Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznanie tónov hry na klavíri

Bakalárska práca

2020
Šimon Babál

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Rozpoznanie tónov hry na klavíri

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Paula Štancelová

Bratislava, 2020
Šimon Babál

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Šimon Babál
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Rozpoznanie tónov hry na klavíri
Piano tones recognition

Anotácia: 1. Prehľad metód pre efektívnu detekciu stlačených kláves pri hre na klavíri v kombinácií s použitím rôznych existujúcich a dostupných technológií.
2. Analýza možností detekcie v reálnom čase.
3. Rozpoznanie tónov hry na základe detekcie stlačených kláves.
4. Export rozpoznaných tónov do štandardizovaných formátov MIDI/ MusicXML.

Vedúci: RNDr. Paula Štancelová
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 03.10.2019

Dátum schválenia: 07.10.2019 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Veľká vďaka patrí hlavne mojej školiteľke RNDr. Paule Štancelovej za jej cenné rady a poskytnutý čas. Taktiež ďakujem svojej rodine za psychickú podporu a za možnosť študovať.

Abstrakt

Cieľom tejto bakalárskej práce je tvorba aplikácie, ktorá vie vykonávať automatickú hudobnú transkripciu hry na klavíri pomocou počítačového videnia. Výsledkom tejto transkripcie je súbor vo formáte MIDI alebo MusicXML. Na začiatku práce je vysvetlená potrebná teória z oblasti hudby a počítačového videnia pre lepšie pochopenie problematiky a zatým nasleduje prehľad jej riešení vo svete. Ďalšou časťou je návrh aplikácie, ktorý bol podkladom pre implementáciu. V implementačnej časti sú najskôr vysvetlené všetky technológie, ktoré sme v práci použili, a následne sú popísané všetky postupy implementácie. V poslednej kapitole sa nachádza analýza výsledkov implementácie a možnosti budúceho výskumu.

Kľúčové slová: automatická hudobná transkripcia, spracovanie obrazu, klavír

Abstract

The aim of this bachelor's thesis is to create an application that can perform automatic musical transcription of piano playing using computer vision. The result of this transcription is MIDI or MusicXML file. At the beginning of the work is explained the necessary theory in the field of music and computer vision for a better understanding of the issue, followed by an overview of its solutions in the world. The following section is the design of the application, which was the basis for implementation. The implementation part first explains all the technologies we used in the work, and then describes all implementation procedures. The last chapter contains an analysis of the results of implementation and the possibilities of future research.

Keywords: automatic music transcription, image processing, piano

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Cieľom tejto bakalárskej práce je tvorba softvéru pre automatickú hudobnú transkripciu hry na klavíri do formátu MIDI alebo MusicXML pomocou počítačového videnia s využitím programovacieho jazyka Python a knižnice OpenCV. V dnešnej dobe sa vyrába množstvo digitálnych hudobných nástrojov. Medzi tieto nástroje patria aj elektrické klavíry, ktoré väčšinou ponúkajú možnosť automatickej hudobnej transkripcie cez rozhranie MIDI. Doteraz však neexistovalo riešenie, ktoré by túto možnosť poskytovalo aj pri klasických neelektrických klavíroch, ako je krídlo alebo pianino. V tejto práci sme chceli vytvoriť prototyp takéhoto riešenia pomocou počítačového videnia, čo bolo pomerne veľkou výzvou, keďže doteraz boli tejto alebo podobnej problematike venované len 3 zahraničné práce, pričom implementačné riešenie žiadnej z nich nebolo dostupné. Hlavná časť tejto práce sa venuje popisom algoritmov na rozpoznanie klávesov v rámci klavíra a na detekciu stlačených klávesov.

Vo východiskovej kapitole vysvetľujeme čitateľovi základné pojmy, ktoré sa tejto práce týkajú. Sú to pojmy z oblasti hudby a spracovania obrazu. V ďalšej časti tejto kapitoly je prehľad možných riešení našej problematiky a prác, ktoré sa venujú problému automatickej hudobnej transkripcie hry na klavíri.

V nasledujúcich dvoch kapitolách sa zaoberáme popisom nášho riešenia daného problému. Najskôr všeobecne definujeme, čo má aplikácia ponúkať a na akých princípoch má vykonávať segmentáciu klávesov a následnú detekciu ich stláčania, potom popisujeme, aký programovací jazyk, knižnice a ďalšie nástroje sme zvolili na implementáciu nášho riešenia. Ďalej sa venujeme konkrétnym postupom pri implementácii, pričom začíname s používateľským rozhraním. Tento popis zároveň môže slúžiť aj ako návod na používanie našej aplikácie. Nasledujú najzložitejšie časti práce, ktorými sú segmentácia klávesov a rozpoznávanie ich stlačenia. Túto časť zakončujeme vysvetlením spôsobu spracúvania záznamu o stlačených klávesoch a exportu tohto záznamu do formátu MIDI alebo MusicXML. Dané formáty sme zvolili z dôvodu ich najväčšej používanosti medzi hudobníkmi, čiže je možné otvoriť ich aj cez rôzne programy na úpravu nôt.

Výsledná aplikácia, ktorú sme v implementačnej fáze tejto práce naprogramovali, umožňuje vložiť video, pričom výstup z nej je súbor v MIDI alebo MusicXML formáte, v ktorom je zachytené, ako boli vo videu stláčané klávesy na klavíri. V aplikácii je po-

trebné po vložení videa určiť informácie o klaviatúre a následne vie aplikácia segmentovať jednotlivé klávesy. Po segmentácii hľadá rozdiely medzi jednotlivými snímkami a z týchto rozdielov vie určiť, kedy bol nejaký kláves stlačený a kedy pustený.

V poslednej kapitole práce rozoberáme testovanie nášho riešenia. Najskôr popisujeme, aký efektívny je náš kód z hľadiska času vykonávania jednotlivých operácii, v ďalšej časti analyzujeme jeho presnosť. Taktiež sa venujeme aj nedostatkom nášho riešenia. V prvom rade analyzujeme nedostatky týkajúce sa časovej efektivity programu a následne popisujeme, s akými videami má naše riešenie najväčší problém. Úplne na záver práce sa zaoberáme potenciálom rozvíjania a vylepšovania nášho riešenia.

Kapitola 1

Prehľad problematiky

1.1 Teória hudby

Klavír

Klavír je strunový hudobný nástroj, ktorého zvuk vzniká šírením vlnenia vyvolaného kmitaním strún. Klavírové struny sa rozkmitajú pomocou kladiviek, ktoré sú ovládané klávesmi. Klávesy sa nachádzajú na klaviatúre, pričom pozícia klaviatúry je znázornená na obrázku 1.1. Štandardný klavír obsahuje 88 kláves, pričom 52 z nich je bielych a 36 čiernych.

Obr. 1.1: Pozícia klaviatúry na klavíri

Okrem štandardných klavírov existujú však aj špeciálne koncertné alebo elektrické klavíry, pri ktorých sa môžu vyššie popísané vlastnosti mierne líšiť.

Tón

Pojem tón predstavuje ľubovoľné počuteľné vlnenie, ktoré bolo spôsobené rozkmitaním klavírových strún pomocou kladiviek. Pri elektrických klavíroch je tón generovaný digitálne a vlnenie sa šíri z reproduktoru, pretože elektrický klavír nemá struny. Výška tónu je daná frekvenciou kmitania klavírových strún a mala by byť relatívne konštantná. Tón je základným prvkom hudby. Počuteľné frekvencie tónov sú pre ľudské ucho od 20 Hz až po 20 kHz [13].

Homofónna hudba

Homofónna hudba je taká, pri ktorej súčasne hrá najviac jeden tón v jednom momente. Hovoríme jej tiež jednohlasná. Hudobné nástroje, pre ktoré je homofónna hudba najtypickejšia, sú väčšinou dychové. Ako príklad môžeme uviesť trúbku, hoboj, trombón a rôzne druhy fláut. Homofónnu hudbu je možné vytvoriť pomocou ktoréhokoľvek hudobného nástroja. Pre tento typ hudby sa robí hudobná transkripcia najjednoduchšie, keďže obsahuje najmenej dát, ktoré je potrebné spracovať.

Polyfónna hudba

V tejto práci budeme polyfónnu hudbu definovať ako niekoľko poskladaných homofónnych hudieb. Polyfónnej hudbe sa zvykne hovoriť hovoriť viachlas. Polyfónnu hudbu dokážu tvoriť zložitejšie hudobné nástroje, ako je napríklad klavír, harfa, organ a gitara. Pre tento typ hudby sa robí hudobná transkripcia zložito, pretože obsahuje veľké množstvo dát, ktoré treba spracovať.

Hudobná transkripcia

Hudobná transkripcia je prepis záznamu znejúcej hudby do inej podoby. V tejto práci sa budeme venovať digitálnej transkripcii, pri ktorej máme tri možnosti, ako ju môžeme zrealizovať. Prvou a najviac používanou je transkripcia v reálnom čase hry na digitálnom klavíri cez rozhranie MIDI, pričom klavír komunikuje s počítačom cez kábel alebo Bluetooth. Pri tomto type transkripcie klavír posiela informácie o stlačených tónoch úplne priamo vo formáte MIDI a počítač ich môže následne uložiť do MIDI súboru, alebo vytvoriť notový záznam stlačených tónov. Ďalšou možnosťou je vykonať transkripciu využitím audio záznamu. Táto možnosť je však veľmi zložitá pri polyfónnej hudbe, pretože je veľmi ťažké oddeliť jednotlivé frekvencie tónov v tomto type záznamu. Poslednou možnosťou je transkripcia s použitím obrazového záznamu ako zdroja informácií, ktorej bude primárne venovaná táto práca. Pri tejto transkripcii bude ako vstup video súbor a ako výstup hudobnej transkripcie bude MIDI alebo

MusicXML súbor, ktorý bude obsahovať dáta, ako zahrať skladbu, ktorá bola na danom videu.

Všetky vyššie popísané možnosti je však možné kombinovať a rovnako dnes existujú aj riešenia s využitím neurónových sietí. Tieto riešenia sú bližšie popísané v ďalšej časti tejto kapitoly.

1.2 Teória spracovania obrazu

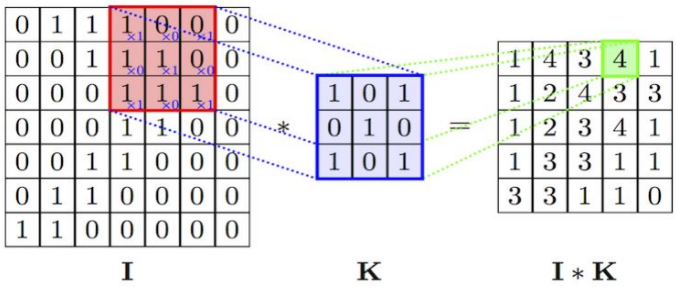
Diskrétna konvolúcia

Diskrétna konvolúcia [16] je veľmi často využívaná na predspracovanie obrazu a môžeme ju nasledovne definovať, pričom $f(x, y)$ je funkcia obrazu a $h(x - m, y - n)$ je konvolučná maska:

$$g(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n)h(x - m, y - n)$$

Operácie diskrétnej konvolúcie sa počítajú podľa vyššie popísanej definície a dnes máme na tieto výpočty k dispozícii vysoko optimalizované implementácie, napríklad pre Intel procesory v knižnici Intel Integrated Performance Primitives (IPP), ktorú používajú Matlab, OpenCV a ďalšie softvérové nástroje s určením pre spracovanie obrazu [16].

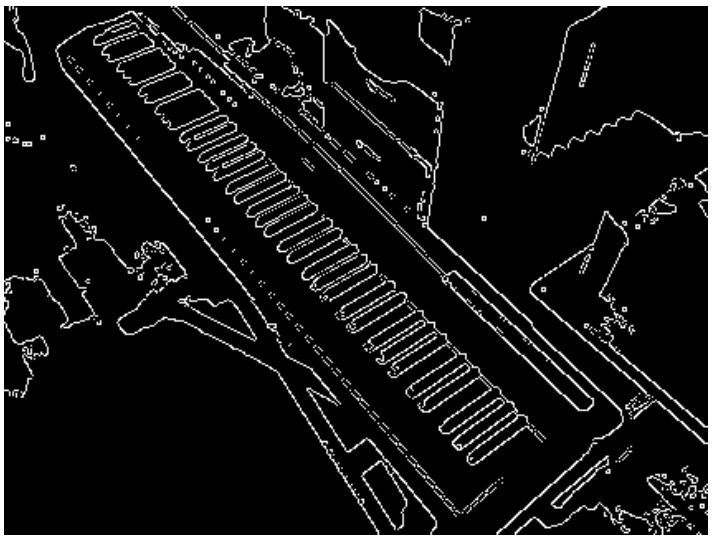
Pre skrátený zápis operácie diskrétnej konvolúcie [16] sa používa znak *hviezdička*, ako je znázornené na obrázku 1.2. Konvolučnú masku vždy definujeme tak, aby jej veľkosť bola nepárna, napríklad $3x3$ alebo $5x5$. Robíme to tak z toho dôvodu, aby bola jednoznačná stredová pozícia, na ktorú budeme zapisovať výslednú hodnotu. Tieto masky vieme využívať na vyhladzovanie, odstraňovanie šumu, potláčanie alebo zvýraznenie hrán. Pokiaľ nie je našim úmyslom obraz zosvetliť alebo stmaviť, maska by mala byť navrhnutá tak, aby bol súčet všetkých hodnôt konvolučného jadra rovný jednej.



Obr. 1.2: Príklad výpočtu konvolúcie [14].

Segmentácia obrazu

Segmentácia obrazu [17] je proces jeho transformácie, ktorého cieľom je rozdeliť obraz na objekty alebo oblasti. Metódy segmentácie môžeme zhrnúť do troch kategórií, ktorými sú: prahovanie, segmentácia založená na hranách a segmentácia založená na oblastiach, pričom prvé dve budú bližšie objasnené v nasledovných odsekoch.



Obr. 1.3: Príklad segmentácie [7]. Na obrázok bola aplikovaná Cannyho detekcia hrán a následne Otsuov algoritmus na optimálne prahovanie.

Prahovanie

Prahovanie [17] je najjednoduchšia, výpočtovo najmenej náročná a najrýchlejšia technika na segmentáciu. Používa sa pri nej jasová konštanta, ktorú nazývame prah. Prahovaním transformujeme obraz takým spôsobom, že prechádzame cez každý pixel obrazu. Ak je jasová hodnota pixlu menšia než prah, hodnotu jasu daného pixlu zmeníme na 0. V opačnom prípade zmeníme hodnotu jasu daného pixlu na maximálnu hodnotu. Toto sa nazýva binarizácia obrazu a môžeme ju vidieť na obrázku 1.3. Existuje však aj niekoľko modifikácii prahovania, ako napríklad čiastočné prahovanie, pri ktorom časť obrázku ostane bez zmeny a prahujú sa len pixle, ktorých hodnota jasu spadá do daného intervalu. Ďalšou modifikáciou prahovania je využitie viacerých prahov. Výsledkom takéhoto prahovania však už nie je binárny obraz, ale obraz s väčším počtom farieb. Počet týchto farieb je daný počtom prahovacích hodnôt zväčšených o jeden. Prah môžeme určiť manuálne, alebo na nájdenie prahu môžeme použiť jeden z nasledovných prístupov:

1. Na základe údajov z histogramu. V tomto prípade sa jedná o globálne prahovanie [17], keďže na celý obraz aplikujeme ten istý prah. V bimodálnych histogramoch

sa zvykne vyberať prah ako minimum medzi dvoma najvyššími lokálnymi maximami.

2. Na základe metódy podielu [17]. Ak vieme, akú časť obrazu pokrýva objekt, tak vyberáme prah takým spôsobom, aby objekt pokrýval dané percento obrazu a pozadie zvyšok.

3. Na základe optimálneho prahovania. Optimálne prahovanie [17] určuje prah ako najbližšiu šedú úroveň zodpovedajúcu minimu pravdepodobnosti medzi maximami dvoch alebo viacerých normálnych rozdelení. Takéto prahovanie vedie k minimálnej chybe pri segmentácii.

Segmentácia založená na hranách

Segmentácia založená na hranách [17] pracuje s lokálnymi hranami, ktoré boli detegované hranovými operátormi. Lokálne hrany sa nachádzajú na tých miestach obrazu, na ktorých dochádza k ostrému prechodu v úrovni šedej, vo farbe, v textúre, alebo v inej vlastnosti. Najväčšími problémami v tejto metóde sú detekcia falošných hrán alebo nedetegovanie niektorých hrán, ktoré sa na obraze nachádzajú.

V tejto práci bolo jednou z možností robiť segmentáciu pomocou detekcie hrán za pomoci ostrenia, ktoré sa zvykne realizovať diskrétnou konvolúciou. Keďže klavír obsahuje pravidelné hrany, ktoré zvyknú byť rovnobežné, bolo možné použiť filtre slúžiace na vertikálne a horizontálne hrany. Na obrázkoch 1.4, 1.5 a 1.6 sa nachádza niekoľko najznámejších konvolučných masiek.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

(a) Horizontálna maska (b) Vertikálna maska

Obr. 1.4: Prewittov filter [16].

Segmentačnou technikou využívajúcou detekciu hrán je aj Houghova transformácia [17]. Používame ju vtedy, keď ideme detegovať objekty so známym tvarom hranice. Houghova transformácia dokáže detegovať hranice objektu, ak sú známe ich analytické vyjadrenia. Pôvodne bola Houghova transformácia navrhnutá na detekciu rovných čiar, ale je možné použiť ju aj na detekciu iných geometrických objektov, ktoré vieme parametricky vyjadriť.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(a) Horizontálna maska (b) Vertikálna maska

Obr. 1.5: Sobelov filter [16].

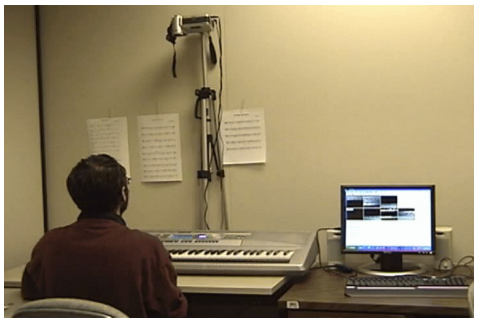
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Obr. 1.6: Diskrétny Laplacian je definovaný ako suma druhej diferencie v danom pixeli vzhľadom na okolie pixela [16].

Na segmentáciu sa okrem vyššie popísaných postupov používajú aj ucelené algoritmy, ktoré vykonávajú aj iné operácie okrem prahovania a ostrenia. Jeden z nich sa nazýva Cannyho detektor hrán [11]. Tento algoritmus najskôr pomocou gaussovského filtra eliminuje šum. Následne aplikuje jeden z gradientných operátorov, najčastejšie Sobelov filter. Po tomto kroku nasleduje stenčenie hrán. Táto funkcia vyberá z hodnôt gradientov len lokálne maximá. Posledným krokom tohto algoritmu je prahovanie, čiže binarizácia obrazu. Cannyho detektor hrán je v praxi veľmi užitočný, pretože je na rozdiel od jednoduchších prístupov na detekciu hrán menej citlivý na šum.

Základné morfologické operácie

Za základné morfologické operácie považujeme dilatáciu a eróziu [16]. Ich výpočet používa podobný mechanizmus, ako pri filtrácii, takže každý pixel obrazu je vypočítaný podľa algoritmu, ktorý využíva hodnoty jeho topologických susedov. Pomocou binárnej dilatácie zväčšujeme plochu bieleho objektu na čiernom pozadí a pri erózií presne naopak, čiže zmenšujeme plochu bieleho objektu na čiernom pozadí [16]. Tieto operácie nemusia byť vždy navzájom inverzné, pretože ak je napríklad eróziou malý objekt úplne odstránený, už nie je možná jeho rekonštrukcia pomocou dilatácie. Kombináciou týchto morfologických operácii vznikajú ďalšie morfologické operácie [16]. Morfologické otvorenie je operácia erózie nasledovaná dilatáciou a v opačnom poradí vzniká operácia morfologického uzatvorenia. Pri morfologickom otváraní a zatváraní sa počíta s tým, že pre obe operácie sa používa rovnaký štrukturálny element.



Obr. 1.7: Pozícia kamery pri natáčaní videa. [18]

1.3 Možné riešenia problematiky

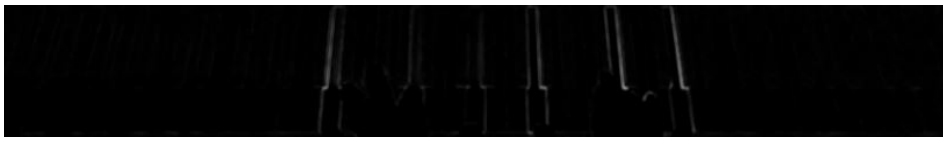
1.3.1 Na základe spracovávania zvuku

Automatická hudobná transkripcia hry na klavíri pomocou spracovania zvuku je predmetom výskumov približne od roku 1975. Pri homofónnej hudbe môžeme v dnešnej dobe považovať tento problém za vyriešený. Komplikovanejšie je to však pri polyfónnej hudbe, ktorej transkripcia sa začala skúmať od roku 1990. Pri tomto type hudby je potrebné oddeliť jednotlivé homofónne zložky hudby, čo je mimoriadne zložité. Najkomplikovanejšie je oddeliť tóny, ktoré sa nachádzajú tesne pri sebe, pretože sa ťažko určuje, kde jeden tón končí a ďalší začína. Najrozsiahlejšiu analýzu v tejto oblasti urobil Klapuri v roku 2004 [4]. Podľa tejto analýzy je jednotlivé tóny možné oddeliť viacnásobnými odhadmi F_0 z audio krivky, pričom F_0 je najnižšia vlnová dĺžka a najnižšia frekvencia v periodickom priebehu. Problém je však v tom, že týmto odhadovaním vieme detegovať iba výšku tónu, ale jeho časová dĺžka zostane neznáma. Na zistenie trvania tónu je preto potrebné analyzovať rytmus danej hudby, čiže treba nájsť pravidelnosť v striedaní dĺžok jednotlivých tónov. Tieto techniky však ani zďaleka nie sú bezchybné a postačujúce na dokonalú transkripciu, pretože nezaručujú správne fungovanie pri rôznych šumoch a zmene rytmu v hudbe.

1.3.2 Na základe spracovávania obrazu

Na hudobnú transkripciu hry na klavíri pomocou spracovania obrazu bez informácie o hĺbke je zameraná táto práca, takže na rozdiel od zvyšných spôsobov riešenia načrtnutej problematiky bude tento spôsob prezentovaný detailnejšie. Doteraz sa tejto problematike venoval jeden vedecký článok [18], jedna diplomová práca [1] a jedna dizertačná práca [7].

V každej z týchto prác sa riešia rovnaké problémy. Prvým z nich je extrahovanie klaviatúry z jednotlivých snímok a detekcia bielych a čiernych kláves. Ďalším je odstránenie rúk hrajúceho na klavíri pomocou detekcie farby pokožky, avšak tento krok nie je vždy potrebný, keďže je možné skúmať len vrchnú časť klaviatúry, do ktorej by



Obr. 1.8: Rozdiel medzi snímkou, na ktorej nie je stlačený žiaden kláves a snímkou, na ktorej je stlačený A-durový akord po odstránení prstov zo snímok [18].

prsty nemali zasahovať. Nasledujúcou úlohou je samotná detekcia stlačených kláves, čiže zisťovanie rozdielov medzi jednotlivými snímkami a počiatočnou snímkou pozadia, na ktorej nie je stlačený žiaden kláves. Tento rozdiel môžeme vidieť na obrázku 1.8. Posledným problémom je riešenie výstupu transkripcie, ktorý je v týchto prácach riešený buď výstupom v notovej reprezentácii alebo vo formáte MIDI. Vo všetkých spomenutých prácach bolo spracovávané video natáčané nad klavírom. V takomto prípade je ideálne, aby bol vektor kamery približne kolmý na klaviatúru, ako je znázornené na obrázku 1.7.

Detekcia stlačených klavírových kláves vo videu

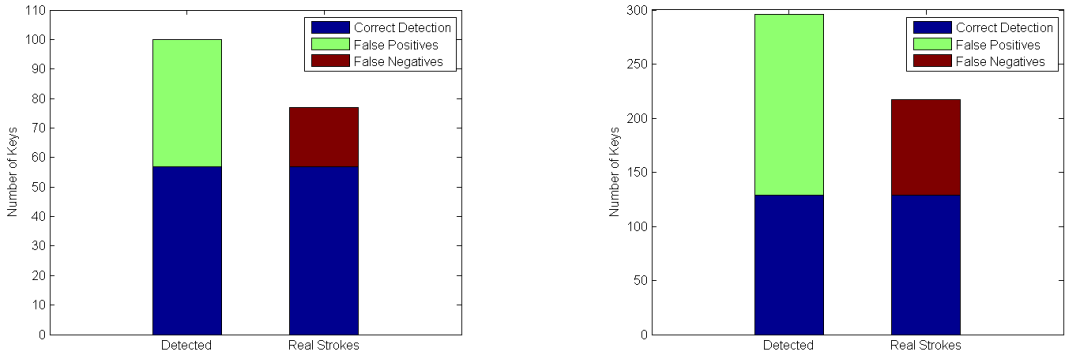
Ako prvý sa venoval tomuto problému pomocou počítačového videnia Suteparuk v roku 2013 [18]. Vo svojej práci autor vytvoril program v programovacom jazyku Matlab, ktorý nie je určený na transkripciu, ale iba deteguje stlačenie jednotlivých kláves pomocou ich grafického zvýraznenenia.

Program najskôr predspracuje video. V tomto predspracovaní ako prvé zvýrazní hrany Sobelovým gradientným operátorom, ktorý bol autorom vybraný z dôvodu najrýchlejšej detekcie hrán v porovnaní s inými technikami. Pre ešte lepšie zvýraznenie hrán sa následne aplikuje prahovanie. Na extrahovanie klaviatúry je použitá Houghova transformácia.

Ďalším krokom tohto programu je odstránenie rúk z klaviatúry. Táto problematika je tu riešená naučením sa farby pokožky na 10 snímkoch z videa. Následne je aplikovaná maska, ktorá ruky z jednotlivých snímkov odstráni.

Program ďalej deteguje čierne a biele klávesy, pričom čierne sa detegujú ako prvé, pretože je to jednoduchšie. V tomto kroku sa označí, kde sa nachádzajú čierne klávesy, pričom je potrebné myslieť na okraje jednotlivých kláves, ktoré môžu neskôr skresľovať detekciu kláves. Následne sa detegujú biele klávesy.

Detekcia stlačených kláves prebieha na základe zisťovania rozdielov medzi jednotlivými snímkami a snímkou pozadia. Autor uvažoval aj nad prípadom, v ktorom by sa počas nahrávania videa s kamerou mierne hýbalo, preto nepoužíval len konštantnú snímku pozadia. Namiesto konštantnej snímky pozadia využíval niekoľko predošlých snímok a zisťoval zmeny, ktoré sa udiali medzi týmito snímkami. V práci sú zverejnené výsledky z detekcie na dvoch videách. Obe majú snímkovú frekvenciu 30 snímkov za



(a) Skladba Canon in D

(b) Skladba Let It Go od Disney

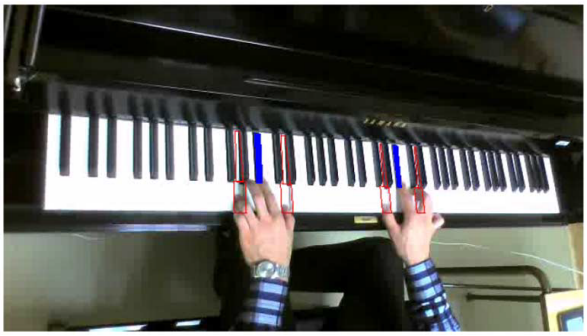
Obr. 1.9: Porovnanie úspešnosti pri rýchlejšej a pomalšej skladbe [18].

sekundu. Výsledky z testovania sú graficky znázornené na obrázku 1.9. Prvá skladba s názvom Canon in D má pomalšie tempo a druhá skladba Let It Go od Disney má tempo rýchlejšie. Pri prvom videu je správne detegovaných 74% stlačení, pri druhom je to 59.4%. Algoritmus však detegoval aj falošné stlačenia, čiže stlačenia, ku ktorým v skutočnosti nedošlo a pravdepodobne boli spôsobené šumom a tieňmi jednotlivých kláves. To znížilo jeho úspešnosť pri prvom videu na 63.3% a pri druhom na 43.6%. Využitím adaptívneho prahovania by bolo podľa vyjadrenia autora možné o niečo zvýšiť úspešnosť detekcie.

Vizuálna automatická hudobná transkripcia klavíra

Ako ďalší sa tejto problematike venoval vo svojej diplomovej práci Akbari v roku 2015 [1]. Akbari vytvoril softvér nazývaný ClaVision v programovacom jazyku *C#*, ktorý robí automatickú hudobnú transkripciu hry na klavíri v reálnom čase, pričom ako vstup použil video s minimálnym rozlíšením 320×240 pixelov a 24 snímkov za sekundu. Ako výstup tento softvér poskytuje MIDI alebo priamo notový záznam hudby. Popri automatickej hudobnej transkripcii aj graficky zvýrazňuje stlačené klávesy, ako je znázornené na obrázku 1.10.

Akbari [1] použil na detekciu stlačených kláves veľmi podobný postup, ako Suteparuk [18]. Hlavný rozdiel je v tom, že na detekciu hrán použil Cannyho metódu. Tá je výpočtovo náročnejšia, ale hrany dokáže detegovať spoľahlivejšie, čo sa ukázalo aj vo výsledkoch testovania. Okrem toho Akbari urobil aj analýzu, ako sa líši úspešnosť detekcie klaviatúry a stlačených kláves v závislosti od uhla kamery 1.11. Softvér mal dosahovať najpresnejšie výsledky, keď vektor pohľadu kamery a klaviatúra zvierali uhol 45 stupňov. Avšak pri tomto rozpoložení sa niektoré klávesy nepodarilo rozpoznať. Naopak, klávesy boli najúspešnejšie rozpoznané, keď bol daný uhol 0 stupňov 1.7. Najdokonalejšia presnosť by sa teda dala dosiahnuť použitím viacerých kamier, ktoré



Obr. 1.10: Grafická vizualizácia detekcie stlačenia kláves, pričom modrou farbou sú čierne klávesy a červenou farbou sú biele klávesy [1].

by klaviatúru snímali pod rôznymi uhlami.

Video	Angle	Keys Detection %	Recall %	Precision %	F_1 Score
V9	0	100.0	74.6	85.4	0.796
V10	30	94.9	66.7	93.3	0.778
V11	45	80.3	63.0	100.0	0.773
V12	60	63.9	40.7	64.1	0.498
V13	75	38.6	15.7	28.6	0.165

Obr. 1.11: Tabuľka úspešnosti rozpoznávania kláves v závislosti od uhla, ktorý zviera kamera s klaviatúrou. [1]

Transkripcia klavírovej hudby na základe počítačového videnia

Najčerstvejšie bola tomuto problému venovaná dizertačná práca od Mccaffreyho v roku 2017 [7]. Autor vytvoril softvér na automatickú hudobnú transkripciu hry na klavíri v programovacom jazyku C++ s použitím knižníc OpenCV na spracovanie obrazu a Midifile na vytváranie súborov vo formáte MIDI. Tento softvér má priemernú úspešnosť detekcie stlačených kláves 93.57% a celkovú presnosť 78.72%. Algoritmus na detekciu stlačených kláves je veľmi podobný algoritmu, ktorý používal Akbari [1], preto algoritmus dosahuje podobnú úspešnosť.

Kapitola 2

Návrh riešenia a implementácie

Cieľom našej práce je detekcia stlačených tónov na klavíri a zápis týchto tónov do formátu, s ktorým vedia pracovať počítače. V druhej kapitole zadefinujeme základné úlohy, ktoré je potrebné implementovať v našej bakalárskej práci. Zároveň v nej aj načrtneme spôsob, akým má aplikácia fungovať, čo má používateľovi ponúkať a ako má s používateľom komunikovať. Túto úlohu môžeme rozdeliť na niekoľko nasledujúcich podúloh:

1. Rozpoznanie klaviatúry a jednotlivých kláves.
2. Rozpoznanie stlačených kláves na základe rozdielov medzi jednotlivými snímkami.
3. Export záznamu stlačených kláves do súboru vo formátoch .midi alebo .musicxml.

Okrem vyššie popísaných podúloh má aplikácia ponúkať grafické používateľské rozhranie. Cez toto rozhranie má aplikácia umožniť používateľovi vložiť video ako vstup a dostať z nej požadovaný výstup. Ďalej má byť cez neho umožnené používateľovi kontrolovať a ovplyvňovať výsledok transkripcie manuálnym nastavením tempa, ktoré má byť vo výstupnom súbore a takisto aj transponovať hudbu do inej tóniny.

2.1 Návrh riešenia detekcie klaviatúry a kláves

Detekcia klaviatúry a kláves sa má robiť vždy len z jednej snímky. Má sa pritom počítať s tým, že kamera, ktorá sníma video, sa nebude pohybovať. Ak by sa aj kamera trochu hýbala, boli by spôsobené len jemné otrasy, voči ktorým má byť algoritmus odolný. Snímku má používateľ aplikácie spomedzi ostatných možných snímok určovať manuálne. Následne z nej má používateľ manuálne vystrihnúť oblasť v tvare obdĺžnika, v ktorej sa nachádza klaviatúra a označiť stred klaviatúry.

Rozpoznanie klávesov má prebiehať v dvoch krokoch, keďže treba rozpoznať osobitne čierne a osobitne biele klávesy. Postup je však pri oboch približne rovnaký a bude

sa líšiť až v implementácii. Pri oboch sa snímka najskôr vyhladí, aby obsahovala čo najmenej šumu. Nasleduje prahovanie, ktoré má byť buď optimálne alebo s prahom, ktorý manuálne zadá používateľ. Pomocou morfologických operácii následne môžme dosiahnuť minimalizáciu chýb, ktoré mohli byť spôsobené prahovaním. Po tomto kroku už môžme na snímke vyhľadať kontúry, ktoré sa ďalej spracujú. Keďže človek, ktorý hrá na klavíri zvykne mať takmer stále prsty na spodnom okraji niektorých bielych klávesov, biele klávesy nie je potrebné brať do úvahy celé, ale len do dĺžky čiernych klávesov. Týmto by sa mala zvýšiť aj efektívnosť programu. Nevýhodou tohto prístupu je, že pri prvotnom rozpoznávaní klávesov sa nám niektoré dvojice zlievajú do jednej, napríklad klávesy *B3* a *C4*, ako môžeme vidieť na obrázku 3.1. Tento nedostatok ľahko vyriešime rozdelením kontúry, ktorú získame pri segmentácii, na dve časti.

Keď máme všetky klávesy vysegmentované, priradíme k nim jednotlivé hodnoty tónov pomocou *MIDI* čísel. Toto priradenie realizujeme na základe pravidelného opakovania čiernych a bielych kláves na klaviatúre. Tým dosiahneme to, že priradenie bude fungovať všeobecne, nezávisle na tom, koľko klávesov daná klaviatúra obsahuje.

2.2 Návrh riešenia rozpoznania stlačených klávesov

Pri rozpoznávaní, ktoré klávesy boli stlačené, budeme pracovať so zmenami jasu klávesov. Počas rozpoznania jednotlivých klávesov na klaviatúre si zapamätáme, aký priemerný jas mal každý z klávesov. Následne pri každej snímke budeme porovnávať tento pôvodný priemerný jas s priemerným jasom snímok, na ktorých sa už hrá na klavíry. Ak je rozdiel medzi týmito jasmi väčší ako parameter zmeny, ktorý budeme určovať až v implementácii, budeme daný kláves považovať za stlačený. Zmeny jasu, ktoré budú spôsobené prstami, budeme ignorovať. Toto dosiahneme buď aplikovaním filtra, ktorý nám odstráni prsty na základe ich farby, alebo tým, že budeme brať do úvahy len tie časti snímok, kde je minimálna pravdepodobnosť, že by tam prsty mohli zasahovať.

2.3 Návrh riešenia záznamu o stlačených klávesoch

Keď rozpoznáme, že nejaký kláves bol stlačený, uložíme si informácie o tomto stlačení. Týmito informáciami je *MIDI* číslo klávesu a čas, kedy bol stlačený. Keď rozpoznáme, že daný kláves je už pustený, z informácii o stlačení získame dĺžku stlačenia. Následne si uložíme informácie o zahranom tóne, čiže jeho *MIDI* číslo, dĺžku hrania a čas, kedy bol zahraný a pôvodné informácie o stlačení odstránime, keďže ich už nebudeme potrebovať. Z týchto informácii následne budeme vedieť vytvoriť *.midi* alebo *.musicxml* súbor.

Kapitola 3

Implementácia

Táto kapitola je venovaná implementácii nášho riešenia, ktoré sme navrhli v predošlej kapitole. Na začiatku kapitoly uvedieme technológie, ktoré sme počas implementácie využívali. V ďalších častiach tejto kapitoly budeme podrobne popisovať princípy, na ktorých pracuje aplikácia, ktorú sme naprogramovali. Výsledky a zhodnotenie implementácie sa nachádza v ďalšej kapitole tejto práce.

3.1 Použité technológie

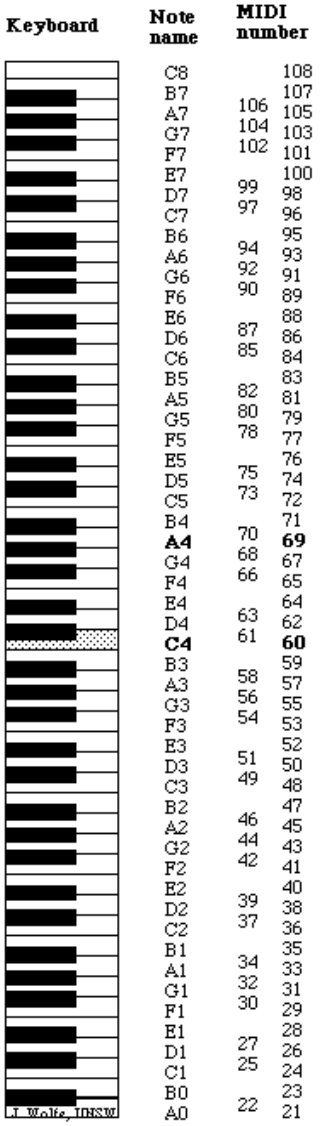
Python

Python [12] je interpretovaný a dynamicky typovaný programovací jazyk, ktorý vytvoril Guido van Rossum a jeho počiatky siahajú do roku 1990. Keďže je to interpretovaný jazyk, príkazy vykonáva priamo za sebou bez toho, aby sa celý program musel na začiatku skompilovať do strojového kódu. Tento jazyk je tiež interaktívny, takže program môžeme upravovať počas jeho behu. Python podporuje štruktúrované, funkcionálne a objektovo orientované programovanie. V tejto práci bol využívaný vo verzii Python 3.8.1. Bežnou súčasťou tohto programovacieho jazyka je knižnica na vytváranie používateľského rozhrania TkInter, ktorá v našej práci bola použitá na komunikáciu medzi používateľom a aplikáciou. Pri implementácii sme využívali vývojové prostredie PyCharm. Toto vývojové prostredie obsahuje *debugger*, čo nám veľmi pomáhalo pri hľadaní a odlaďovaní chýb, keďže sme si mohli jednotlivé časti programu pohodlne odkrokovať.

OpenCV

OpenCV (Open source computer vision) [11] je knižnica s otvoreným zdrojovým kódom, ktorá obsahuje funkcie na spracovanie obrazu v reálnom čase. Je napísaná v jazyku C++, ale je podporovaná aj jazykmi Java a Python. V našej práci bola

využívaná vo verzii OpenCV 4.1.1.



Obr. 3.1: Názvy nôt, MIDI čísla a frekvencie [7].

MIDI

MIDI (Musical Instrument Digital Interface) je rozhranie, ktoré bolo štandardizované v roku 1983. Slúži na prenos a zaznamenávanie dát o hudbe, ktoré v *MIDI* formáte obsahujú informácie, ako bola nejaká melódia zahraná. Následne je možné z *MIDI* záznamu túto melódiu opätovne zreprodukovať. MIDI-protokol je v dnešnej dobe podporovaný vo väčšine elektrických hudobných nástrojov a tiež vo väčšine zvukových kariet. MIDI je podporované aj vo veľkom množstve softvérov na prácu s hudbou, napríklad v Sibeliuse alebo MuseScore. Na obrázku 3.1 sú znázornené prislúchajúce *MIDI* čísla k jednotlivým klávesom.

MusicXML

MusicXML je formát veľmi podobný *MIDI* s tým rozdielom, že je založený na XML (eXtensible Markup Language) a slúži na ukladanie informácii o postupe zahrania hudby. V porovnaní s MIDI je o niečo novší a vo väčšine softvérov na prácu s hudbou je taktiež podporovaný.

Knižnica MIDIUtil

MIDIUtil [8] je knižnica v jazyku Python na tvorbu súborov vo formáte *MIDI*. V tejto práci bola použitá vo verzii *MIDIUtil 1.2.1*.

Knižnica music21

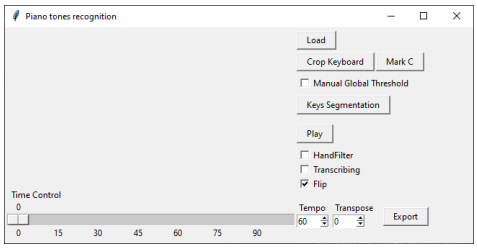
Music21 [9] je knižnica v jazyku Python, ktorá slúži na vytváranie a spracovávanie zvuku. Poskytuje funkcie na pomoc s prácou vo formátoch *MIDI*, *MusicXML* a v mnohých ďalších. Rovnako tak poskytuje aj vizualizáciu hudby pomocou knižnice Matplotlib. V našej práci bola táto knižnica využitá vo verzii *music21 5.7.2*.

3.2 Používateľské rozhranie

Pri implementácii používateľského rozhrania sme na komunikáciu medzi používateľom a aplikáciou použili knižnicu Tkinter [19]. Okrem nej sme ako pomocnú knižnicu na komunikáciu medzi knižnicami OpenCV a Tkinter použili knižnicu PIL (Python Imaging Library), keďže knižnica Tkinter nevedela zobrazovať snímky priamo z matíc, ktoré sa používajú na reprezentáciu obrazu v knižnici OpenCV.

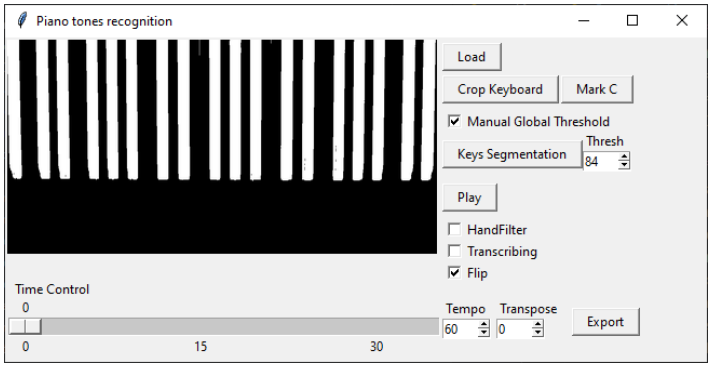
Aplikácia na komunikáciu s používateľom prostredníctvom Tkinter používa štítky (*label*), tlačidlá (*button*), označovacie tlačidlá (*checkbutton*), počítadlá (*spinbox*), mierky (*scale*), plátno (*canvas*) a dialógové správy a okná, ktoré slúžia na vybratie súboru, z ktorého sa číta a do ktorého sa zapisuje. Aplikácia na komunikáciu s používateľom používa tiež nástroje z knižnice OpenCV, ale v tomto prípade ide len o nástroje na manuálne vystrihnutie klaviatúry a manuálne určenie stredného klávesu klaviatúry. Pre lepšiu organizáciu kódu sme pre používateľské rozhranie vytvorili tri osobitné triedy, ktoré sú v samostatných moduloch. Prvá z týchto tried manažuje všetky prostriedky knižnice Tkinter, takže prostredníctvom tejto triedy používateľ ovláda celú aplikáciu. Ďalšie dve triedy sú pomocné, pričom jedna z nich slúži na vystrihnutie klaviatúry a druhá na určenie stredného klávesu klaviatúry.

Po spustení samotnej aplikácie sa zobrazí hlavné okno používateľského rozhrania, ako môžme vidieť na obrázku 3.2. Na načítanie videa treba, aby používateľ klikol na



Obr. 3.2: Okno, ktoré sa zobrazí hneď po spustení aplikácie.

tlačidlo *Load* a následne cez dialógové okno vybral videosúbor. Ak bolo video natáčané z opačného pohľadu, ako je pohľad človeka hrajúceho na klavíri, je potrebné zaškrtnúť označovacie tlačidlo *Flip*, pretože aplikácia spracúva snímky z pohľadu človeka hrajúceho na klavíri. Ďalej je potrebné vystrihnúť klaviatúru pomocou tlačidla *Crop Keyboard* a to tak, že sa stlačením, podržaním a pustením myšky kurzorom vystrihneme obdĺžnik, v ktorom sa klaviatúra nachádza. Po vystrihnutí klaviatúry je potrebné označiť stred klaviatúry pomocou tlačidla *Mark C* a kliknúť na biely kláves, ktorý sa nachádza v strede klaviatúry. Nasleduje spustenie rozpoznania jednotlivých kláves stlačením tlačidla *Keys Segmentation*. V niektorých prípadoch je potrebné pred rozpoznaním jednotlivých kláves prejsť na snímku, na ktorej sa nachádza len klaviatúra bez rúk pomocou mierky *Time Control*. Pri rozpoznávaní kláves je predvolene využívané optimálne prahovanie pomocou algoritmu Otsu, no aplikácia ponúka aj manuálne nastavenie globálneho prahu. Pred týmto nastavením je však potrebné označiť označovacie tlačidlo *Manual Global Threshold*. Ako reakcia na to sa zobrazí počítadlo *Thresh*, na ktorom je možné prah manuálne zvoliť, a to medzi hodnotami 50 až 200, pričom predvolená hodnota počítadla je 85. Túto funkciu však vo väčšine prípadov nie je nutné použiť, keďže v tomto prípade optimálne prahovanie funguje dostatočne spoľahlivo. Svoje využitie nájde vtedy, ak by používateľa zaujímal prah medzi čiernymi a bielymi klávesami, keďže pri každej zmene hodnoty na počítadle *Thresh* sa zobrazí vyprahovaná aktuálna snímka, na ktorej môže používateľ vidieť, ako na daný prah reaguje.



Obr. 3.3: Na obrázku je zobrazená situácia manuálneho nastavovania prahu.

Pred spustením samotnej transkripcie treba označiť označovacie tlačidlo *Transcribing*. Po tomto kroku sa transkripcia spúšťa tlačidlom *Play*, ktoré sa hneď po stlačení premenuje na *Stop*, takže pre zastavenie transkripcie je ho potrebné znovu stlačiť. Za účelom minimalizovať detekciu falošných stlačení klávesov, ktoré môžu byť spôsobené prstami, je možné pri transkripcii aplikovať filter na odstránenie rúk. Tento filter zapíname označovacím tlačidlom *HandFilter*.

Po vykonaní transkripcie aplikácia ponúka export zaznamenaných stlačených kláves vo formáte *MIDI* alebo musicxml. Pre export je potrebné kliknúť na tlačidlo *Export*. Po kliknutí vyskočí dialógové okno, v ktorom je potrebné zadať typ výstupného súboru a jeho názov. Pred samotným exportom používateľ môže nastaviť tempo danej skladby pomocou počítadla *Tempo* a rovnako tak môže skladbu aj transponovať pomocou počítadla *Transpose* do inej tóniny. Tempo sa udáva v počte úderov za minútu, pričom jeden úder pri našej implementácii zodpovedá jednej sekunde v reálnej hudbe. Minimálne možné tempo je 20 úderov za minútu a maximálne je 150 úderov za minútu. Záznam pre export sa dá transponovať po jednom poltóne, pričom transponovanie je možné o 10 poltónov oboma smermi.

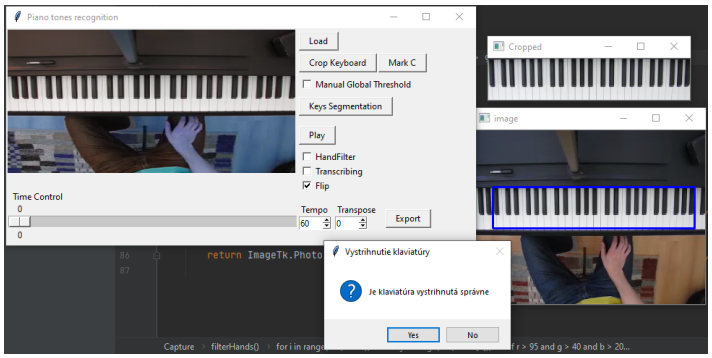
3.3 Získavanie snímok

Na získavanie jednotlivých snímok sme vytvorili triedu *Capture* odvodenú od triedy *VideoCapture* z knižnice OpenCV, pričom počas celého behu aplikácie sa využíva len jedna inštancia tejto triedy. Trieda bola týmto spôsobom navrhnutá pre jej vysokú flexibilitu, nech je do nej možné v budúcnosti jednoducho dorobiť ďalšie potrebné funkcie, napríklad aby namiesto načítavania jednotlivých snímok z videosúboru načítavali snímky z kamery, ktorá by bola pripojená k počítaču. Táto trieda ponúka jednotlivé snímky v potrebných formátoch a veľkostiach buď pre metódy z OpenCV alebo pre TkInter prostredníctvom knižnice PIL. Okrem toho tiež implementuje odfiltrovanie rúk zo snímky, čo je možné využívať pri transkripcii.

3.4 Rozpoznanie klaviatúry a klávesov

Rozpoznávanie klaviatúry prebieha manuálne. Spúšťa sa ako inštancia triedy *Cropper*, pričom je potrebné ručne vystrihnúť klaviatúru, ako môžeme vidieť na obrázku 3.4. Po jej vystrihnutí je potrebné potvrdiť správnosť jej vystrihnutia. Následne treba označiť biely prostredný kláves celej klaviatúry, čo sa spúšťa ako inštancia triedy *Marker*.

Samotné rozpoznanie kláves spočíva v niekoľkých krokoch. Najskôr sa rozpoznávajú čierne klávesy a následne sa rozpoznávajú biele klávesy. Po rozpoznaní klávesov je



Obr. 3.4: Ukážka vystrihnutia klaviatúry.

potrebné ku klávesom priradiť, aké tóny ktoré klávesy reprezentujú.

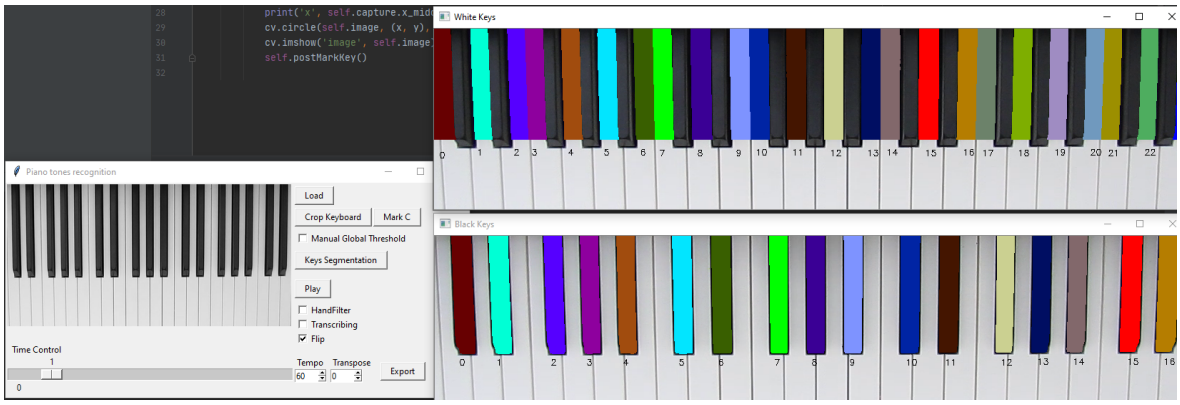
Na rozpoznávanie kláves sme vytvorili tri triedy. Prvá z nich sa nazýva *Segmentation* a obsahuje všeobecnejšie metódy, ktoré sa používajú pri rozpoznaní čiernych aj bielych klávesov. Ďalšie dve sa nazývajú *SegmentBlack* a *SegmentWhite*. Ich inštancie vykonávajú už konkrétnu segmentáciu čiernych a bielych klávesov, pričom obe sú spúšťané z triedy *Segmentation*. Keďže rozpoznávanie jednotlivých kláves nebolo potrebné robiť pre každú snímku, ale iba pre snímku pozadia, implementácia tohto rozpoznávania bola zameraná na presnosť rozpoznávania a nie na časovú efektivitu a zložitosť. Pri oboch sme však spracovávali snímku v čiernobielom formáte, pretože klaviatúra obsahuje štandardne čierne a biele klávesy, takže sme si vystačili s odtieňmi šedej farby. Túto snímku sme vždy najskôr vyhladili gaussovským filtrom, aby sme eliminovali náhodný šum. Gaussovský filter sme použili z viacerých dôvodov. Ide napríklad o zachovanie pôvodných hrán v obraze a o vynikajúcu prípravu obrazu na prahovanie, keďže pri prahovaní nás zaujímajú spojité oblasti obrazu, pričom gaussovský filter nám práve túto spojitosť zabezpečuje.

Rozpoznávanie čiernych klávesov prebieha ako prvé a to z viacerých dôvodov. Hlavným dôvodom je to, že pri ňom získame informáciu o dĺžke čiernych klávesov, ktorú následne využijeme pri rozpoznávaní bielych klávesov. Ďalším dôvodom je to, že rozpoznanie čiernych klávesov je jednoduchšie, ako rozpoznanie bielych. V prvom kroku sa vyprahuje snímka pozadia, a to buď optimálnym prahovaním Otsu, alebo využitím prahu daného používateľom. Toto prahovanie je zároveň inverzné, takže po jeho aplikovaní máme na snímke čierne klávesy biele a zvyšok snímky zostane čierny. Následne sa na snímku aplikuje operácia dilatácie, čím sa vyplnia prípadné diery, ktoré mohli vzniknúť prahovaním v predošlom kroku. Tieto diery bývajú spôsobené svetlom, ktoré v niektorých častiach klávesov vytvára svetlejšie plochy. Po tomto kroku aplikujeme eróziu v dvoch iteráciách, aby sme klávesy stenčili a dostali do pôvodnej veľkosti, pričom klávesy môžu ostať aj o niečo tenšie, čím sa na klávesoch vykoná morfologické uzatvorenie. Pri dilatácii a erózii sme využívali masku veľkosti 5×5.

Ďalším krokom je spustenie funkcie, ktorá nám vyhľadá všetky kontúry, čiže čierne

klávesy. Na tieto kontúry aplikujeme filter, ktorý nám odstráni kontúry s menšou plochou ako 0,6-násobok priemernej plochy všetkých nájdených kontúr. Zabezpečíme si tým to, že nebudeme ďalej pracovať s kontúrami, ktoré boli chybne rozpoznané a v skutočnosti nie sú čiernymi klávesmi. Po tomto kroku usporiadame kontúry do zoznamu podľa súradnice x ich ľavého spodného rohu. Následne ich rozdelíme do dvoch zoznamov. V prvom zozname budú kontúry, ktoré sa nachádzajú vľavo od stredného klávesu a v druhom budú zvyšné. Pri delení zachováme ich predošlé poradie, pričom delenie robíme tiež podľa súradnice x ich ľavého spodného rohu. Stredný kláves sme v tomto kroku zaradili do ľavého zoznamu. Po týchto krokoch môžeme ku kontúram priraďovať konkrétne klávesy, čiže MIDI čísla. Priraďovanie prebieha osobitne pre ľavý a osobitne pre pravý zoznam. Ako už bolo spomínané v návrhu, pri implementovaní priraďovania využívame opakujúce sa vzory v striedaní bielych a čiernych klávesov na klaviatúre. Pri tomto priraďovaní sme zmenili spôsob ukladania klávesov. Namiesto ukladania kontúry pre každý kláves sme začali ukladať všetky body, ktoré kontúra pokrýva. Tento spôsob je náročnejší na pamäť, ale určite by nemal spôsobovať problémy. Výhoda ukladania všetkých bodov jednotlivých klávesov spočíva v tom, že ich nemusíme pri každej snímke osobitne vypočítavať z kontúr, čo sa výrazne odráža na využívaní výpočtovej kapacity. Aby sme ušetrili ešte o niečo viac, body jednotlivých klávesov sme okresali o tie, ktoré sa nachádzajú v spodnej časti klávesov, kde sa často môžu nachádzať prsty. Týmto sme znížili potrebu využívania filtra prstov a rúk.

Posledným krokom pri segmentácii čiernych klávesov je zistenie a uloženie ich priemerného jasu. V tomto kroku počítame aritmetický typ priemeru, čiže spočítame hodnoty jasov všetkých bodov a následne ich vydelíme ich počtom. Tieto informácie neskôr využijeme pri detekcii stláčania klávesov.



Obr. 3.5: Ukážka segmentácie čiernych a bielych klávesov.

Pri segmentácii bielych klávesov sme použili veľmi podobný postup, preto pri vysvetľovaní jednotlivých krokov nepôjdeme úplne do detailov a spomenieme hlavne tie, ktoré boli rozdielne. Hlavná odlišnosť medzi oboma segmentáciami je v tom, že sme nespracovávali celú snímku s klaviatúrou, ale len jej vrchnú časť, na ktorej sa nachádzajú

aj čierne klávesy. To znamená, že biele klávesy sme nevysegmentovali úplne celé, ako je to znázornené na obrázku 3.5, a to z dôvodu rýchlejšej a jednoduchšej segmentácie. Súradnicu y, podľa ktorej určujeme vrchnú časť, sme odvodili z dĺžky čiernych klávesov, ktorú sme získali počas ich segmentácie. Ďalej nasleduje prahovanie - rovnaké, ako pri segmentácii čiernych klávesov - avšak tento raz nie inverzné, keďže chceme získať biele oblasti s bielymi klávesmi. Následne aplikujeme morfologické uzatvorenie, vyhľadáme kontúry a rovnako, ako pri segmentácii čiernych klávesov, z nich odfiltrujeme tie, ktoré sú podozrivo malé, takže pravdepodobne klávesmi nebudú. Kontúry s väčšou plochou ako 1,4-násobok priemernej plochy všetkých nájdených kontúr vertikálne rozdelíme na dve polovice. Týmto dosiahneme dokončenie segmentácie aj tých klávesov, ktoré sa nám pri prahovaní zliali do seba, napríklad klávesy *B3* a *C4*. Uloženie bodov jednotlivých klávesov a ich priemerných jasov prebieha podľa rovnakých princípov, ako to bolo pri čiernych klávesoch.

3.5 Rozpoznanie stlačených klávesov

Rozpoznávanie stlačených klávesov robí náš program v hlavnej triede. Pri tomto procese využívame priemerný jas každého klávesu, ktorý sme si uložili počas segmentácie. Počas hrania na klavíri dochádza k tomu, že po stlačení čierneho klávesu je tento kláves o niečo svetlejší a biely o niečo tmavší. Tento jav využívame na to, aby sme detegovali stlačenie jednotlivých klávesov. Keďže sme si pri segmentácii uložili pre každý kláves jeho jednotlivé body, teraz môžeme jednoduchým spôsobom vypočítať, aký je priemerný jas každého klávesu počas hry na klavíri. Ten následne porovnáme s jasom, ktorý sme vypočítali počas segmentácie. Pri čiernych klávesoch považujeme za stlačenie to, keď sa priemerná hodnota jasu zmení aspoň o 5 a pri bielych aspoň o 4. Za pustenie klávesu teda pokladáme, ak je zmena priemernej hodnoty jasu menšia. Pri rozpoznávaní stlačených klávesov je možnosť využiť filter rúk. V takomto prípade sa body, ktoré pokrýva ruka alebo prsty, neberú pri počítaní priemeru jasu jednotlivých klávesov do úvahy.

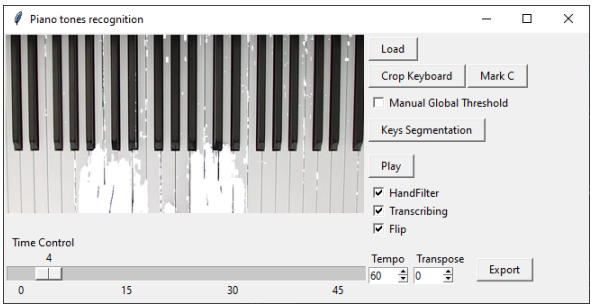
3.6 Filter rúk a prstov

Pri vytváraní filtra pre pokožku sme sa inšpirovali prácou, ktorá bola venovaná detekcii ľudskej pokožky v reálnom čase [10]. Filter sme implementovali iba pomocou niekoľkých podmienok pre farbu pokožky. Využívali sme pri tom *RGB* model. Podmienky zložiek jednotlivých pixlov boli nasledovné:

- $r > 95$
- $g > 40$

- b > 20
- r > g
- r > b

Uvedené podmienky sú veľmi zjednodušené, čo sa odrazilo na výsledkoch filtra, ktoré nie sú ani zďaleka dokonalé. Môžeme to vidieť na obrázku 3.6. Vo väčšine prípadov však filter vôbec nie je potrebný, keďže sme sa snažili rozpoznávanie stlačených kláves implementovať tak, aby bola minimálna pravdepodobnosť, že do skúmaných oblastí budú prsty zasahovať.



Obr. 3.6: Ukážka použitia filtra rúk. Pokožka je zvýraznená bielou farbou.

3.7 Vytváranie záznamu o stlačených klávesoch a export

Na zaznamenávanie stlačených kláves pre účely tejto práce používame viacero dátových štruktúr. Ako pomocné dátové štruktúry pri stlačení a pustení klávesu používame slovníky a množiny. Po rozpoznaní toho, že kláves je na jednej snímke stlačený, sa táto udalosť uloží do slovníka, pričom ako kľúč sa používa *MIDI* číslo klávesu a ako hodnota počet sekúnd od začiatku videa. Tento čas sa získava z triedy *Capture*, takže to nie je reálny čas, ako dlho je už video spustené, ale čas, ako dlho by bolo video spustené za podmienok, že by nebolo ničím spomaľované. V našom prípade je video spomaľované výpočtami na rozpoznávanie stlačených klávesov. Tento záznam v slovníku o stlačení klávesu sa v ňom uchováva až do jeho pustenia. Po jeho pustení sa záznam zo slovníka odstráni. Do zoznamu, ktorý patrí hlavnej triede celej aplikácie, sa následne pridá n-tica, v ktorej sú údaje o note, ktorá sa na konci zapíše do *MIDI* alebo *musicxml* súboru. Do tejto n-tice patrí *MIDI* číslo klávesu, čas od začiatku videa v sekundách a dĺžka trvania stlačenia klávesu v sekundách, ktorá sa počíta ako rozdiel času pustenia a stlačenia klávesu.

Vytvorenie súboru zo záznamu vo formáte *MIDI* alebo *musicxml* sa vykonáva v triede *Export*. Táto trieda dostáva ako jeden zo vstupných parametrov zoznam s n-ticami, ktoré reprezentujú stlačené klávesy, ako je bližšie popísané v predošlom odseku.

Ďalšími vstupnými parametrami tejto triedy sú názov súboru, tempo a transponovanie, pričom každý z týchto parametrov pochádza z triedy *Gui*, čiže používateľ ich zadáva cez používateľské rozhranie.

Prvým krokom pri vytváraní inštancie triedy *Export* je rozhodnutie, či názov súboru obsahuje príponu *.midi* alebo *.musicxml*. Ak neobsahuje ani jednu, program by mal vyhodiť výnimku, avšak tento stav by nikdy nemal nastať, pretože používateľské rozhranie je implementované tak, aby používateľ zvolil jeden z formátov súborov na export.

V prípade exportu do *MIDI* formátu sa využíva na vytvorenie výstupného súboru trieda *MIDIFile* z knižnice midiutil. Po vytvorení inštancie tejto triedy jej treba nastaviť tempo, ktoré je jeden z jej atribútov. Ďalej treba preliezť všetky n-tice so zaznamenanými stlačenými klávesmi v jednom cykle. Každá jedna iterácia reprezentuje jeden tón, ktorý pridáme do inštancie triedy *MIDIFile* s tým, že každému tónu nastavíme výšku, dĺžku jeho trvania a jeho začiatok, čiže čas, ktorý by ubehne od pustenia hudby až po realizáciu daného tónu. Pri nastavení výšky tónu aplikujeme transponovanie, ktoré si používateľ zvolil.

Kapitola 4

Vyhodnotenie

V tejto kapitole budeme vyhodnocovať našu implementáciu. Na začiatku budeme prezentovať, aký bol čas vykonávania jednotlivých metód našej aplikácie. Tieto časy sme merali pri rôznych rozlíšeniach. Následne ukážeme úspešnosť implementácie z hľadiska presnosti a taktiež sa budeme venovať aj jej slabším stránkam.

4.1 Testovacie videá

Videá, ktoré sme používali pri testovaní našej aplikácie, sme natáčali pomocou viacerých zariadení. Uvedieme si tri z nich spolu s ich maximálnymi rozlíšeniami:

- Canon PowerShot SX10IS - 640 × 480 (30 fps)
- Fujifilm FinePix XP10SE - 1280 × 720 (30 fps)
- GoPro HERO6 - 3840 × 2160 (30 fps)

Obr. 4.1: Ukážka spôsobu získavania testovacích videí

Videá sme natáčali na elektrickom klavíri Yamaha YDP-144, ako to môžeme vidieť na obrázku 4.1. Tento klavír mal klávesy zo syntetickej slonoviny, čo malo výhodu v

tom, že klávesy boli viac podobné tým, ktoré sa používajú na bežných neelektrických klavíroch, napríklad na krídle alebo na pianine. Videá sme skúšali natočiť aj na iných elektrických klavíroch s obyčajnými plastovými klávesmi, ale spôsobovalo to problémy pri ich rozpoznávaní, pretože boli veľmi lesklé.

4.2 Čas vykonávania jednotlivých metód

Pri testovaní aplikácie sme robili merania, ako dlho trvajú jednotlivé kroky programu. Podľa našich očakávaní trvali najdlhšie tie časti, pri ktorých sa spracúvali snímky. Vykonali sme teda merania, koľko trvala segmentácia bielych a čiernych klávesov. Taktiež sme odmerali, koľko trvá spracovanie jednej snímky pri detekcii stlačených klávesov. Tieto merania sme skúšali pri viacerých rozlíšeniach vstupného videa, pričom pri meraní sme nemali zapnutý filter rúk. Výsledky meraní sme si zaznamenali v sekundách a nachádzajú sa v tabuľke 4.1.

Tabuľka 4.1: Čas vykonávania metód v aplikácii v závislosti od rozlíšenia vstupného videa.

Rozlíšenie	960×540	1280×720	1920×1080
Segmentácia čiernych klávesov	4.62s	9.25s	20.13s
Segmentácia bielych klávesov	8.57s	14.58s	43.49s
Detekcia stlačenia čiernych klávesov	0.16s	0.31s	0.69s
Detekcia stlačenia bielych klávesov	0.27s	0.45s	1.06s

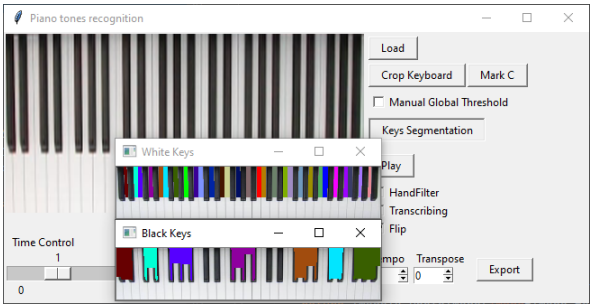
Z tabuľky 4.1 môžeme vyvodiť, že spracúvanie bielych klávesov trvalo výrazne viac, než spracúvanie čiernych. Tento jav je spôsobený viacerými faktormi. Najhlavnejší je ten, že bielych klávesov je viac, než čiernych. Pri segmentácii je ďalším faktorom vykonávanie viacerých operácii pri spracúvaní bielych klávesov než pri spracúvaní čiernych. V prípade čiernych sa nám všetky klávesy vysegmentovali počas prahovania, ale pri bielych sa nám niektoré dvojice klávesov do seba zliali. To znamená, že sme museli zistiť, o ktoré klávesy sa presne jedná a rozdeliť ich. Všeobecne nám pri segmentácii neprekážalo, ak trvala dlhšie, keďže sa robila iba na začiatku spracúvania videa. Na nameraných údajoch pri rozpoznávaní stlačených klávesov môžeme pozorovať, že tieto operácie trvali veľmi dlho vzhľadom k ich vykonávaniu pri každej snímke. Z toho sme usúdili, že našu aplikáciu by nebolo možné používať na rozpoznávanie tónov v reálnom čase, keďže už pri rozlíšení 960×540 trvalo spracovanie jednej snímky približne 0,6 sekundy. Tento problém sme sa počas implementácie pokúšali vyriešiť paralelizáciou. Náš pokus bol však neúspešný, keďže sme pri ňom narazili na obmedzenia programovacieho jazyka Python. Pri použití viacerých vlákien by bol problém v tom, že vlákna v Pythone sú implementované tak, že dokážu bežať len asynchónne, nie však skutočne

paralelne, aby využívali viacero jadier procesora. Alternatívou, ako využiť viacero jadier procesora na výpočty v jazyku Python, bolo využiť knižnicu *multiprocessing*, cez ktorú je možnosť vytvárať procesy v operačnom systéme. Tu sme zas narazili na iné obmedzenia, a to na problém viazanosti hlavnej triedy na grafické rozhranie realizované knižnicou *tkinter*, ktorá bránila vytvoreniu synovského procesu. Okrem toho je vytváranie procesov časovo náročnou záležitosťou, preto by sme tým s vysokou pravdepodobnosťou žiadne zrýchlenie nedosiahli. Lepšie riešenia nášho problému sú popísané v poslednej časti tejto kapitoly.

4.3 Úspešnosť segmentácie klávesov

Pri testovaní segmentácie jednotlivých klávesov sme vyskúšali videá s rôznymi rozlíšeniami a svetelnými podmienkami. Pri dostatočne veľkom rozlíšení a dobrých svetelných podmienkach sa klávesy segmentovali dokonale bez akejkoľvek chyby za predpokladu, že klaviatúra bola vystrihnutá správne, pričom sa jednalo o rozlíšenia od 640×480. Pri horších svetelných podmienkach so slabším osvetlením sme klávesy vedeli vysegmentovať takmer dokonale, s tým rozdielom, že sme nemohli použiť optimálne prahovanie, pretože prah sa neurčil dostatočne presne. Pri manuálnom nastavení prahu sa však segmentácia podarila takmer rovnako presne, ako pri lepšom osvetlení s optimálnym prahovaním.

Naše riešenie malo pri segmentácii výraznejšie problémy pri nízkom rozlíšení, u ktorého dochádzalo k tomu, že niektoré čierne klávesy sa zliali do seba, ako to môžeme vidieť na obrázku 4.2. Problémy sa prejavovali od rozlíšenia 320×240 smerom k nižšiemu. S najväčšou pravdepodobnosťou boli spôsobené jednou z morfologických operácii, konkrétne dilatáciou, pri ktorej sa odstránili medzery medzi klávesmi, čo spôsobilo ich zliatie sa do seba.



Obr. 4.2: Ukážka testovania segmentácie klávesov z videa s rozlíšením 320×240.

4.4 Úspešnosť rozpoznávania stlačených klávesov

Testovanie úspešnosti rozpoznávania stlačených klávesov sme vykonávali prevažne stupnicami. Stupnica je rad tónov zoradených v stúpajúcom alebo klesajúcom smere podľa určitých pravidiel [**droppov\IeC {\'a}1998z\IeC {\'a}klady**]. Vybrali sme stupnicu C dur, pri ktorej sa hrá len na bielych klávesoch a stupnicu H dur, pri ktorej sa hrá skoro vždy iba na čiernych klávesoch. Pri testovaní bolo úspešnejšie rozpoznávanie stlačenia bielych klávesov, než čiernych. Bolo to spôsobené hlavne tým, že po stlačení čiernych klávesov boli zmeny jasu menej výrazné v porovnaní so stlačením bielych. Pri čiernych klávesoch by bolo problematické spozorovať zmenu aj voľne ľudským okom. Potenciálne riešenie tohoto problému je bližšie popísané v poslednej časti tejto kapitoly.

Naše riešenie malo za určitých podmienok problém aj s rozpoznávaním stlačenia bielych klávesov. Išlo o detekciu falošných stlačení, čiže náš program rozpoznal stlačenie klávesa, ku ktorému v skutočnosti nedošlo. Bolo to spôsobené stláčaním čiernych klávesov v jeho okolí, čo spôsobilo zmenu jasu aj na ňom napriek tomu, že stlačený nebol. Potenciálne riešenie tohto problému je spomenuté taktiež v poslednej časti tejto kapitoly.

4.5 Možnosti ďalšieho výskumu

V priebehu tvorby tejto práce nám napadlo veľa podnetov a vylepšení, ako by bolo možné dosiahnuť v danej problematike ešte lepšie výsledky, pričom pre nedostatok času sa nám nepodarilo všetky z nich implementovať. Niektoré boli sčasti spomenuté už v kapitole o implementácii alebo vo vyhodnotení, avšak komplexne ich zhrnieme v nasledujúcich odsekoch.

Keďže program vykonával niektoré výpočty veľmi pomaly a pri nízkych rozlíšeniach dobre nefungoval, do budúcna by určite bolo vhodné implementovať takú funkcionalitu, aby aplikácia vedela správne fungovať aj s nižším rozlíšením. Výpočty by sa dali zrýchliť aj vytvorením efektívnejšieho kódu a využitím iného programovacieho jazyka. Do úvahy by prichádzali napríklad jazyky Java, C++ a Go, keďže všetky podporujú využívanie vlákien, ktoré dokážu bežať skutočne paralelne. Všetky tieto jazyky podporujú OpenCV, podobne ako Python. Vytvorenie vlákna však tiež nie je úplne časovo nenáročná záležitosť, takže do budúcna by bolo lepšie zo všetkých vymenovaných jazykov uprednostniť jazyk Go a na paralelizmus využiť gorutiny, ktoré vedia bežať paralelne ako vlákna. Gorutiny majú výhodu v tom, že sa vytvárajú oveľa rýchlejšie ako vlákna a zaberajú menej pamäte.

Obraz, ktorý sme spracovávali, niekedy neobsahoval dostatočne veľa informácii na to, aby sme vedeli spoľahlivo rozpoznať stlačenie jednotlivých klávesov. Do budúcna by bolo vhodné využiť viacero typov vstupných dát. Do úvahy by prichádzala 3D kamera,

použitie viacerých klasických 2D kamier, alebo spracovávanie zvuku popri spracovávaní obrazu. Posledná spomenutá možnosť by v tomto prípade mohla výrazne zvýšiť úspešnosť, keďže na základe zvuku by sa dalo vždy veľmi dobre verifikovať, či boli určité klávesy skutočne stlačené.

Pri rozpoznávaní stlačenia klávesov na základe zmien jasu je do budúcna určite potrebné brať do úvahy celé skupiny klávesov, keďže ich jasy sa vedia vzájomne ovplyvňovať. Zároveň ich vie ovplyvňovať aj človek hrajúci na klavíri, ak lúče svetla neprichádzajú len spred klavíra. Z tohto dôvodu bude treba vymyslieť všeobecnejšie riešenie, ktoré by si vedelo poradiť aj s klamlivými zmenami jasu. Na tento účel by bolo vhodné využiť niektorú z neurónových sietí. Neurónová sieť by sa do budúcna dala využiť aj na automatické vystrihnutie klaviatúry na základe určitých príznakov, aby to nemusel vykonávať používateľ manuálne.

Úspešnosť rozpoznávania stlačených klávesov by sa dala vylepšiť aj tak, že by pohľad kamery zvieral s klaviatúrou menší uhol, než 90 stupňov, čo by však mierne bránilo segmentácii klávesov. Preto by bolo vhodné použiť aspoň dve kamery a spracúvať obraz z oboch. Tým by sa zvýšila úspešnosť rozpoznávania stlačených klávesov a taktiež by sa tým ponechala vysoká úspešnosť ich predošlej segmentácie.

Aplikáciu tohto typu by bolo možné vyvíjať aj pre smartfóny, keďže s klasickými osobnými počítačmi sa pri detekcii klávesov komplikovane mechanicky manipuluje a takýto softvér by sa v ich prípade dal v reálnom čase využívať len veľmi zložito. Pri aplikácii pre smartfóny by bol tento problém vyriešený, keďže smartfóny vieme veľmi jednoducho postaviť na statív.

Záver

Cieľom tejto bakalárskej práce bolo implementovať prototyp aplikácie, ktorá by dokázala vo videu rozpoznať klávesy a následne vedieť detegovať ich stláčanie. Dáta o stlačení týchto kláves mala aplikácia následne vedieť exportovať vo formáte *MIDI* alebo *MusicXML*. Tento cieľ sa nám aj podarilo úspešne naplniť, pričom implementácia bola urobená v programovacom jazyku Python. Vymysleli sme aj vlastný postup riešenia daného problému, ktorý doteraz ešte nikde nebol použitý.

V prvej časti práce sme vysvetlili základné pojmy, s ktorými sme pracovali. Boli to pojmy z oblasti teórie hudby a spracovávania obrazu. Taktiež sme tu vytvorili prehľad, kto sa tejto problematike doteraz venoval a aké boli výsledky.

Ďalšie dve časti tejto práce hovorili o návrhu nášho riešenia a o jeho implementácii. Spomenuli sme aj všetky technológie, ktoré sme v priebehu implementácie využívali. Sú tu rozobraté aj problémy, na ktoré sme v priebehu implementácie narazili spolu s ich úplnými alebo aspoň čiastočnými riešeniami.

V poslednej časti tejto práce sme sa venovali testovaniu našej implementácie. Najskôr sme uskutočnili merania, koľko trvali výpočty jednotlivých častí programu pri rôznych rozlíšeniach a z nameraných dát sme spravili tabuľku. Ďalej sme hovorili o úspešnosti segmentácie klávesov a detekcii ich stláčania. Pri analyzovaní tejto úspešnosti sme sa venovali aj slabším stránkam našej implementácie a taktiež sme popísali, čím boli spôsobené.

V poslednej kapitole sme sa venovali potenciálu ďalšieho rozvoja výskumu tejto problematiky a popísali sme rôzne spôsoby, ktorými by sa dala naša implementácia vylepšiť. Jedná sa hlavne o zlepšenie efektivity výpočtov. Ako jedno z najdôležitejších možných potenciálnych vylepšení našej aplikácie sme vyhodnotili vytvorenie aplikácie pre telefóny, ktorá by vedela vykonávať automatickú hudobnú transkripciu v reálnom čase.

Zoznam použitej literatúry

[1] Mohammad Akbari. "Real-Time Piano Music Transcription Based on Computer Vision". In: *IEEE Transactions on Multimedia* 17.12 (2015), s. 2113–2121. ISSN: 15209210. DOI: 10.1109/TMM.2015.2473702.

[2] A. Droppová. *Základy hudobnej teórie: učebný text pre odbor učiteľstvo pre 1. st. ZŠ*. Vysokoškolské učebné texty / Pedagogická fakulta Prešovskej univerzity. Prešovská univerzita, 1998. ISBN: 9788088697398.

[3] D. Forró. *Svět MIDI*. Grada Publishing, 1997. ISBN: 9788071694120.

[4] Anssi P. Klapuri. "Automatic Music Transcription as We Know it Today". In: *Journal of New Music Research* 33.3 (2004), s. 269–282. ISSN: 17445027. DOI: 10.1080/0929821042000317840.

[5] S. Kolkur et al. "Human Skin Detection Using RGB, HSV and YCbCr Color Models". In: aug. 2017. DOI: 10.2991/iccasp-16.2017.51.

[6] Amit Kumar a Shivani Malhotra. "Real-time Human Skin Color Detection Algorithm using Skin Color Map". In: *2015 International Conference on Computing for Sustainable Global Development*. Mar. 2015. ISBN: 9789380544168.

[7] Robert Mccaffrey. "Piano Music Transcription Based on Computer Vision". Diz. pr. University of Dublin, Trinity College, 2017. URL: https://www.scss.tcd.ie/publications/theses/diss/2017/TCD-SCSS-DISSERTATION-2017-087.pdf (cit. 10. 02. 2020).

[8] *MIDIUtil documentation*. URL: https://midiutil.readthedocs.io/en/1.2.1/ (cit. 28. 04. 2020).

[9] *music21 Documentation*. URL: https://web.mit.edu/music21/doc/index.html (cit. 28. 04. 2020).

[10] Dujana A Nuzra K a Minnu C Jayan. "Real-time human skin color detection using OTSU thresholding". In: *International Journal of Advance Research* 4.3 (2018), s. 400–407.

[11] *OpenCV 2.4.13.7 documentation*. URL: https://docs.opencv.org/2.4.13.7/ (cit. 10. 02. 2020).

[12] *Python 3.8.1 documentation.* URL: `https://docs.python.org/3/` (cit. 10. 02. 2020).

[13] S. Rosen a P. Howell. *Signals and Systems for Speech and Hearing.* Research in the Sociology of Organizations. Emerald, 2011. ISBN: 9781848552265.

[14] Ihab S. Mohamed. "Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques". Diz. pr. Sept. 2017. DOI: `10.13140/RG.2.2.30795.69926`.

[15] Mehmet Sezgin a Bülent Sankur. "Survey over image thresholding techniques and quantitative performance evaluation". In: *Journal of Electronic Imaging* 13.1 (2004), s. 146–165. DOI: `10.1117/1.1631315`. URL: `https://doi.org/10.1117/1.1631315`.

[16] Elena Sikudova et al. *Počítačové videnie: Detekcia a rozpoznavanie objektov. - 1. vydanie.* Wikina, 2014. ISBN: 978-80-87925-06-5.

[17] Milan Sonka, Vaclav Hlavac a Roger Boyle. *Image processing, analysis and and machine vision (3. ed.).* Jan. 2008. ISBN: 978-0-495-24438-7.

[18] Potcharapol Suteparuk. "Detection of piano keys pressed in video". In: *Dept. of Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep* (2014).

[19] *Tkinter documentation.* URL: `https://docs.python.org/3/library/tkinter.html` (cit. 13. 03. 2020).