UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matevž Lipanje

Klavir za pešce - ustvarjanje glasbe z računalniškim vidom

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Franc Solina
Somentor: dr. Borut Batagelj

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01614/2009
Datum: 15.10.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEVŽ LIPANJE**

Naslov: **KLAVIR ZA PEŠCE - USTVARJANJE GLASBE Z RAČUNALNIŠKIM VIDOM**

PIANO CROSSING - GENERATING MUSIC USING COMPUTER VISION

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Kandidat naj v svoji diplomski nalogi izdela program, ki bo omogočal v realnem času detekcijo pešcev pri prečkanju cestišča na območju zebre. V ta namen naj preuči metode računalniškega vida za razpoznavo zebre ter sledenje objektov v realnem času. Na podlagi razpoznanih objektov naj izdela umetniško instalacijo, ki bo omogočala igranje peščev po navidezni klaviaturi. Kot ovrednotenje uporabljenih metod računalniškega vida naj kandidat svojo aplikacijo tudi preizkusi v resničnem okolju.

Mentor:

prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Matevž Lipanje**,

z vpisno številko **63010086**,

sem avtor diplomskega dela z naslovom:

Klavir za pešce - ustvarjanje glasbe z računalniškim vidom

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  prof. dr. Franca Soline

  in somentorstvom

  dr. Boruta Batagelja

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 12.01.2010 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju prof. dr. Francu Solini in somentorju dr. Borutu Batagelju za strokovno vodstvo in pomoč.

Posebna zahvala za sodelovanje in vsestransko podporo gre staršem, Neži in Poloni (tudi Bavcu).

Kazalo

Seznam uporabljenih kratic in simbolov

2-D - Dvodimenzionalen

API - Programski vmesnik (ang. Application Program Interface)

MIDI - Standardni protokol za komunikacijo elektronskih glasbenih naprav (ang. Musical Instrument Digital Interface)

RGB - Barvni model, pri katerem se za opis barve uporabi kombinacija rdeče, zelene in modre barve (ang. Red, Green, Blue)

USB - Univerzalno serijsko vodilo za priklop različnih perifernih naprav na računalnik (ang. Universal Serial Bus)

VGA - Grafični standard ločljivosti 640x480 slikovnih elementov (ang. Video Graphics Array)

YUV - Barvni model, pri katerem se komponente barvnega modela RGB ločijo na svetlost (Y) in barvo (UV)

Povzetek

Klavir za pešce je interaktivna umetniška instalacija, ki prehod za pešce spreminja v delujočo klaviaturo, kjer pešci s svojimi koraki ustvarjajo glasbo. Instalacijo sestavlja zebratura - v klaviaturo predelan prehod za pešce; tako da so k belim črtam prehoda dodane črne črte - tipke, digitalna kamera usmerjena v zebraturo in računalnik z zvočniki. Na računalniku teče posebej izdelana programska oprema, ki z uporabo metod računalniškega vida programske knjižnice OpenCV zaznava stike posameznih pešcev in tipk na klaviaturi ter v skladu s tem generira notno zaporedje MIDI, ki se nato sproti pretvarja v glasbo.

Delovanje sestavljata dva ključna procesa. Generiranje klaviature predstavlja inicializacijo, pri kateri se iz posnetka ustvari abstraktna predstavitev klaviature MIDI. Posnetek se segmentira na posamezna območja, ki pripadajo tipkam. Tipka vsebuje predstavitev območja - konture, ki ga zaseda na sliki, notno številko MIDI in status tipke. Abstraktna predstavitev klaviature se nato uporabi v procesu Sledi in igraj, ki pomeni dejansko izvajanje. Za določanje položaja pešcev se lahko uporabita dve tehniki za ločevanje ozadja - enostavna tehnika ali napredna tehnika s kodirno knjigo.

Programska oprema, ki podpira delovanje instalacije, je bila preizkušena na dveh modelih Klavirja za pešce - v naravni velikosti na prostem in na pomanjšanem v zaprtem prostoru.

Ključne besede:

računalniški vid, umetniška instalacija, glasba, klavir

Abstract

Piano Crossing is an interactive art installation which transforms the zebra crossing into a working piano keyboard where pedestrians generate music by walking across it. Installation consists of the zeboard - zebra crossing turned keyboard - by adding black stripes to the existing white ones, serving as keys, a digital camera overlooking the zeboard and of a laptop computer with speakers. The computer runs a specially designed software, which recognizes pedestrian's contacts with the keys by using OpenCV's computer vision methods, and by that generates a MIDI sequence, which furthermore turns into music.

The application consists of two main processes. Generate Keyboard is about creating abstract presentation of MIDI keyboard from an input frame. The image is segmentated to individual regions belonging to the keys. Each key includes a contour of the region, a MIDI note number and a key status. The abstract keyboard presentation is then used in the Track and Play process, which brings the actual execution. For the pedestrian positioning two different background substraction techniques can be used; the simple technique or the advanced codebook model technique.

The installation software was tested on two different models of the Piano Crossing; on an actual zebra crossing outdoor and on the reduced-in-size model indoor.

Key words:

computer vision, art installation, music, piano

Poglavje 1

Uvod

Klavir za pešce je interaktivna umetniška instalacija, ki prehod za pešce spreminja v delujočo klaviaturo, kjer pešci s svojimi koraki ustvarjajo glasbo. Goli namembnosti varnega prehoda čez cesto je dodana nova funkcionalnost - generiranje notnega zapisa, ki ga računalnik sproti pretvarja v glasbo. K belim črtam prehoda so dodane črne črte - tipke, ki skupaj tvorijo virtualno klaviaturo. Pešec s koraki aktivira posamezne tipke, podobno kot to počne pianist s prsti pri klavirju.

Instalacijo sestavlja nekoliko predelan prehod za pešce - dodane so črne tipke, digitalna kamera, usmerjena v klaviaturo in računalnik z zvočniki. Na računalniku teče posebej izdelana programska oprema, ki s segmentacijo posnetkov zaznava stike posameznih pešcev in tipk na klaviaturi ter v skladu s tem generira notni zapis, ki se pretvarja v glasbo.

V diplomski nalogi je predstavljeno tehnično ozadje instalacije, še posebej način uporabe metod računalniškega vida s poudarkom na metodah segmentacije posnetkov za prepoznavo klaviature in ločevanje pešcev od ozadja.

1.1 Računalniški vid in umetnost

Računalniški vid (Slika 1.1) je bil zaradi visokih zahtev po procesorski in spominski kapaciteti nekdaj rezerviran le za področja, ki so zmogla visoke vložke (vojaške, medicinske, industrijske aplikacije). Danes pa je možno zaradi višjih in cenejših procesorskih kapacitet ter cenenih vizualnih senzorjev metode računalniškega vida implementirati že na namiznih računalnikih. Zato je spekter problemov, ki se ga danes lotevamo z metodami računalniškega vida, veliko širši in bogatejši. Zaradi večje zmogljivosti računalnikov se razvijajo metode za interpretacijo video sekvenc in metod, ki delajo v realnem času [1].

Eno izmed področij, na katerega se je v zadnjem času razširila uporaba metod računalniškega vida, je tudi področje sodobne umetnosti oziroma umetnosti novih medijev. Interaktivne instalacije, kjer obiskovalci s svojo prisotnostjo in gibanjem aktivno sodelujejo pri nastajanju umetniškega dela, brez sistemov računalniškega vida za analizo slik pravzaprav sploh ne bi bile izvedljive [2]. Hitro razvijajoče področje računalniške znanosti omogoča aplikacijo nevsiljivih uporabniških vmesnikov, ki postavljajo obiskovalca v samo središče umetniškega dela in s tem obogatijo njegovo izkustvo.

Najpogosteje uporabljene metode računalniška vida v umetnosti so tiste, ki omogočajo neke vrste interakcijo med uporabnikom in sistemom. To so zaznava gibanja, iskanje obrazov in sledenje [3]. Veliko napora se vlaga v izboljšanje procesov razpoznave predmetov, saj bi to omogočalo nove načine interakcije in uporabo predmetov za krmiljenje [4]. Procesorska moč sodobnih namiznih računalnikov je že tako visoka, da se na njih lahko izvajajo aplikacije računalniškega vida, kar v nedavni preteklosti še ni bilo mogoče, zaradi česar se to področje pospešeno razvija.

Poleg interaktivnosti, ki uporabnike intergrira v samo delo, je za umetniške instalacije značilna tudi multimedijalnost. To pomeni, da umetniško delo vključuje več medijev naenkrat (usklajevanje gibanja, podobe zvoka ...) in s tem stimulira več čutov hkrati.

Slika 1.1: Računalniški vid

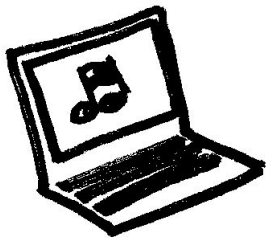
1.2 Glasba računalnikov

Napredek v računalniški tehnologiji je dramatično vplival tudi na ustvarjanje in izvajanje glasbe. Trenutna generacija mikro-procesorjev je sposobna izvajati sofisticirane zvočne sinteze z uporabo pestre palete algoritmov in pristopov. Računalniška tehnologija je danes v glasbenem ustvarjanju vse povsod prisotna; programski sintetizatorji zvoka in digitalni mešalci so tako običajni, da je izbira digitalne tehnologije pred analogno za ustvarjanje in snemanje glasbe

prej pravilo kot izjema. Glasbi, ustvarjeni z računalniško tehnologijo, pravimo računalniška glasba [6].

Poleg večinske uporabe računalnika kot orodja za glasbeno produkcijo, kjer glasbo še vedno piše človek-skladatelj s pomočjo računalnika, pa obstajajo tudi primeri, ko računalnik prevzame vlogo skladatelja - tako nastali glasbi bi lahko rekli *glasba računalnikov* (Slika 1.2). Kot se izkaže, so taki sistemi za komponiranje glasbe obstajali že veliko pred računalniki. Poznan je primer *Musikalisches Würfelspiel*, kjer se je uporabila igralna kocka za naključno izbiro kratkega dela že prej napisane glasbe, ki je potem sestavljena skupaj tvorila novo nastalo kompozicijo. Za tako nastalo glasbo seveda ne moremo reči, da so jo ustvarili računalniki v pravem pomenu besede, vendar je uporaba naključnosti kot gonila ustvarjalnosti značilna za glasbo računalnikov. S pojavom zmogljivejših računalnikov so nastale prve algoritmične kompozicije. Za izvor glasbe se tako uporabijo fraktali, statistični modeli ali celo struktura molekule DNA. Nekoliko drugačen pristop so ubrali tisti, ki ustvarjajo glasbo s pomočjo senzorjev za zaznavo premikanja. Zanimiv je tudi način, ko glasba nastaja z analizo barv video posnetkov oziroma posameznih slik.

Čeprav je ustvarjanje glasbe te vrste brez uporabe računalnikov praktično nemogoče, je to pravzaprav glasba določene konkretne stvari, ki to glasbo generira, bodisi matematični model, naravni pojav ... V skladu s tem bi lahko glasbo, ki nastaja pri instalaciji Klavir za pešce, imenovali glasba pešcev pri prehodu čez cesto.



Slika 1.2: Glasba računalnikov

Poglavje 2

Tehnično ozadje izvedbe

Želimo realizirati idejo, da bi prehod za pešce s pomočjo računalniškega vida spremenili v delujočo klaviaturo.

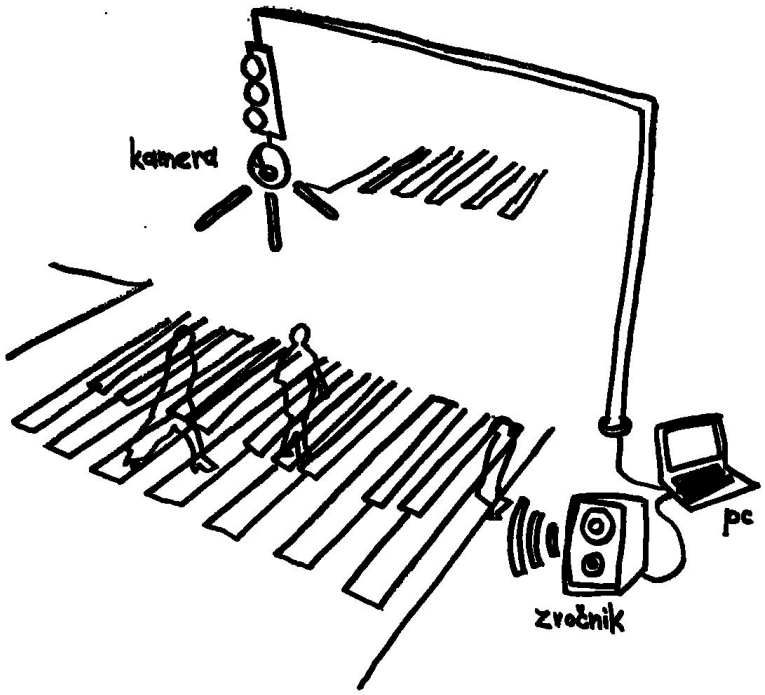
Ker bi radi, da ostane prehod za pešce povsem funkcionalen in le vizualno nekoliko podobnejši klaviaturi poleg obstoječih belih črt - tipk naredimo še črne črte - tipke. Samolepilna folija, ki se navadno uporablja za ravne lesene površine, se izkaže za povsem primerno tudi na asvaltu, saj se zlahka ne odlepi. Pravokotno nad prehodom za pešce je na primerni višini nameščena digitalna kamera, in sicer tako, da zajema celotno klaviaturo. Ena izmed možnosti je, da spletno kamero z ločljivostjo VGA (640 x 480 slikovnih elementov) pritrdimo na semafor, kot to prikazuje slika 2.1. Spletna kamera je preko vmesnika USB povezana s prenosnim računalnikom, na katerem je nameščena posebej izdelana programska oprema.

2.1 Zebratura

Klaviatura je skupina črnih in belih tipk, razvrščenih v točno določenem zaporedju. Pojavlja se pri glasbenih instrumentih, pri katerih so tipke sestavni del mehanizma za proizvajanje zvoka (instrumenti s tipkami, npr. klavir). Pritisk na posamezno tipko povzroči, da glasbilo zaigra točno določen ton. Osnovni vzorec klaviature obsega 12 tonov (7 diatoničnih in 5 kromatičnih) v okviru ene oktave.

Zebra - prehod za pešce je površina, namenjena varnemu prehodu pešcev čez cesto. Sestavljajo jo vzporedni beli pravokotniki z daljšo stranico v smeri ceste.

Ideja je združiti podobo zebre s podobo klaviature v t.i. *zebraturo* (Slika 2.2). To dobimo, če zebri dodamo črne tipke oziroma klaviaturi odstranimo

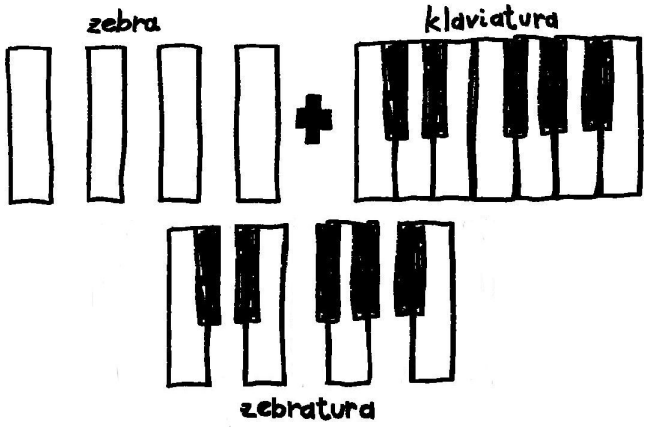


Slika 2.1: Shema instalacije. Spletna kamera, usmerjena v zebraturo in prenosni računalnik z zvočniki.

vsako drugo belo tipko. Dobljena podoba zebrature ima lastnosti zebre, saj želimo, da ohrani funkcionalnost prehoda za pešce, hkrati pa postane vizualno podobna okrnjeni klaviaturi.

2.2 OpenCV

Pri izbiri programske knjižnice za delo z računalniškim vidom nismo imeli velikih problemov. Programski paket OpenCV vsebuje veliko število metod računalniškega vida, omogoča hitro procesiranje posnetkov v realnem času, in je zaradi neodvisnosti od operacijskega sistema lahko prenosljiva na druge platforme. Knjižnica je namenjena široki uporabi, saj je izdana pod licenco BSD (prvotno je bila razvita pod okriljem podjetja Intel), kar omogoča prosto uporabo tako za raziskovalne kot tudi komercialne aplikacije. Napisana je v programskih jezikih C in C++, kar dopušča manipulacijo na najnižjem nivoju in omogoča hitro izvajanje.



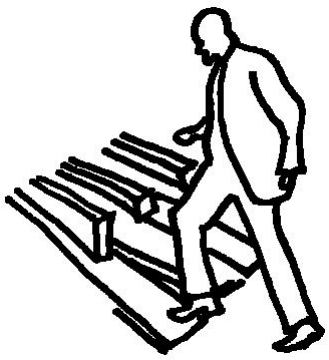
Slika 2.2: Zebratura. Podoba zebre združena s podobo klaviature.

Delovanje aplikacije je avtomatizirano in razdeljeno na dva poglavitna procesa. Prvi proces - Generiranje klaviature je neke vrste inicializacija, pri kateri se iz posnetka ustvari abstraktna predstavitev klaviature. Posnetek se segmentira na homogena območja. Kontura območja se hrani v pomnilniku kot posamezna tipka, ki skupaj s konturami ostalih območij tvori klaviaturo. Razpoznava je razdeljena na generiranje najprej belih in nato črnih tipk. Vsaki tipki se v skladu s položajem na klaviaturi dodeli ustrezna notna številka MIDI. Abstraktna predstavitev klaviature se nato uporabi v drugem procesu s pomenljivim nazivom - Sledi in igraj, ki pomeni dejansko izvajanje. Na vsakem posnetku se z uporabo tehnike za ločevanje ozadja zazna pozicija posameznega pešca. V kolikor se pozicija pešca ujema z območjem določene tipke, se generira sporočilo MIDI za aktivacijo ustrezne note (ang. Note-On message). Programski sintetizator zvoka zaporedje MIDI pretvarja v zvočni signal, ki se preko zvočnika širi v prostor.

2.3 MIDI

MIDI (ang. Musical Instrument Digital Interface) je elektronski standardni protokol, s katerim med seboj komunicirajo različne elektronske glasbene naprave [7]. Namesto zvočnega signala se pri protokolu MIDI pošilja zaporedje dogodkovnih sporočil, ki natančno opredeljujejo glasbo, kot npr. nota C#, ki naj se aktivira (ang. Note-On) (Slika 2.3), glasnost (ang. volume), kanal (ang. channel) ... V bistvu gre za neke vrste razširjeno elektronsko notno transkrip-

cijo. Če želimo zaporedje MIDI pretvoriti v zvočni signal, potrebujemo zvočno enoto, bodisi strojni ali programski sintetizator zvoka, in seveda zvočnik.



Slika 2.3: Aktivacija note (Note-On)

Tehnično gledano je naša instalacija (ozirom programska oprema, ki jo podpira) MIDI naprava (ang. MIDI controller), saj oddaja zaporedje sporočil MIDI. S tem ko smo reprodukcijo zvoka ločili od same implementacije, je mogoča manipulacija zvočne podobe kompozicije, ki nastaja pri prehodu pešcev čez cesto. Standardni sintetizator MIDI ponuja 128 različnih glasbil (ang. patches). Pešci lahko tako zvenijo v celi paleti glasbenih odtenkov (od klavirja do bobnov).

Programiranje s protokolom MIDI na najnižjem nivoju omogoča programski vmesnik MIDI API. Poleg tega obstaja veliko knjižnic na višjih nivojih, ki programiranje MIDI precej poenostavljajo. Odločili smo se za majhen odprtokodni projekt - "Wrapper Library for Windows MIDI API"[8], ki našim potrebam - pošiljanje osnovnih sporočil MIDI povsem ustreza.

Poglavje 3

Generiranje klaviature

Bele in črne črte bi radi spremenili v virtualno klaviaturo. Če želimo to doseči, je potrebno iz slike na nek način razčleniti - segmentirati območja, ki pripadajo posameznim tipkam, in v pomnilniku shraniti predstavitev teh območij, ki skupaj tvorijo klaviaturo. Velikost klaviature je poljubna, poznamo le njeno strukturo. Kljub temu želimo proces karseda avtomatizirati. Glede na to, da je kamera fiksna, zadostuje, če generiramo klaviaturo na začetku, saj lahko predpostavimo, da se pozicija tipk med delovanjem aplikacije ne bo spreminjala.

Abstraktna predstavitev klaviature MIDI se uporabi kot vhodni podatek naslednjega procesa - Sledi in igraj. Tipka vsebuje predstavitev območja - konture, ki ga zaseda na sliki, notno številko MIDI in status tipke. Notna številka MIDI se dodeli vsaki tipki, skladno z njenim položajem na klaviaturi. Status tipke hrani informacijo o tem, ali je določena tipka aktivirana ali ne.

Kako iz posnetka segmentirati konture posameznih tipk in generirati abstraktno predstavitev klaviature MIDI, si bomo podrobneje ogledali v pričujočem poglavju.

3.1 Postopek

Postopek generiranja klaviature sestoji iz naslednjih korakov:

1. Predpriprava slike
    - Odpravljanje šuma
    - Sivinska slika
2. Piramidna segmentacija tipk

3. Iskanje kontur belih in črnih tipk
    - Upragovanje
    - Analiza kontur
4. Generiranje klaviature MIDI

3.2 Predpriprava slike

Najprej želimo iz posnetka, ki ga zajame kamera, odstraniti šum, saj nas moti pri nadaljni obdelavi.

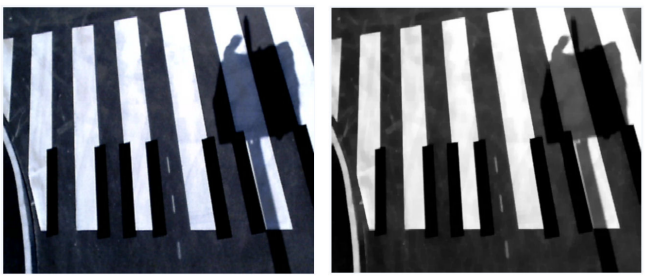
Najpogostejši je Gaussov šum, kar pomeni, da je pravim vrednostim slikovnih elementov dodana vrednost šuma, ki ima naključno spremenljivko porazdeljeno po Gaussovi porazdelitvi. To odpravimo, če izračunamo konvolucijo posnetka z Gaussovo funkcijo. Vsak slikovni element iz posnetka konvoliramo z 2-D matriko, ki predstavlja vzorce realnega Gaussovega jedra z določeno deviacijo σ. Uporabimo matriko velikosti 3 x 3.

Poleg odpravljanja šuma je pred nadaljnjo obdelavo sliko potrebno pretvoriti iz barvnega prostora RGB v sivinsko sliko.

Slikovni elementi sivinske slike so sestavljeni iz ene same vrednosti, za razliko od barvnega prostora RGB, kjer vsak slikovni element sestavlja kombinacija treh vrednosti oziroma barv (rdeča-R, zelena-G in modra-B). Sivinski nivo sivega slikovnega elementa S izračunamo iz uteženega povprečja vseh treh barvnih kanalov (R,G,B) po enačbi:

$$S = 0.299 * R + 0.587 * G + 0.114 * B \tag{3.1}$$

Rezultat postopka predpriprave slike prikazuje slika 3.1.



Slika 3.1: Predpriprava slike. Iz originalne slike (levo) dobimo sivinsko sliko z odpravljenim šumom (desno).

3.3 Piramidna segmentacija

Sivinsko sliko želimo razdeliti na homogena območja - segmente, ki bodo sestavljena iz slikovnih elementov enakih vrednosti. Vsako območje predstavlja zaključeno celoto, v našem primeru želimo doseči, da bo območje pripadalo posamezni tipki. V najboljšem primeru bo po končani segmentaciji posnetek sestavljen iz treh barv - črne in bele (tipke) ter sive (ozadje).

Ena izmed tehnik segmentacije se izvaja s piramidami - *piramidna segmentacija*. Slikovna piramida je zbirka slik, nastalih iz ene slike z zmanjševanjem natančnosti vzorčenja (ang. down-sampling). Število slikovnih elementov zmanjšujemo do željene ločljivosti. Seveda se proces ustavi, ko pridemo do enega slikovnega elementa. Za dosego nivoja $(i+1)$ začetno sliko G_i konvoliramo z Gaussovim jedrom kot pri Gaussovem glajenju, nato iz dobljene slike odstranimo slikovne elemente sodih stolpcov in vrstic. Če ta postopek ponavljamo, dobimo iz začetne slike G_0 celotno Gaussovo piramido. (Obratni postopek delamo z Laplacovo piramido.)

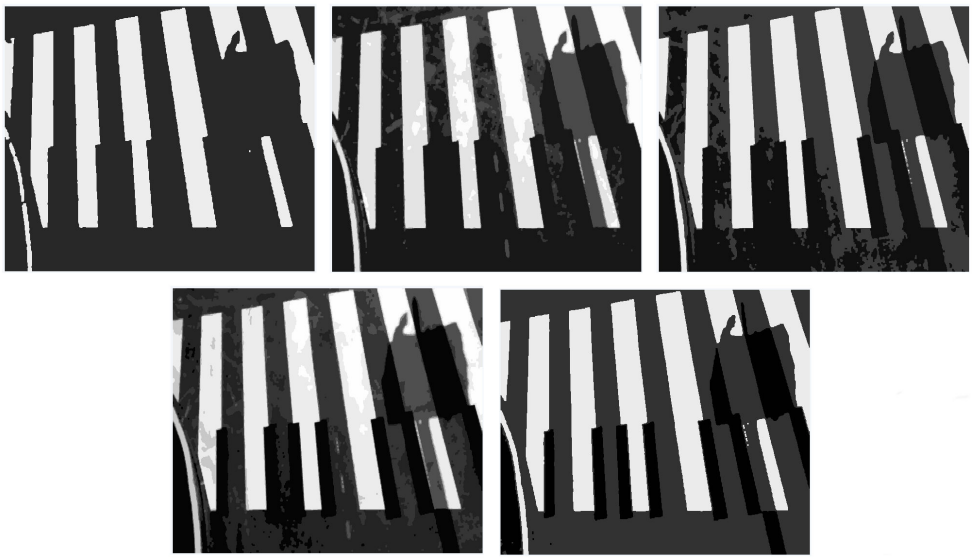
Piramidna segmentacija je iterativen algoritem, pri katerem se med slikovnimi elementi na nivoju G_i in tistimi na višjem nivoju G_{i+1} vzpostavijo relacije dedovanja (oče-sin). Segmentacija se začne na najvišjem nivoju piramide s sliko najmanjše ločljivosti in se postopoma razširi na nižje nivoje, do začetne slike. Algoritem vsebuje naslednje korake [9]:

- Izračun Gaussove piramide
- Povezovanje slikovnih elementov
- Združevanje povezanih slikovnih elementov v skupine

Korak 2 in 3 se ponavljata, dokler ni dosežen željen nivo segmentacije. Sin ima na nivoju i štiri potencialne očete na višjem nivoju $i+1$. Povezava med slikovnim elementom a na nivoju i in slikovnim elementom b na nivoju $i+1$ (relacija oče-sin) se vzpostavi, če je $p(c(a), c(b)) < prag1$. Nastale povezave med slikovnimi elementi tvorijo povezana območja, ki se združujejo v skupine. Območje A in B pripadata isti skupini, če velja $p(c(A), c(B)) < prag2$, kjer pri sivinskih slikah c predstavlja vrednost slikovnega elementa in $p(c_1, c_2) = |c_1 - c_2|$. Vhodna parametra algoritma sta vrednosti $prag1$, $prag2$ in željen nivo segmentacije i.

Optimalne vrednosti vhodnih parametrov poiščemo s poizkušanjem. Vsakemu parametru dodelimo drsnik, ki omogoča spremembo vrednosti posameznega vhodnega parametra in takojšen ogled rezultata segmentacije. Z določanjem različnih pozicij drsnikov hitro preizkusimo vse možnosti in ugotovimo, da za

dosego najboljših rezultatov zadostuje segmentacija na prvem nivoju. Pregled rezultatov segmentacij pri različnih vrednostih vhodnih parametrov prikazuje slika 3.2.



Slika 3.2: Piramidna segmentacija. Preizkušanje različnih vrednosti vhodnih parametrov: od nepravilne (levo zgoraj) do optimalne segmentacije (desno spodaj).

3.4 Iskanje kontur

S piramidno segmentacijo smo posnetek uspešno razčlenili na homogena območja, ki pripadajo posameznim tipkam oziroma ozadju. Kljub temu ne vemo nič o tem, kje se določeno območje nahaja ali kateri slikovni elementi pripadajo območju. Potrebujemo abstraktno predstavitev območja. Območje je natančno določeno z urejenim zaporedjem mejnih slikovnih elementov - kontur (ang. contour).

Algoritem za iskanje kontur potrebuje za vhodni podatek binarno sliko. Binarna slika vsebuje samo slikovne elemente z vrednostjo 0 in slikovne elemente z vrednostjo 1. Množica povezanih slikovnih elementov z vrednostjo 0 ali slikovnih elementov z vrednostjo 1 tvori povezano področje (ang. connected component). Pri povezljivosti gre za dotik slikovnega elementa s svojimi sosedi. V 2D-prostoru poznamo dve vrsti povezljivosti slikovnega elementa: povezljivost-8 in povezljivost-4 (Slika 3.3).

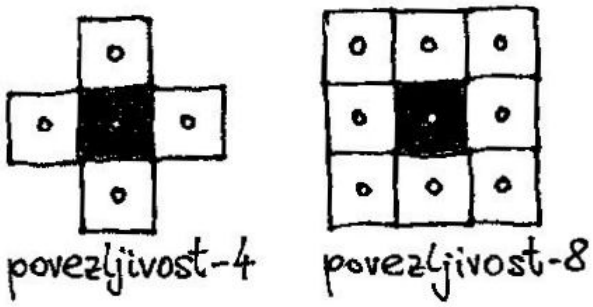
- Povezljivost-4 velja za tista slikovna elementa, ki se dotikata s svojimi stranicami. Slikovna elementa sta lahko povezana vodoravno ali navpično. Slikovna elementa s koordinatami (x', y') in (x'', y'') sta povezana-4 če velja

$$|x' - x''| + |y' - y''| = 1 \tag{3.2}$$

- Povezljivost-8 velja za tista slikovna elementa, ki se dotikata s svojimi stranicami ali robovi. Slikovna elementa sta lahko povezana vodoravno, navpično ali diagonalno. Pri povezljivosti-8 v koordinatah velja

$$max(|x' - x''|, |y' - y''|) = 1 \tag{3.3}$$

S pomočjo dveh relacij povezljivosti je mogoče binarno sliko razdeliti na povezana-8 ali povezana-4 področja. Povezano področje sestavljajo slikovni elementi enake vrednosti (0 ali 1) in vsak par slikovnih elementov iz povezanega področja je povezan z zaporedjem povezanih-8 ali povezanih-4 slikovnih elementov. Z drugimi besedami, med poljubnima slikovnima elementoma iz povezanega področja mora obstajati pot, sestavljena iz povezanih-8 ali povezanih-4 slikovnih elementov.



Slika 3.3: Povezljivost slikovnih elementov

Imamo povezano področje, sestavljeno iz slikovnih elementov z vrednostjo 1 in ozadje, ki ga sestavljajo slikovni elementi z vrednostjo 0. Mejni slikovni element področja je slikovni element, ki pripada področju in je hkrati povezan-4 z ozadjem. Vsi mejni slikovni elementi področja tvorijo konturo - očrt področja. Kontura področje natačno definira.

Knjižnica za delo z računalniškim vidom - OpenCV vključuje implementacijo algoritma za iskanje kontur [10]. Algoritem se sprehodi čez vse slikovne elemente - vrstico za vrstico, v večini primerov zadostuje en obhod. Ko naleti

na slikovni element, ki pripada še ne odkriti konturi, začne s proceduro sledenja mejnim slikovnim elementom, ki odkrije in shrani novo konturo.

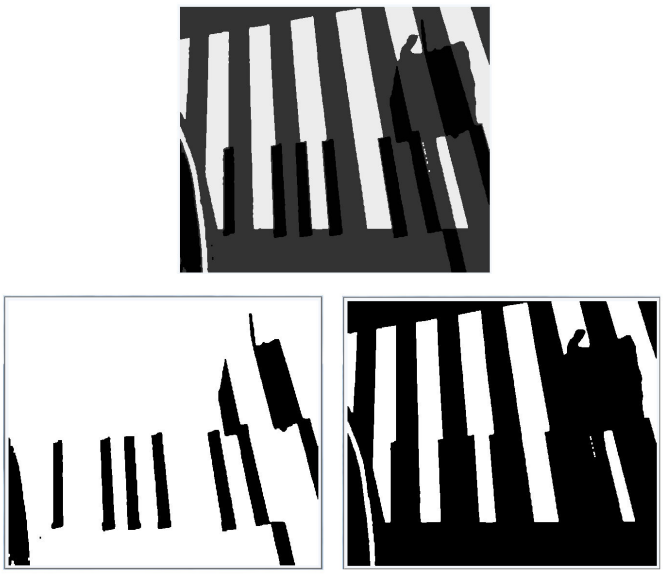
3.4.1 Upragovanje

Sivinsko sliko, ki smo jo dobili po piramidni segmentaciji, je potrebno binarizirati, v kolikor želimo poiskati konture povezanih področij. Slikovni elementi nad določenim pragom naj bodo 1, pod pragom pa 0. To je postopek, ki mu pravimo upragovanje (ang. thresholding):

$$g(i,j) = \begin{cases} 1 & \text{če velja } f(i,j) \geq T \\ 0 & \text{sicer,} \end{cases} \tag{3.4}$$

kjer je $f(i,j)$ vrednost slikovnega elementa (i,j) in T vrednost praga.

Z upragovanjem razdelimo sliko na predmete in ozadje. V primeru prepoznave klaviature imamo črne tipke, bele tipke in ozadje. Proces iskanja kontur celotne klaviature je zato potrebno razdeliti na dva dela. Z upragovanjem pretvorimo segmentirano sliko v dve binarni sliki: črne tipke z ozadjem in bele tipke z ozadjem, kot to prikazuje slika 3.4. (V primeru belih tipk so ozadje pravzavprav bele tipke, kar pa za proces iskanja kontur ni bistveno.)



Slika 3.4: Upragovanje. Segmentirano sliko (zgoraj) binariziramo v črne tipke z ozadjem (levo) in bele tipke z ozadjem (desno).

Ključni problem pri upragovanju je določitev praga. Ker poznamo barvi območij, ki jih želimo izpostaviti, je naloga nekoliko lažja. Poleg tega sta ti barvi najmanjša in največja vrednost barvne lestvice (črna-0, bela-255 za 8-bitne sivinske slike). Težava je kvečjemu pri črnih tipkah, saj je razlika med črno barvo tipk in sivim ozadjem relativno majhna. Funkcija piramidne segmentacije nam poleg segmentiranih območij kot rezultat vrne tudi vrednost slikovnih elementov območja. Povprečna vrednost slikovnih elementov dveh različnih območij se uporabi kot idealen prag za upragovanje.

3.4.2 Analiza kontur

Konture belih oziroma črnih tipk je potrebno analizirati. Želimo ugotoviti, ali najdene konture res pripadajo tipkam in konture v nasprotnem primeru odstraniti. Potrebujemo kriterije za kategorizacijo kontur v tipke.

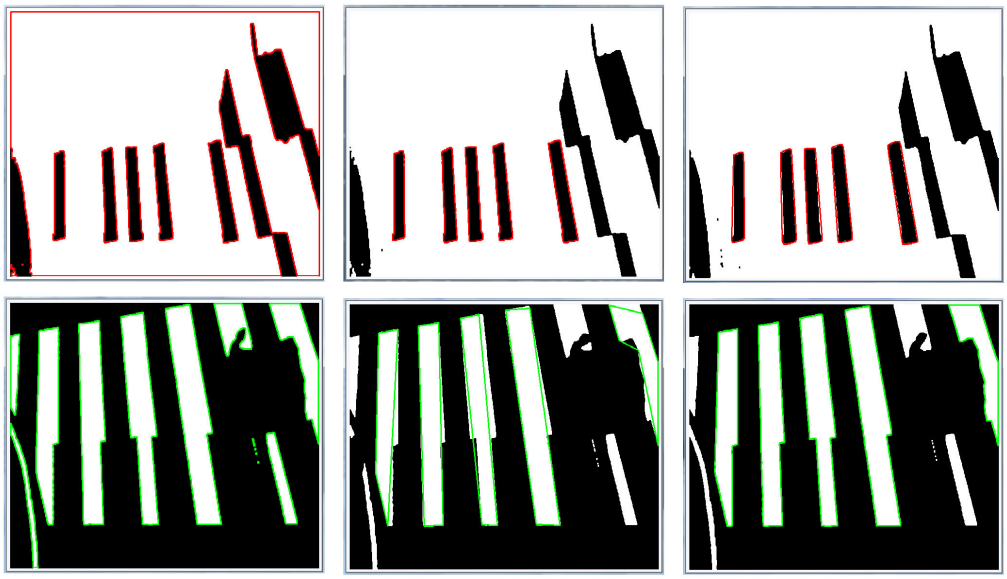
Obstaja več možnosti za predstavitev krivulje konture. Po Freemanovi metodi se kontura shrani kot zaporedje števil, ki predstavljajo sosledje sosedov trenutnega slikovnega elementa. Za vsak slikovni element obstajajo oštevilčeni sosedi od 0 do 7. Predstavitev konture se začne z koordinatama začetnega slikovnega elementa, ki mu sledijo oštevilčeni sosedi, glede na trenutni slikovni element. Pri mnogokotni predstavitvi je krivulja konture sestavljena iz zaporedja točk, kjer je posamezna točka oglišče mnogokotnika. Mnogokotna predstavitev je za analizo kontur veliko bolj primerna od Freemanove metode.

Za nadaljnjo analizo želimo mnogokotnik, ki opisuje tipko, čimbolj poenostaviti. Zaradi predhodne obdelave (segmentacije) je kontura tipke nekoliko popačena. S poenostavitvijo mnogokotnika (ang. polygon approximation) želimo zmanjšati število ogljišč, a hkrati ohraniti dominantno obliko mnogokotnika. Algoritem Douglas-Peucker omogoča aproksimacijo mnogokotnika na željeno natančnost:

- Na danem mnogokotniku se izbereta diametralno čimbolj oddaljeni točki. Daljica, ki povezuje ti dve točki, postane začasna aproksimacija mnogokotnika. Algoritem iterativno dodaja nove točke, ki ležijo na mnogokotniku, k začetni aproksimaciji, dokler ni dosežen željen nivo natančnosti.

- Na danem mnogokotniku se izbere nova točka, ki je čimbolj oddaljena od daljice, ki povezuje začetni točki. Če je razdalja od daljice do nove točke manjša od željene napake, je aproksimacija dosežena. V nasprotnem primeru se nova točka doda k aproksimaciji in postopek se nadaljuje na novo nastalih odsekih.

Poleg aproksimacije mnogokotnika je eden izmed najbolj osnovnih pojmov v računskih geometriji tudi konveksna ovojnica (ang. convex hull). Če je dana množica točk $S \subset \mathbb{R}^n$, je konveksna ovojnica $conv(S)$ definirana kot najmanjša konveksna množica v $\mathbb{R}^n$, ki vsebuje S.



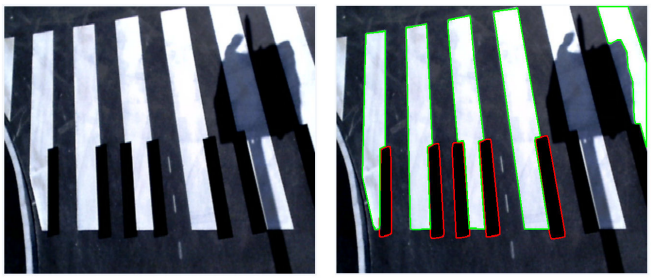
Slika 3.5: Analiza kontur črnih (zgoraj) in belih tipk (spodaj). Najdene konture (levo) filtriramo (sredina) in aproksimiramo v mnogokotnik oz. konveksno ovojnico (desno).

S poenostavitvijo mnogokotnika oziroma s konveksno ovojnico smo dosegli, da je kontura tipke pravilnejše oblike. Tipko naj bi sestavljali vodoravni in navpični odseki, med katerimi so pravi koti, kar pa velja samo v primeru, če je kamera navpično nad klaviaturo. (Črne tipke so v idealnih pogojih pravokotniki.) To v praksi ni mogoče, zato smo za kriterij, po katerem se filtrirajo konture, izbrali dolžino kontur oziroma višino tipke. Na ta način se občutno prekratke konture zavržejo, saj lahko predpostavimo, da ne pripadajo tipkam. Opisan postopek analize kontur prikazuje slika 3.5.

3.5 Generiranje klaviature MIDI

Potrebno je generirati abstraktno predstavitev klaviature (Slika 3.6), ki se uporabi kot vhodni podatek naslednjega procesa - 'Sledi in igraj'. Podatkovno strukturo klaviature (ZKeyboard) sestavljata seznama belih in črnih tipk.

Osrednji atribut podatkovne strukture tipke (ZKey) je predstavitev konture območja, ki ga tipka zaseda na sliki. Podatkovna struktura tipke vsebuje poleg predstavitve konture območja še notno številko MIDI, status tipke in kazalec na naslednjo tipko.



Slika 3.6: Generiranje klaviature. Iz zajete slike (levo) generiramo abstraktno predstavitev klaviature MIDI (desno).

Vsaki tipki se dodeli notna številka MIDI, skladno s položajem tipke na klaviaturi. Pred generiranjem klaviature se kot vhodni podatek določi notno številko prve tipke in številko oktave. Status tipke služi kot indikator, ali je tipka aktivirana al ne, ki se uporablja pri algoritmu za generiranje zaporedja MIDI.

```
typedef struct ZKey {
   CvSeq* keyContour;   //predstavitev konture tipke
   int note;            //notna številka MIDI
   bool noteOn;         //status tipke
   ZKey* next;          //kazalec na naslednjo tipko
};
```

Podatkovno strukturo klaviature sestavljata kazalca na seznama tipk.

```
typedef struct ZKeyboard {
   ZKey* whiteKeys;    //seznam belih tipk
   ZKey* blackKeys;    //seznam črnih tipk
};
```

3.6 Diskusija

Predstavljen je bil postopek segmentacije in razpoznave tipk iz slike ter generiranje klaviature MIDI. Upragovanje slike na osnovi vrednosti slikovnih elementov segmentiranih območij se izkaže za povsem učinkovito metodo, v kolikor je

celotna klaviatura enakomerno osvetljena. V primeru popačenja zaradi močnih senc, ko tipka ni več enotne barve, se popači tudi oblika tipke in s tem njena kontura.

Pomankljivost smo poiskušali odpraviti z metodo, kot je npr. Houghov transform za iskanje ravnih linij na sliki, vendar se pojavi težava, kako iz množice črt sestaviti konture tipke. Podoben problem se pojavi pri uporabi metode za odkrivanje robov - Cannyjev detektor robov. Robovi tipk so navkljub sencam jasno izraženi, vendar se pojavijo še robovi senc. Poleg tega robovi v večini primerov niso zvezni, ampak so sestavljeni iz množice prekinjenih črt. Kriterij za kategorizacijo kontur v tipke smo poiskušali izboljšati tako, da se izločijo konture, ki nimajo pravih kotov med stranicami (oziroma približek pravega kota). V primeru črnih tipk iščemo celo pravokotnike. Kljub temu analiza kontur tipk na podlagi pravih kotov ni obrodila sadov, saj so popačene tipke ostale prezrte.

Poglavje 4

Sledi in igraj

Za delovanje aplikacije je potrebno na nek način ločiti posamezne pešce od ozadja na sliki. Gre za ločevanje predmetov, ki so v ospredju - premikajoči se predmeti, od predmetov, ki so v ozadju - statični predmeti, čemur pravimo ločevanje ozadja (ang. background substraction). Če želimo to uspešno izvesti, moramo najprej zgraditi model ozadja, ki ga potem uporabimo za ločevanje predmetov v ospredju. Model ozadja med sledenjem posodabljamo glede na trenutno sliko. Na ta način se model prilagaja spremembam in omogoča zaznavo predmetov v ospredju. Aplikacija omogoča uporabo dveh prilagodljivih tehnik za ločevanje ozadja. Podrobnosti si oglejmo v nadaljevanju.

4.1 Enostavna tehnika ločevanja ozadja

Pri tej tehniki se za model ozadja vzame kar prvi posnetek iz videa, ki mu pravimo tudi referenčni posnetek. Pomebno je, da je referenčni posnetek brez predmetov, ki jih želimo kasneje opredeliti kot predmeti v ospredju. Razlika med trenutnim posnetkom in referenčnim posnetkom, ki predstavlja ozadje, po formuli:

$$\|I_t - B\| > T, \tag{4.1}$$

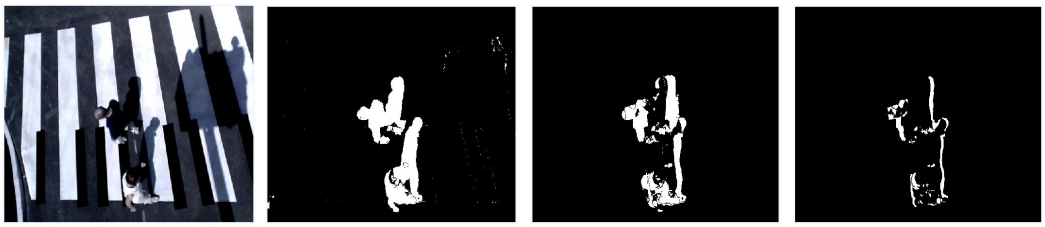
kjer I_t predstavlja posnetek videa ob času t, B je posnetek ozadja, se označi kot predmet v ospredju, če je večja kot prag razlike T. Vsak posnetek je primerjan s prednastavljenim posnetkom ozadja. Če je razlika večja od vrednosti praga, ki predtavlja šum, se razlika označi za predmet v ospredju. Ta tehnika daje zadovoljive rezultate v okolju, kjer se osvetlitev ne spreminja veliko (zaprti prostori). Če želimo, da bo naša aplikacija pravilno delovala tudi zunaj, potrebujemo mehanizem, s katerim se bo referenčni posnetek ozadja prilagajal

spremembam (npr. osvetlitve). V naši enostavni tehniki se vsi slikovni elementi ozadja spreminjajo, kot v sistemu PFINDER[11], po formuli prilagajanja:

$$B_t = (1 - \alpha)B_{t-1} + \alpha I_t \tag{4.2}$$

Na ta način dobimo model ozadja, ki je sposoben prilagajanja spremembam v okolju. Hitrost prilagajanja določa vrednost $\alpha \in [0, 1]$. Večji α pomeni, da je trenutni posnetek integriran v ozadje hitreje. S preizkušanjem različnih vrednosti α (Slika 4.1) smo ugotovili, da je za počasno spreminjanje okolja optimalna vrednost 0.003. Ta enostavna tehnika daje pri našem razmeroma statičnem ozadju zadovoljive rezultate, vendar ima tudi svoje pomankljivosti. Mehanizem prilagajanja ozadja obravnava vse slikovne elemente trenutnega posnetka enakovredno in jih s hitrostjo α integrira v ozadje. Z drugimi besedami, predmeti v ospredju so pri počasnem premikanju lahko čez čas integrirajo v ozadje. Pešec, ki se ustavi sredi klaviature in stoji tam dalj časa, tako postane del ozadja in ni več zaznan kot predmet v ospredju. To pomankljivost odpravimo, če za slikovne elemente, ki pripadajo predmetom v ospredju, uporabimo manjšo hitrost prilagajanja β, in sicer $\beta = 0,25\alpha$, ali pa celo $\beta = 0$ kot v [12], s čimer se izognemo neželjeni oskrunitvi ozadja.

Naslednja tehnika za ločevanje ozadja, ki jo implementira naša aplikacija, uporablja nekoliko zahtevnjejši model ozadja in rešuje zgoraj opisane težave na drugačne načine. Podrobnosti si oglejmo v naslednjem poglavju.



Slika 4.1: Enostavna tehnika ločevanja ozadja. Preizkušanje različnih vrednosti hitrosti prilagajanja α. Pri prevelikih vrednostih α se zazna samo obris predmetov v ospredju.

4.2 Napredna tehnika ločevanja ozadja

Z razliko od zaprtih prostorov, kjer se osvetlitev ne spreminja veliko, imamo na prostem pogosto situacijo veliko bolj kompleksnega ozadja. Okolje kot je cestišče oziroma prehod za pešce lahko vsebuje premikajoče predmete, kot

npr. premikanje sence drevesa, različne osvetlitve med dnevom, ki jih želimo vključiti v model ozadja. Potrebno je na nek način modelirati vsak posamezen slikovni element ali skupino preko določenega časovnega obdobja - zgraditi model ozadja.

Trenutno v literaturi najpopularnejša metoda - MOG, kjer je vsak slikovni element modeliran kot mešanica K Gaussovih porazdelitev, se je izkazala za neustrezno. Za obdelavo videa ločljivosti 640x480 v realnem času z zmogljivostjo povprečnega osebnega računalika je občutno prepočasna. Primernejša je tehnika ločavanja ozadja s kodirno knjigo (ang. codebook).

Tehnika s kodirno knjigo se za izgradnjo modela ozadja poslužuje metod kodiranja. Vzorci posameznega slikovnega elementa se tako združujejo v množico kodnih besed (ang. codewords), ki predstavljajo kompresirano obliko modela ozadja. Metoda je primerna tako za statična ozadja kot za ozadja s premikajočimi predmeti [13].

4.2.1 Izgradnja modela ozadja

Naj bo $\mathcal{X}$ zaporedje vzorcev posameznega slikovnega elementa sestavljeno iz N vektorjev RGB: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$. $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_L\}$ predstavlja kodirno knjigo slikovnega elementa sestavljeno iz L kodnih besed. Velikost kodirne knjige je odvisna od variranja zaporedja vzorcev slikovnega elementa, iz katerih se gradi model ozadja. Vsaka kodna beseda $\mathbf{c}_i, i = 1...L$ sestoji iz vektorja RGB: $\mathbf{v}_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$ in šestorčka $\mathbf{aux}_i = \langle \check{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \rangle$ spodaj opisanih spremenljivk.

$\check{I}, \hat{I}$: *največja* in *najmanjša* svetlost, zabeležena v kodirni knjigi

f : *frekvenca* pojavitve kodne besede

λ : najdaljši interval, ko se kodna beseda ni ponovila v času izgradnje modela ozadja

p, q : *prvi* in *zadnji* čas dostopa do kodne besede

Model ozadja gradimo za vsak slikovni element posebej in sicer iz zaporedja vzorcev $\mathcal{X}$. To v praksi pomeni nekaj sekund ali minut videa. Za vsak vzorec $\mathbf{x}_t$ se v kodirni knjigi poišče ustrezna kodna beseda $\mathbf{c}_m$, ki predstavlja šifrirano obliko vzorca. Za pogoj ustreznosti se uporabi merjenje barvnega odstopanja in svetlosti. Podrobnosti si oglejmo v nadaljevanju (Algoritem 1).

Pogoja sta izpolnjena, če sta barvi $\mathbf{x}_t$ in $\mathbf{c}_m$ dovolj blizu in če svetlost $\mathbf{x}_t$ leži znotraj dovoljenega območja svetlosti $\mathbf{c}_m$. Ker se večina variacij v ozadju zgodi v svetlosti in ne barvi, uporabimo barvni model, ki ima ločeni barvno

Algoritem 1 Izgradnja kodirne knjige

$L \leftarrow 0$, $\mathcal{C} \leftarrow \emptyset$
for $t = 1$ to N **do**
  $\mathbf{x}_t = (R, G, B)$, $I \leftarrow R + G + B$
  $m \leftarrow 0$
  for $i = 1$ to L **do**
    if $(colordist(\mathbf{x}_t, \mathbf{v}_i) \leq \epsilon_1)$**and**$(brightness(I, \langle \check{I}_i, \hat{I}_i \rangle) =$ **true**$)$ **then**
      {Poiščemo prvo kodno besedo $\mathbf{c}_m$, ki ustreza pogojema}
      $m \leftarrow i$
    end if
  end for
  if $(\mathcal{C} = \emptyset)$**or**$(m = 0)$ **then**
    {Ustvarimo novo kodno besedo $\mathbf{c}_L$}
    $L \leftarrow L + 1$
    $\mathbf{v}_L \leftarrow (R, G, B)$
    $\mathbf{aux}_L \leftarrow \langle I, I, 1, t-1, t, t \rangle$.
  else
    {Posodobimo najdeno kodno besedo $\mathbf{c}_m$}
    $\mathbf{v}_m \leftarrow (\frac{f_m \bar{R}_m + R}{f_m + 1}, \frac{f_m \bar{G}_m + G}{f_m + 1}, \frac{f_m \bar{B}_m + B}{f_m + 1})$
    $\mathbf{aux}_m \leftarrow \langle min\{I, \check{I}_m\}, max\{I, \hat{I}_m\}, f_m + 1, max\{\lambda_m, t - q_m\}, p_m, t \rangle$.
  end if
end for
for $i = 1$ to L **do**
  $\lambda_i \leftarrow max\{\lambda_i, (N - q_i + p_i - 1)\}$.
end for

in svetlostno komponento. Opaziti je, da so vzorci slikovnih elementov razporejeni okrog osi v smeri izhodišča (0,0,0). Uporabi se barvni model, upodobljen na sliki 4.2, ki omogoča ločeno vrednotenje barvnega in svetlostnega odstopanja. Vzorci slikovnih elementov ozadja ležijo v območju kodnih besed, omejenih z najmanjšo in največjo vrednostjo svetlosti. Imamo slikovni element $\mathbf{x}_t = (R, G, B)$ in kodno besedo $\mathbf{c}_i$, kjer je $\mathbf{v}_i = (\bar{R}_i, \bar{G}_i, \bar{B}_i)$, tako sledi:

$$\begin{aligned} \|\mathbf{x}_t\|^2 &= R^2 + G^2 + B^2 \\ \|\mathbf{v}_i\|^2 &= \bar{R}^2 + \bar{G}^2 + \bar{B}^2 \\ \langle \mathbf{x}_t, \mathbf{v}_i \rangle^2 &= (\bar{R}_i R + \bar{G}_i G + \bar{B}_i B)^2. \end{aligned}$$

Barvno odstopanje δ izračunamo s pomočjo p:

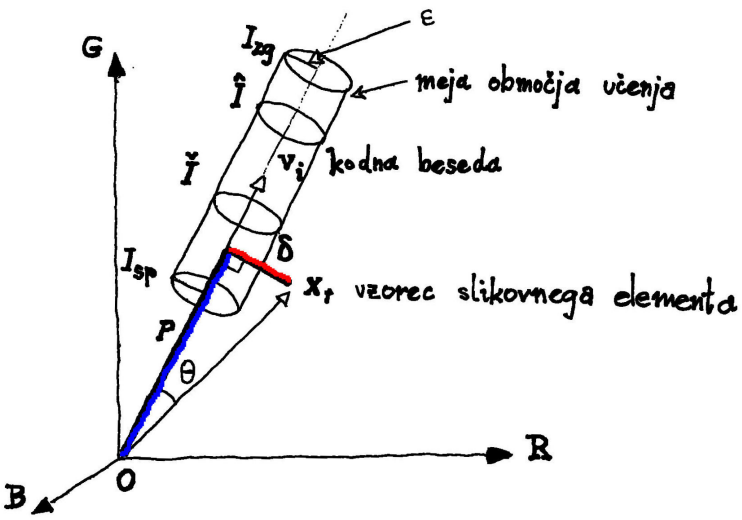
$$\begin{aligned} p^2 = \|\mathbf{x}_t\|^2 \cos^2\theta &= \frac{\langle \mathbf{x}_t, \mathbf{v}_i \rangle^2}{\|\mathbf{v}_i\|^2} \\ colordist(\mathbf{x}_t, \mathbf{v}_i) = \delta &= \sqrt{\|\mathbf{x}_t\|^2 - p^2}. \end{aligned} \tag{4.3}$$

Za spremembe v svetlosti so v šesterčku predvideni spremenljivki $\check{I}$ in $\hat{I}$, ki predstavljata zgornjo in spodnjo mejo svetlosti, ki jo pokriva določena kodna beseda. Meji imata še t.i. mejni območji učenja - I_{sp} in I_{zg}. Območje svetlosti se pri gradnji modela ozadja povečuje, če novi vzorci ozadja padejo znotraj mejnega območja. Če vzorec pade izven območja svetlosti in izven mejnih območij učenja, se za ta vzorec ustvari nova kodna beseda in s tem novo območje. $[I_{sp}, I_{zg}]$ se definira kot: $I_{sp} = \alpha\hat{I}, I_{zg} = min\{\beta\hat{I}, \frac{\check{I}}{\alpha}\}$, kjer je $\alpha < 1$ in $\beta > 1$. Vrednost α je v območju $0.4 - 0.7$, β pa v $1.1 - 1.5$. Funkcija svetlosti se definira:

$$brightness(I, \langle \check{I}, \hat{I} \rangle) = \begin{cases} \textbf{true} & \text{če velja } I_{sp} \leq \|\mathbf{x}_t\| \leq I_{zg} \\ \textbf{false} & \text{v ostalih primerih.} \end{cases} \tag{4.4}$$

Z zgoraj opisano tehniko za izgradnjo modela ozadja dobimo t.i. debelo kodirno knjigo. Sledi proces filtracije, med katerim ločimo kodne besede ozadja in kodne besede, ki pripadajo predmetom v ospredju. Z razliko od večine ostalih tehnik, je pri tehniki s kodirno knjigo dopustno, da so pri gradnji modela ozadja prisotni predmeti v ospredju. Temu služi parameter λ, ki se definira kot najdaljši interval, med katerim se kodna beseda ni ponovila v času izgradnje modela ozadja. Naj bo $\mathcal{M}$ model ozadja, ki ga dobimo po filtriranju:

$$\mathcal{M} = \{\mathbf{c}_m | \mathbf{c}_m \in \mathcal{C} \ \wedge \ \lambda_m \leq T_{\mathcal{M}}\}. \tag{4.5}$$



Slika 4.2: Kodna beseda v barvnem prostoru. Vzorec slikovnega elementa leži znotraj kodne besede, ki je omejena z barvnim odstopanjem (δ) in mejima svetlosti ($\check{I}, \hat{I}$).

Navadno je prag $T_{\mathcal{M}}$ enak polovici števila posnetkov, iz katerih smo zgradili model ozadja, torej $\frac{N}{2}$. (Polovičnemu času, med katerim smo gradili model ozadja.)

V skladu z enačbo 4.5 se kodne besede z velikim λ v procesu filtriranja odstranijo iz kodirne knjige. Čeprav ima kodna beseda visoko frekvenco f, velika vrednost λ pomeni, da gre najverjetneje za predmet v ospredju, ki je bil prisoten le za čas frekvence f. Nizka vrednost f in majhen λ pa sta značilna za dogodek v ozadju, ki se občasno ponavlja.

4.2.2 Ločevanje ozadja

Ločevanje trenutnega posnetka od ozadja je pri tehniki s kodirno knjigo enostavno. Z razliko od tehnik, kjer se uporablja izračun verjetnosti (mešanica Gaussov), se pri tej tehniki računa razdaljo vzorca slikovnega elementa do najbližje kodne besede. Pravzaprav gre za pogoja, ki smo ju srečali že pri gradnji kodirne knjige, to sta barvno odstopanje in svetlost. V kolikor se vzorec nahaja v bližini območja, ki ga pokriva katerakoli kodna beseda, se vzorec označi kot ozadje. V nasprotnem primeru je vzorec predmet v ospredju. Oddaljenost do območja določa parameter ϵ_2. Operacijo ločevanja ozadja $BGS(\mathbf{x})$ za slikovni

element **x** podrobno opisuje algoritem 2:

Algoritem 2 Ločevanje ozadja - BGS(x)

$\mathbf{x} = (R, G, B), I \leftarrow R + G + B$
$m \leftarrow 0$
for $i = 1$ to L **do**
  {Poiščemo prvo kodno besedo $\mathbf{c}_m$, ki ustreza pogojema}
  if $(colordist(\mathbf{x}, \mathbf{v}_i) \leq \epsilon_2)$**and**$(brightness(I, \langle \check{I}_i, \hat{I}_i \rangle) =$ **true**$)$ **then**
    $m \leftarrow i$
  end if
end for
if $m = 0$ **then**
  return *foreground*
else
  return *background*
end if

4.2.3 Implementacija

Za realizacijo modela ozadja s kodirno knjigo izberemo nekoliko poenostavljen algoritm, ki je opisan v knjigi "Learning OpenCV"[5].

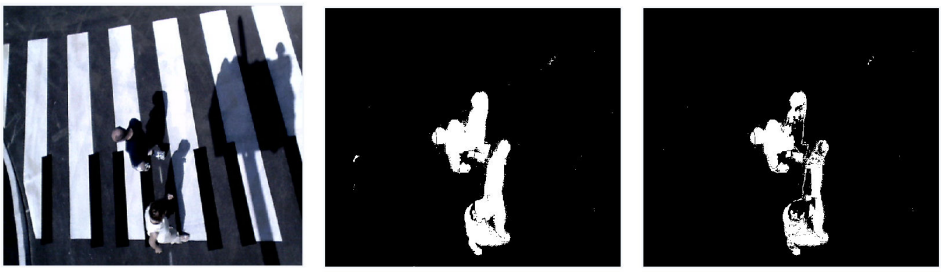
Uporaba barvnega modela RGB v praksi ni najbolj optimalna. Od modela RGB, kjer osi barvnega prostora predstavljajo tri barve, so primernejši barvni modeli, ki imajo osi uravnane s svetlostjo, kot npr. barvni model YUV (HSV je pravtako primeren barvni model, kjer je os V svetlost sama). Razlog za to tiči v tem, da se v praksi večina sprememb v ozadju dogodi v območju svetlosti, torej vzdolž osi svetlosti, ne pa v barvi. Z izbiro ustreznega barvnega modela se tako poenostavi predstavitev kodnih besed, ki smo jih natančneje opisali v prejšnjem poglavju. V barvnem modelu RGB so kodne besede območja valjastih oblik, usmerjene v izhodišče barvnega prostora, medtem ko se v modelu YUV, kodne besede predstavijo kot omejitve, torej zgornjo in spodnjo mejo spremenljivega območja, na vsaki od treh osi barvnega prostora. Pri ugotavljanju, ali slikovni element ustreza kodni besedi, pogoj barvnega odstopanja δ tako odpade, potrebujemo le še funkcijo svetlosti, ki določa spremenljivo območje. Vendar potrebujemo sedaj za vsako kodno besedo tri taka spremenljiva območja, za vsako os barvnega modela po eno. Računska zahtevnost algoritma se s to poenostavitvijo zmanjša na minimum, pravtako zasedenost pomnilnika, kar pa za delovanje aplikacije v realnem času ni nepomembno.

Podatkovna struktura kodne besede je naslednja:

```
typedef struct kb {
   uchar learnHigh[3];  //zgornja meja območja učenja
   uchar learnLow[3];   //spodnja meja območja učenja
   uchar max[3];        //zgornja vrednost območja svetlosti
   uchar min[3];        //spodnja vrednost območja svetlosti
   int t_last_update;   //zadnji čas dostopa do kodne besede
   int stale;           //najdaljši interval med katerim se kodna
                          beseda ni ponovila
} kodna_beseda;
```

Zgornjo in spodnjo mejo svetlosti ($\check{I}, \hat{I}$) predstavljata spremenljivki `max` in `min`, medtem ko sta `learnHigh` in `learnLow` zgornja in spodnja meja območij učenja (I_{zg}, I_{sp}).

Ključne funckije implementiranega algoritma so `update_codebook()` - s katero gradimo model ozadja, `clear_stale_entries()` - filtriranje modela ozadja in `background_diff()` - za ločevenja ozadja. Model ozadja je potrebno najprej zgraditi. S procesom filtracije odstranimo morebitne predmete v ospredju, ki so se "vgradili"v ozadje. S funkcijo za ločevanje ozadja nato ugotavljamo, ali določen slikovni element pripada predmetom v ospredju ali ozadju. Periodično ponovno kličemo fukcijo `update_codebook()`, s čimer posodabljamo model ozadja. S tem je zagotovljeno, da se ozadje prilagaja spremembam v okolju. V nekoliko daljših presledkih se s procesom filtracije odstranijo kodne besede, ki ne pripadajo ozadju.



Slika 4.3: Tehnika ločevanja ozadja s kodirno knjigo: Zaznava predmetov v ospredju s filtracijo (levo) in brez filtracije kodirne knjige (desno) po izgradnji modela ozadja.

4.3 Diskusija

Predstavitev enostavne in napredne tehnike za ločevanje ozadja zaključimo z medsebojno primerjavo.

Pri enostavni tehniki se za model ozadja vzame prvi posnetek, ki se nato sproti prilagaja trenutnem posnetku. Pri naprednjejši tehniki se najprej iz zaporedja posnetkov zgradi šifriran model ozadja, ki se ga nato periodično posodablja.

V idealnih pogojih - statično ozadje s hitro premikajočimi se predmeti v ospredju - deluje enostavna tehnika brezhibno. Zaradi sprotnega posodabljanja se je sposobna prilagoditi globalnim spremembam osvetlitve, kot npr. zatemnitev sonca. Vendar odpove v razmerah, ko imamo premikajoče predmete, ki so del ozadja, kot npr. konstantno premikanje sence drevesa ali ko se predmeti v ospredju premikajo prepočasi in se zato integrirajo v ozadje.

Naprednejša tehnika se izkaže ravno v teh razmerah, vendar je zaradi periodičnega posodabljanja nekoliko okornejša v prilagajanju. Ta problem pri naprednješi tehniki lahko rešimo, če zgradimo različne modele ozadja za posamezne razmere globalnih sprememb, recimo enega za dnevno svetlobo in enega za zatemnitev, ter nato uporabimo te razmere za izbiro ustreznega modela ozadja.

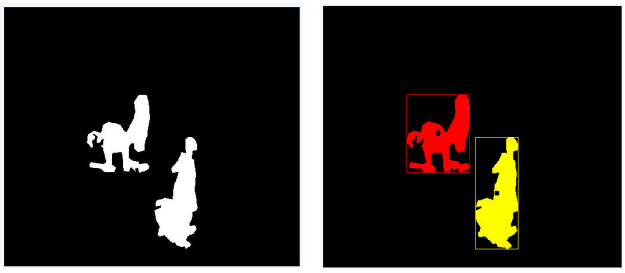
Kot kaže, je pri izbiri tehnike potreben kompromis med zmožnostjo prilagajanja spremembam in zahtevnostjo modeliranja ozadja. Obe tehniki delujeta zadovoljivo tako v zaprtih prostorih kot tudi na prostem, vendar se v določenih razmerah posamezna izmed njiju izkaže za boljšo. Na prvi pogled se zdi logično, da je enostavna tehnika primernejša za zaprte prostore, medtem ko je napredna primernejša za na prostem, vendar je ravno obratno. V zaprtem prostoru je veliko bolj verjetno, da se pešci ustavljajo na zebri, s čimer povzročajo enostavni tehniki probleme pri ločevanju ozadja. Po drugi strani pa je na prostem veliko več nepredvidenega spreminjanja, ki terja sunkovito prilagajanje, kot pa periodičnega variranja v samem ozadju, zaradi česar damo prednost napredni tehniki.

4.4 Označevanje povezanih področij

Rezultat ločevanja ozadja je binarna slika s kopico povezanih področij (ang. connected component, tudi blob, region), ki predstavljajo predmete v ospredju. V našem primeru gre za pešce. Povezana področja so predmet našega zanimanja, saj jih želimo podrobneje analizirati.

V splošnem obstajata dva pristopa pri označevanju povezanih področij: (ang. connected component labeling, tudi blob extraction, region labeling) rekurzivni in iterativni. Princip iterativnega algoritma, ki smo ga uporabili [14], je pregledovanje dveh vrstic hkrati, in sicer preverjanje podobnosti med trenutnim slikovnim elementom, tistim, ki leži levo od opazovanega, in slikovnim elementom, ki leži eno vrstico nad opazovanim. Hkrati mora obstajati tabela oznak, kjer je označeno, kateremu področju pripada kateri slikovni element. V tabeli se oznake med pregledom slike spreminjajo glede na povezanost elementov. Tak postopek zahteva le en obhod po sliki in en po tabeli oznak.

Povezano področje (ZBlob) se ponavadi opiše z mejnih okvirjem (ang. bounding box). Ogljišča mejnega okvirja se uporabijo za izračun središča povezanega področja (center). Pešec je zreduciran na središče območja, ki ga zaseda na sliki (Slika 4.4).



Slika 4.4: Označevanje povezanih področij. Na binarni sliki predmetov v ospredju (levo) sta identificirani dve povezani področji, ki se opišeta z mejnim okvirjem in središčem (desno).

4.5 Generiranje zaporedja MIDI

Osnova zaporedja MIDI je sporočilo za aktivacijo določene note (ang. MIDI Note-on message). V praksi to pomeni, da smo pritisnili tipko, ki jo določa notna številka MIDI. Tipke Klavirja za pešce se aktivirajo v skladu s položaji pešcev.

Na sliki imamo množico območij, ki predstavljajo tipke, in množico točk, ki predstavljajo pešce. Če se središče pešca (ZBlob.center) nahaja znotraj konture območja tipke (ZKey.keyContour), se generira sporočilo MIDI za aktivacijo notne številke (ZKey.note), ki pripada tipki. Tipka se označi kot aktivna (ZKey.noteOn = true), kar pomeni, da je ni možno aktivirati ponovno.

S tem se prepreči, da bi sprehod središča pešca čez območje tipke povzročil večkratno aktivacijo istega tona. V trenutku, ko se središče pešca ne nahaja več na območju tipke (tipka je prosta), se tipka sprosti (ZKey.noteOn = false) in možna je ponovna aktivacija.

Na ta način se poustvari situacija igranja na klavir. Če na tipko pritisnemo in jo držimo, na to tipko ni mogoče igrati, dokler je pritisnjena. Ponovna aktivacija je možna šele, ko tipko spustimo.

Podrobnosti generiranja zaporedja MIDI prikazuje algoritem 3:

Algoritem 3 Generiranje zaporedja MIDI

```
for all ZKey in ZKeyboard do
  keyDown ← false
  for all ZBlob in ZBlobs do
    if ZBlob.center is inside ZKey.keyContour then
      if ZKey.noteOn is false then
        Generiraj MIDI sporočilo z noto ZKey.note
        ZKey.noteOn ← true
      end if
      keyDown ← true
    end if
  end for
  if keyDown is false then
    ZKey.noteOn ← false
  end if
end for
```

Poglavje 5

Rezultati

Programsko opremo, ki podpira delovanje instalacije, želimo preizkusiti v praksi. V ta namen sta bila narejena dva modela Klavirja za pešce - v naravni velikosti na prostem in pomanjšan v zaprtem prostoru.

Prehod za pešce smo spremenili v klaviaturo na novozgrajenem, še ne odprtem križišču. Z dvižno košaro smo se povzpeli na višino 4m in z internetno kamero naredili več posnetkov pešcev, ki prečkajo cesto. Te posnetke smo uporabili za testiranje. Poleg resnične situacije smo naredili še pomanjšano različico, klaviaturo iz lepenke, v velikosti 20x30 cm, na kateri smo preizkusili delovanje programske opreme v realnem času. Z zatemnilnim stikalom, ki omogoča spreminjanje jakosti svetlobe, smo poustvarili različne osvetlitvene pogoje, ki se pojavljajo na prostem. Tako smo preverili robustnost sistema in zmožnost prilagoditve različnim razmeram.

Generiranje klaviature preizkusimo z naslednjimi parametri. Pri piramidni segmentaciji je $prag1 = 160$ in $prag2 = 80$. Za upragovanje belih oziroma črnih tipk se kot prag uporabi največja oziroma najmanjša vrednost slikovnih elementov območij po piramidni segmentaciji. Uspeh generiranja klaviature je odvisen od uspeha piramidne segmentacije.

Slika 5.1 prikazuje rezultat testiranja generiranja klaviature na modelu v naravni velikosti. Večina tipk je uspešno prepoznanih. Problem se pojavi pri eni črni in eni beli tipki zaradi močnih senc.

Na pomajnšanem modelu klaviature preizkusimo generiranje ob različnih osvetlitvah. Rezultati so na sliki 5.2, kjer je za vsak poiskus prikazan po en par posnetkov - zajeta slika z izrisanimi konturami prepoznanih tipk na levi in piramidna segmentacija na desni. Kot je razvidno iz slike 5.2, je prepoznana klaviatura popolna v polovici primerov, v dveh primerih so tipke nekoliko popačene, medtem ko v zadnjih dveh primerih klaviatura ni bila generirana v

celoti. Na uspelih primerih je lepo viden učinek aproksimacije kontur v konveksno ovojnico, s čimer se odpravijo napake pri segmentaciji zaradi svetlobnih odbojev.

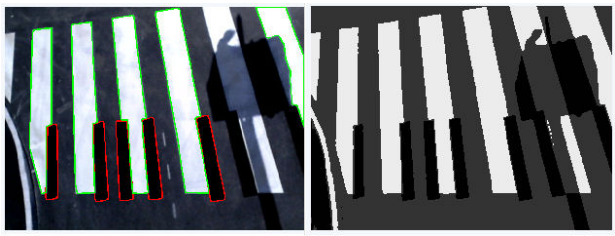
Postopek generiranja klaviature daje ob primerni osvetlitvi zadovoljive rezultate.

Enostavno tehniko za ločevanje ozadja preiskusimo s parametri: vrednost praga $T = 20$ in $\alpha = 0.003$. Pri tehniki s kodirno knjigo se za izgradnjo modela ozadja uporabi prvih 100 posnetkov ($N = 100$), poleg tega se model posodablja vsakih 500 posnetkov.

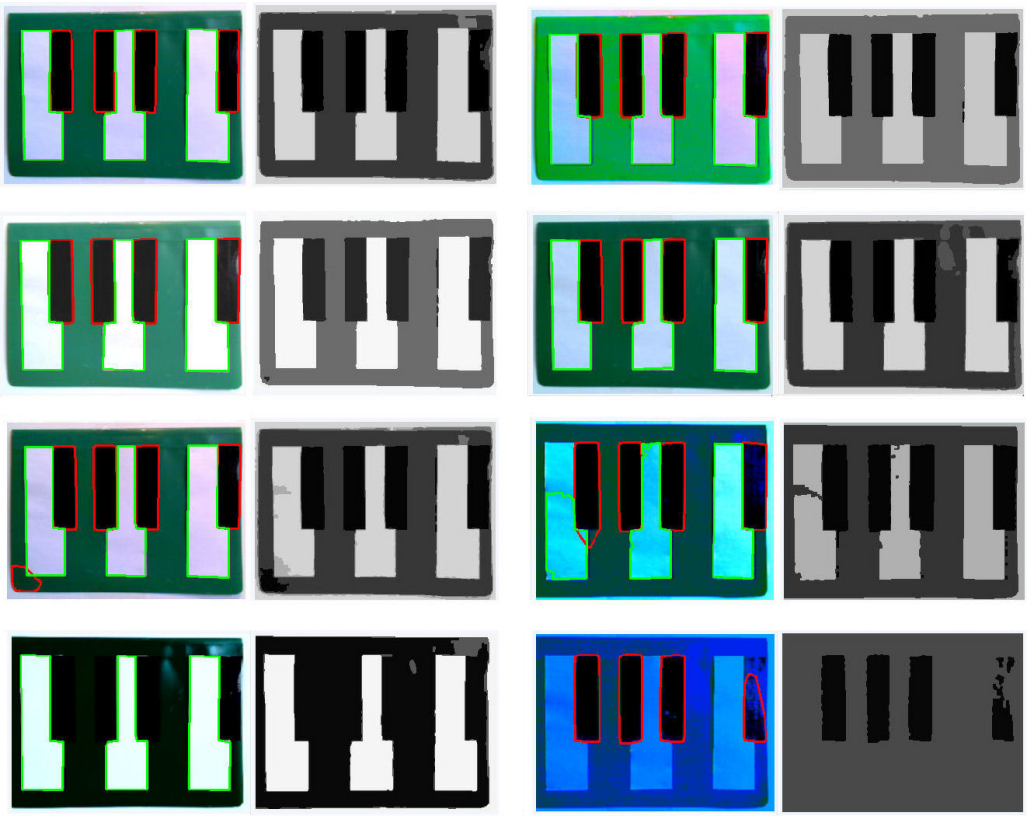
Obe tehniki za ločevanje ozadja se na posnetkih klavirja v naravni velikosti odrežeta brezhibno. Binarizirana oblika predmeta v ospredju je skoraj enaka pri obeh tehnikah, kot to prikazuje slika.

Na pomanjšanem modelu izpostavimo obe tehniki globalnim spremembam osvetlitve. Jakost svetlobe spremenimo od osvetlitve do delne zatemnitve. Enostavna tehnika potrebuje približno 15 sekund časa, da se prilagodi novim razmeram. Med tem časom delovanje klaviature ni pravilno, saj sprememba osvetlitve povzroči zaznavo predmetov v ospredju. Sosledje posnetkov prikazuje slika 5.4.

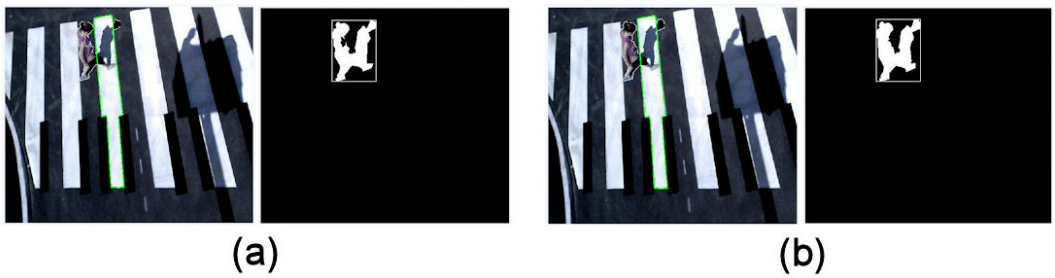
Pri napredni tehniki spremenimo jakost svetlobe najprej med izgradnjo modela ozadja. Na ta način model ozadja "navadimo"na spremembo. Ko spremembo osvetlitve ponovimo med delovanjem - ločevanjem ozadja, se sprememba ne zazna kot predmet v ospredju, ampak kot variacija ozadja. Na sliki 5.5 vidimo, da ostane binarizirana slika predmetov v ospredju enaka po spremembi osvetlitve. Napredna tehnika ločevanja ozadja deluje pravilno kljub globalni spremembi osvetlitve, vendar samo v primeru, če gre za predvidljivo spremembo, ki se predhodno vključi v model ozadja.



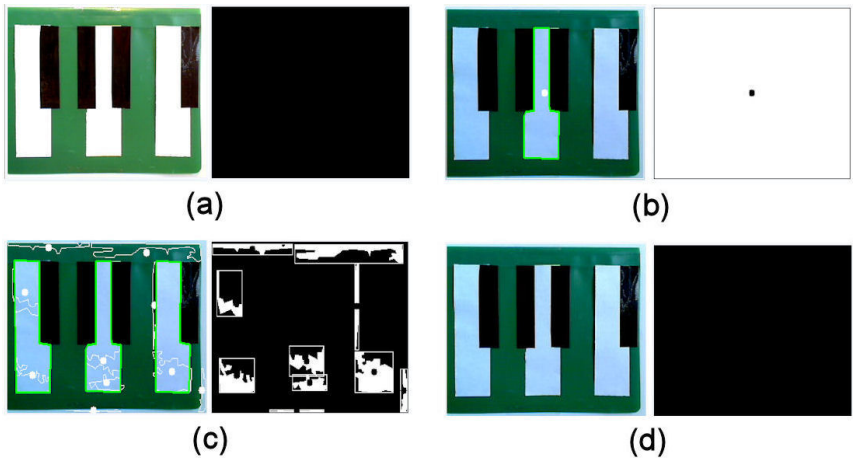
Slika 5.1: Generiranje klaviature v naravni velikosti. Izrisane konture prepoznanih tipk (levo) in piramidna segmentacija (desno).



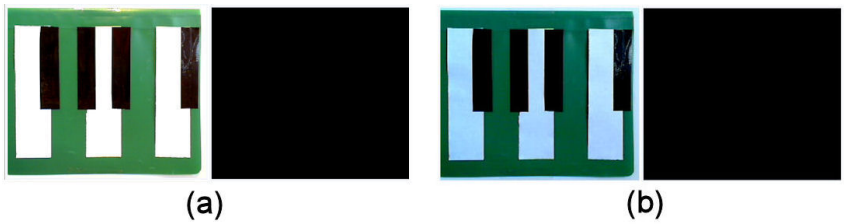
Slika 5.2: Generiranje klaviature ob različnih osvetlitvah na pomanjšanem modelu. Za vsak poizkus je prikazan po en par posnetkov; z izrisanimi konturami prepoznanih tipk (levo) in piramidno segmentacijo (desno).



Slika 5.3: Primerjava tehnik za ločavanje ozadja. (a) - enostavna tehnika, (b) - tehnika s kodirno knjigo. Originalna slika z aktiviranimi tipkami (levo) in binarna slika predmetov v ospredju (desno).



Slika 5.4: Prilagajanje enostavne tehnike za lovečanje ozadja. Od začetne osvetlitve - (a) do delne zatemnitve - (b), nepravilno delovanje - (c) in prilagoditev - (d).



Slika 5.5: Prilagajanje tehnike za ločevanje ozadja s kodirno knjigo. Od začetne osvetlitve - (a) do delne zatemnitve - (b); po predhodni vključitvi spremembe v model ozadja.

Poglavje 6

Zaključek

Z metodami računalniškega vida smo realizirali Klavir za pešce (Slika 6.1), interaktivno umetniško instalacijo, ki prehod za pešce spreminja v delujočo klaviaturo. Ideja je bila preizkušena na dveh modelih, vendar zaenkrat še ni doživela javne uprizoritve. Kljub temu se že porajajo številne zamisli za izboljšave in nadgradnjo osnovne ideje.

Klaviaturo smo generirali samo na začetku, saj smo prepodstavili, da je kamera fiksna. V primeru, da se kamera premakne (recimo zaradi vetra), se premaknejo tudi območja, ki pripadajo tipkam, zaradi česar klavir ne deluje več pravilno. Težava bi bila odpravljena, če bi programska oprema zaznala spremembo položaja tipk in ponovila generiranje klaviature med samim delovanjem.

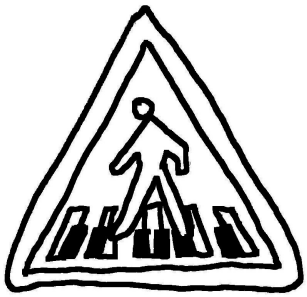
Tipka se aktivira v trenutku, ko se središče pešca nahaja znotraj območja tipke. To pomeni, da za aktivacijo tipke ni potrebno, da pešec dejansko stopi na tipko. Ključna izboljšava je torej segmentacija stopala pešca ob dotiku tal - aktivno stopalo. V idealnih razmerah; kamera vertikalno nad pešcem, brez senc in pretiranega mahanja rok, se aktivno stopalo lahko označi kot skrajna točka na konturi območja pešca v smeri gibanja, v trenutku, ko je razdalja med točko in središčem pešca največja. Problem je, da so idealne razmere praktično nedosegljive.

Samo delovanje klaviature bi lahko nekoliko popestrili, če bi integrirali dejansko sledenje pešcev (ne le ločavanje ozadja). Na ta način bi lahko vsakemu pešcu, naključno ali pa po nekem kriteriju, dodelili drugo glasbilo. Tipke bi pod pešcem A zvenele kot klavir, hkrati pa bi pešec B igral na kitaro. Glasba pešcev bi tako dobila polnejšo zvočno podobo.

Glasba pešcev pri prehodu čez cesto je pri trenutni izvedbi omejena na lokacijo instalacije oziroma na domet zvočnika, ki predvaja glasbo. Za širši

doseg poslušalcev bi bilo potrebno omogočiti poslušanje glasbe pešcev preko interneta. Za pretakanje zaporedja MIDI v realnem času preko omrežij se uporablja prenosni protokol RTP-MIDI. RTP (ang. Real-Time Transport Protocol) standardizira način prenosa avdio in video vsebin preko interneta. S tem se odpira možnost razširitve Klavirja za pešce v t.i. Orkester pešcev; več instalacij Klavirja na različnih lokacijah, kjer posamezna instalacija predstavlja določeno glasbilo, skupaj generira kompozicijo, ki bi jo lahko poslušali preko interneta.

Instalacija ima poleg umetniške tudi potencialno uporabno vrednost. Lahko se uporabi kot sredstvo oglaševanja storitve oziroma proizvoda, povezanega z glasbo; npr. glasbenega festivala ali proizvajalca glasbil.



Slika 6.1: Pozor! Klavir za pešce

Slike

Algoritmi

Literatura

[1] F. Solina, "Računalniški vid nekdaj in danes," v zborniku *Računalniška obdelava slik in njena uporaba v Sloveniji ROSUS 2006*, Maribor, Slovenija, mar. 2006.

[2] S. Juvan, F. Solina, B. Batagelj, P. Peer, "15 sekund slave - interaktivna umetniška inštalacija," v zborniku *Eleventh Electrotechnical and Computer Science Conference*, Portorož, Slovenija, sep. 2002.

[3] B. Batagelj, "Iskanje obrazov na osnovi barv s pomočjo statističnih metod razpoznavanja vzorcev," *magistrsko delo*, 2004

[4] B. Batagelj, "Prepoznavanje človeških obrazov s pomočjo hibridnega sistema," *doktorska disertacija*, 2007

[5] G. Bradski, A. Kaehler, *Learning OpenCV*, Sebastopol: O'Reilly, 2008.

[6] (2009) Computer music. Dostopna na: http://en.wikipedia.org/wiki/Computer_music

[7] (2009) Musical Instrument Digital Interface. Dostopno na: http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface

[8] (2009) Wrapper Library for Windows MIDI API. Dostopno na: http://www.codeproject.com/KB/audio-video/midiwrapper.aspx

[9] B. Jaehne, *Digital Image Processing*, Berlin: Springer-Verlag, 2005, str. 455-467

[10] S. Suzuki, K. Abe, "Topological Structural Analysis of Digital Binary Images by Border Following," CVGIP, št. 30, zv. 1, str. 32-46, 1985

[11] C. R. Wren, A. Azarbayejani, T. Darrell, A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE transactions on pattern analysis and machine intelligence*, št. 7, zv. 3, jul. 1997.

[12] D. Migliore, M. Matteucci, M. Naccari, "A revaluation of frame difference in fast and robust motion detection," v zborniku *Proceedings of the 4th ACM international workshop*, Santa Barbara, ZDA, 2006, str. 215-218

[13] K. Kim, T. H. Chalidabhongsb, D. Harwood, L. Davis, "Real-time foreground–background segmentation using codebook model," *Real-Time Imaging*, št. 11, zv. 3, str. 172-185, jun. 2005

[14] F. Chang, C. J. Chen, C. J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, št. 93, zv. 2, str. 206-220, feb. 2004