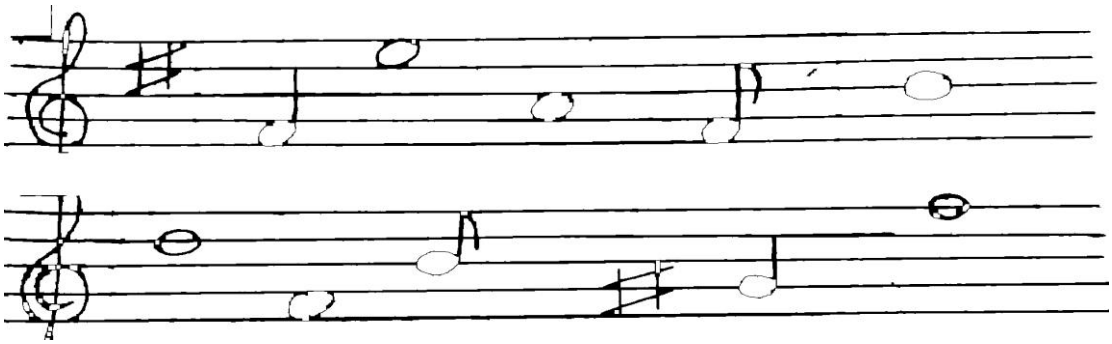
Grupa I4 Środa 8:00

1. **Wstęp**

   Celem naszego projektu było zaimplementowanie i zbadanie metod rozpoznawania nut na pięciolinii. Każdy z uczestników projektu w równym stopniu przyczynił się do jego rozwoju. Nie było podziału obowiązków, gdyż wszystkie czynności wykonywaliśmy razem.

2. **Algorytm - wczytywanie i filtracja zdjęcia**

   Obraz wczytywany jest od razu w skali odcieni szarości za pomocą funkcji **laduj_szary_obraz()** z pliku **obraz.py**. Następnie obraz zostaje przeskalowany do szerokości 2000px, aby parametry funkcji operujących na obrazie były niezmienne(**funkcja skaluj_szerokosc()** z pliku **obraz.py**). W ostatnim etapie filtrujemy zdjęcie, aby uzyskać tzw. maskę która jest pierwotnym zdjęciem poddanym dylatacji i następnie erozji.



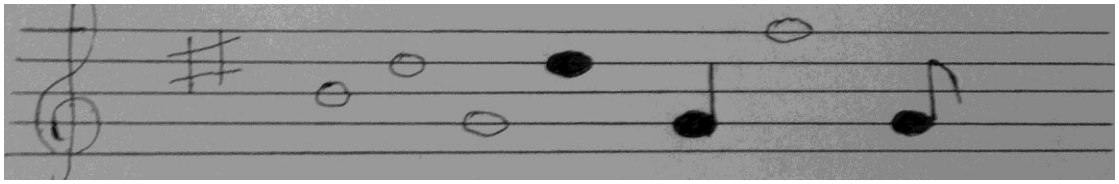
Rysunek 1 Maska

3. **Wyszukiwanie obszaru pięciolinii za pomocą maski**

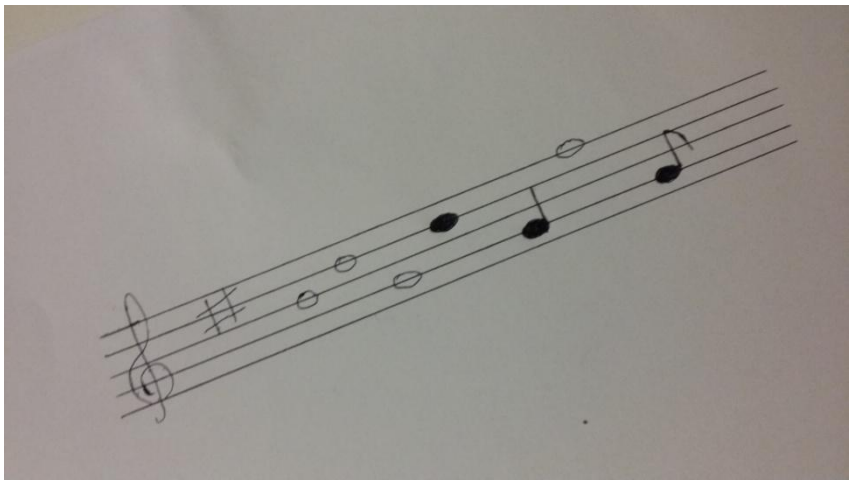
   Na początku uruchomiona zostaje funkcja **wyszukaj_pieciolinie()** z pliku **pieciolinia.py,** która korzysta z **cv2.GaussianBlur()** w celu rozmycia i pozbycia się poszarpanych linii. Następnie wyszukujemy krawędzie funkcją **cv2.Canny().** Finalnie funkcja zwraca linie znalezione przez funkcję **cv2.HoughLinesP()**.

   Kolejną funkcja **kat_obrotu()** z pliku **obraz.py** oblicza średni współczynnik „a" równania prostej dla wszystkich znalezionych linii i zamienia go na kąt nachylenia prostej. Obraz zostaje obrócony o uprzednio policzony kąt. Używamy do tego funkcji **cv2.getRotationMatrix2D()** i **cv2.warpAffine()** .

   Aby przyciąć prawidłowo nowe zdjęcie znów szukamy na obróconej masce pięciolinii funkcją **wyszukaj_pieciolinie()** z pliku **pieciolinia.py.** Funkcja **przytnij()** z pliku **obraz.py** szuka minimalnych współrzędnych linii w celu znalezienia pierwszej i ostatniej linii na obrazie a następnie zostawiając 20% zapasu wysokości przycina zdjęcie.

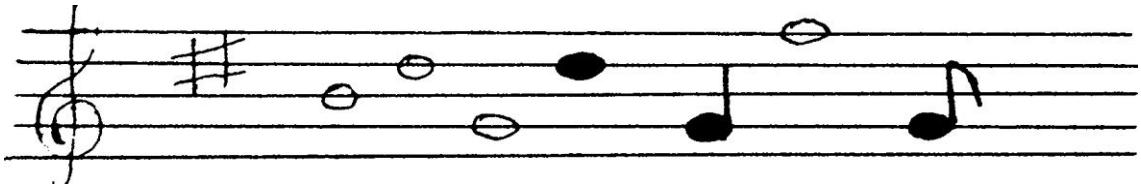


Rysunek 2 Przycięte zdjęcie



Rysunek 3 Oryginalne zdjęcie

Tak przygotowane zdjęcie poddawane jest progowaniu (**progowanie()** i **rozmaz_proguj()** z pliku **obraz.py**).



Rysunek 4 Po progowaniu

4. Oddzielanie pięciolinii

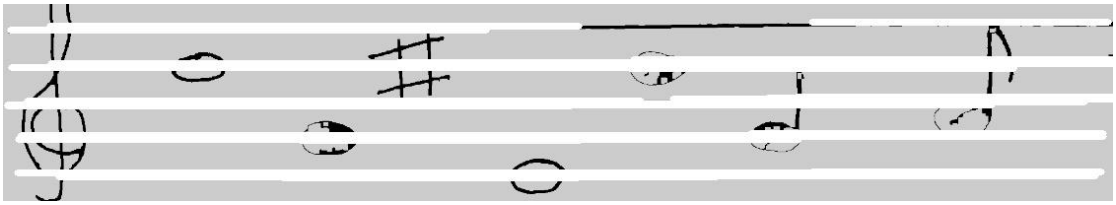
W przypadku zdjęć, które mają wiele pięciolinii na jednym obrazie konieczne jest ich wyodrębnienie. Zajmuje się tym funkcja **wydzielpieciolinie()** z pliku **pieciolinia.py** która :

- zaczynając od połowy obrazu przesuwając się 100px w lewo i w prawo skanuje go w pionie zliczając ilość czarnych pikseli (zmienna delta),
- wybierana jest współrzędna x zawierająca najmniejszą wartość delty tj. brak nut na danej szerokości,
- na wybranej współrzędnej x poszukujemy współrzędnych linii, sprawdzając czy zmienił się kolor piksela,
- współrzędne linii grupujemy po 5,
- za pomocą powstałych współrzędnej x i cząstkowych y wycinamy pięciolinie,
- funkcja zwraca tablice obrazów wyciętych z oryginalnego zdjęcia i z maski.

Kolejne punkty realizowane są dla każdej wyciętej pięciolinii osobno.

5. Przygotowanie do skanowania

Na początku maska i obraz za pomocą funkcji **skaluj_wysokosc()** z pliku **obraz.py** normalizowane są do wysokości 200px, aby ujednolicić rozmiar pięciolinii. Ponownie wyszukujemy pięciolinie na masce i na podstawie położenia pięciolinii przycinamy obraz, aby pozbyć się ramki powstałej przy punkcie 4. Przycięty obraz zostaje poddany progowaniu, ponieważ proces skalowania wprowadza piksele o kolorach innych niż czarny i biały.



Rysunek 5 Wykrywanie linii

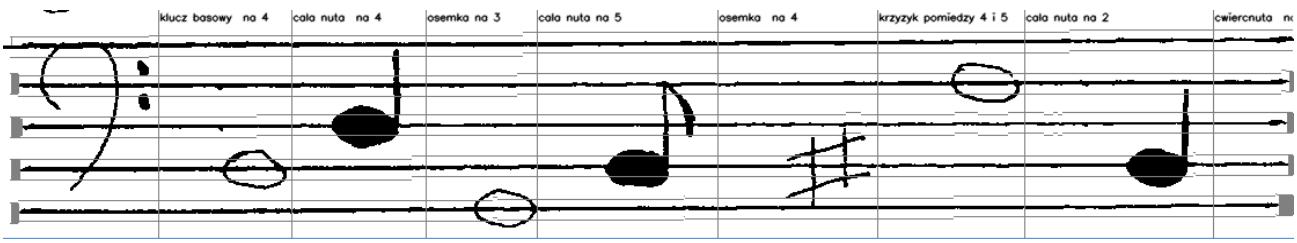
6. **Wyszukiwanie nut - zbieranie parametrów skanera**

   Na podstawie 1/3 szerokości obrazu ustalamy rzeczywiste współrzędne początkowe pięciolinii (funkcja **znajdz_rzeczywista_pieciolinie()** z pliku **pieciolinia.py**), aby nie brać pod uwagę krzywizn w dalszej części obrazu. Na podstawie tych współrzędnych wyznaczamy 10 linii skanujących.

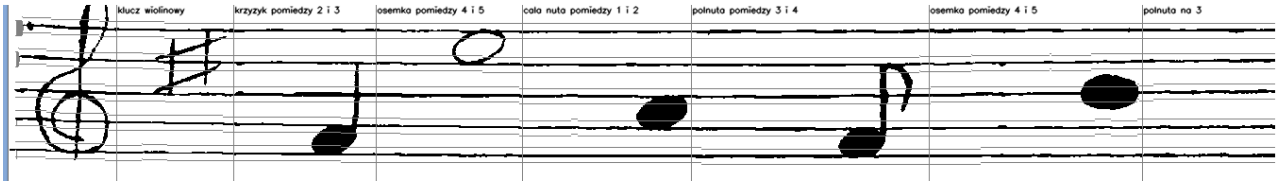
7. **Wyszukiwanie nut**

   Za rozpoznawanie nut odpowiedzialna jest funkcja **skaner()** z pliku **main.py**. Skaner analizuje piksel po pikselu pięciolinie na wysokości linii skanujących. Kroki algorytmu:

   - inicjalizacja stałych np. rozmiar nuty to 130% średniej odległości między liniami,
   - inicjalizacja zmiennych (wypełnienie ich zerami),
   - zliczamy ilość zmian koloru pikseli oraz na której linii skanującej ta zmiana nastąpiła(zmienne delta i wykryte_linie[]),
   - rozróżniamy dwie początkowe zmiany koloru: powolną i szybką. Potrzebne jest to, aby odróżnić całą nutę od innych nut. Jeżeli po pierwszej zmianie koloru nie nastąpiła zmiana przez określoną ilość pikseli to zmiana uznawana jest za wolną to wyklucza pełną nutę,
   - interpretacja wyników na podstawie zebranych danych:
     - wszystkie symbole posiadają unikalną kombinację wartości parametrów, np. krzyżyk zawsze znajduje się w obszarze 4 linii skanujących oraz delta wynosi ok. 16,
     - na początku oczekujemy klucza, po wykryciu któregoś dopiero szukane są nuty,
     - informacje o wybranym symbolu zwracane są do okna konsoli,
   - po znalezieniu nuty wszystkie parametry są resetowane.



Rysunek 6 Efekt końcowy



Rysunek 7 Efekt końcowy 2

8. Wnioski z działania algorytmu

Algorytm radzi sobie z prostymi przypadkami przekrzywionych, wyraźnie narysowanych pięciolinii na jasnym i równomiernym tle.

Mimo prób dobrego odfiltrowania zdjęcia nie udało nam się uzyskać zadowalających efektów w przypadku prześwietlonych zdjęć i/lub z nierównym oświetleniem. Zamiast progowania należałoby wykorzystać bardziej złożony filtr, który korzysta z gwałtownych zmian zawartości pikseli, aby informacja o wypełnieniu nut nie była tracona, tak jak w przypadku naszej maski. Oprócz tego zbiór warunków rozpoznających symbol powinien być przetestowany na większej ilości przypadków testowych, aby dobrać bardziej uniwersalne parametry. W przypadku wyszukiwania poziomych linii powinno się usunąć pionowe, gdyż funkcja znajduje je w kluczu wiolinowym. Nie udało nam się wyodrębnić pionowych linii, dlatego że **HougeLineP()** zwraca specyficzną tablicę z biblioteki **numpy** i nie poradziliśmy sobie z modyfikacją tej tablicy.