SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4315

Praćenje više lica

Vilim Šoštarić

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 8. ožujka 2016.

ZAVRŠNI ZADATAK br. 4315

Pristupnik: **Vilim Šoštarić (0036462941)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Praćenje više lica**

Opis zadatka:

Praćenje lica je postupak kojim se tehnikama računalnog vida slijedi ljudsko lice i njegove ključne točke u video slici. Ova tehnologija ima niz praktičnih primjena od kojih neke zahtijevaju istovremeno praćenje više lica vidljivih u video slici.
Na Zavodu za telekomunikacije dostupan je sustav za praćenje lica u stvarnom vremenu. Sustav može istovremeno pratiti samo jedno lice.
Vaša je zadaća proučiti postojeći sustav, te predložiti i implementirati modifikaciju sustava za praćenje više lica istovremeno.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.

Mentor:

Prof. dr. sc. Igor Sunday Pandžić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

SADRŽAJ

1. Uvod

U današnem svijetu tehnologija računalnog vida postaje sve zastupljenija i sve je češća pojava programa i alata koji se njome koriste. Sve poznatiji postaju programi i aplikacije koje koriste tehnologiju praćenja ljudskoga lica, takozvani *face tracking*, te obradu tih podataka u razne svrhe. Neki od primjera takvih programa su virtualna šminka, zamjena lica na slici (*face swap*), praćenje pogleda na zaslonu, kontroliranje virtualnih 3D lica pokretima vlastitoga lica te analiza lica kojom se mogu odrediti spol, dob i emocije osobe.

Sve većom zastupljenošću ove tehnologije na današnjem tržištu pojavili su se zahtjevi za programima koji zahtjevaju paralelno praćenje više lica istovremeno. Cilj ovog rada je ostvariti taj zahtjev te, proširenjem koda već postojećeg *face trackera*, napraviti alat za višestruko praćenje lica (*Multiple Face Tracker*) koji bi se mogao koristiti na raznim platformama. Temelj ovoga rada je skupina alata VisageSDK tvrtke Visage Technologies i biblioteka OpenCV.

2. VisageSDK

Skup alata VisageSDK tvrtke Visage Technologies nudi korisnicima tehnologiju računalnoga vida i praćenja lica razvijenu za vrlo jednostavno integriranje i korištenje u aplikacijama na raznim platformama [1]. Funkcionalnosti koje ti alati omogućuju korisnicima uključuju detekciju i praćenje lica, praćenje pogleda, analiza lica (određivanje emocija, spola te dobi osobe) i upravljanje 3D modela lica pomoću računalnog vida. Temelj ovog rada je projekt *FaceTracker2*.

Projekt *FaceTracker2* demonstrira detekciju i praćenje lica iz slike, videa ili kamere te pronalazi karakteristične točke na licu i prikazuje ih na zaslonu. Glavni razredi iz projekta *FaceTracker2* korišteni u ovom radu su *VisageTracker* i *FaceData*.

2.1. VisageTracker

VisageTracker je alat za praćenje koji omogućava praćenje pozicije glave u tri dimenzije, karakterističnih točaka na licu te smjer pogleda videa, kamere ili nekog drugog medija. Rezultati dobiveni iz praćenja lica spremaju se u objekt razreda *FaceData*. Funkcije razreda *VisageTracker* korištene u ovom radu su:

- `VisageTracker(char* defaultConfigFile)`
  Konstruktor, `defaultConfigFile` je putanja konfiguracijske datoteke koja u sebi sadrži parametre trackera potrebne za njegov rad. Konfiguracijska datoteka koju se općenito koristi je `Facial Features Tracker - High.cfg`.
- `~VisageTracker()`
  Destruktor.
- `track(int frameWidth, int frameHeight, const char *p_imageData, FaceData *facedata, int format = 0, int widthStep = 0, long timeStamp = -1)`
  Obavlja operaciju praćenja lica i vraća rezultate punjenjem objekta `facedata` razreda `FaceData` i status trackera.

Argumenti `frameWidth` i `frameHeight` su širna i visina slike na kojoj se prati lice. Potrebni su zbog početne alokacije memorije za podatke slike prilikom prvog poziva funkcije ili promjene veličine slike. Argument `p_imageData` je pokazivač na niz vrijednosti koje predstavljaju RGB (red, grean and blue) vrijednosti pojedinih piksela slike. Argument `format` određuje kako su spremljene vrijednosti piksela u slici, pretpostavljena vrijednost 0 odgovara formatu u RGB obliku. U objekt `facedata` se pune rezultati funkcije (podatci o praćenom licu).

Ova se funkcija treba zvati na svaku novu sliku dobivenu iz videa ili kamere kako bi praćenje bilo konzistentno. Razlog tome je način rada funkcije koja se interno dijeli na dva glavna dijela:

- Funkcije `DetectFacialFeatures` koja pronalazi najveće lice na dobivenoj slici i vraća njegove podatke. Poziva se na prvom pozivu funkcije `track` te nakon što tracker izgubi praćeno lice. Ako uspije pronaći lice vraća status `TRACK_STAT_OK`, u suprotnom vraća `TRACK_STAT_RECOVERING` ili `TRACK_STAT_INIT` nakon što više uzastopnih slika nije pronašla lice. Funkcija se zove na svaku sljedeću sliku dok lice na slici nije pronađeno.
- Praćenja već pronađenog lica iz prošle slike (u slučaju da je dobiveni status prošle funkcije bio `TRACK_STAT_OK`). U slučaju da više ne pronađe praćeno lice na slici vraća status `TRACK_STAT_RECOVERING` nakon kojeg se u sljedećoj slici zove funkcija `DetectFacialFeatures`. Praćenje već pronađenog lica je višestruko puta brže od funkcije `DetectFacialFeatures`.

2.2. FaceData

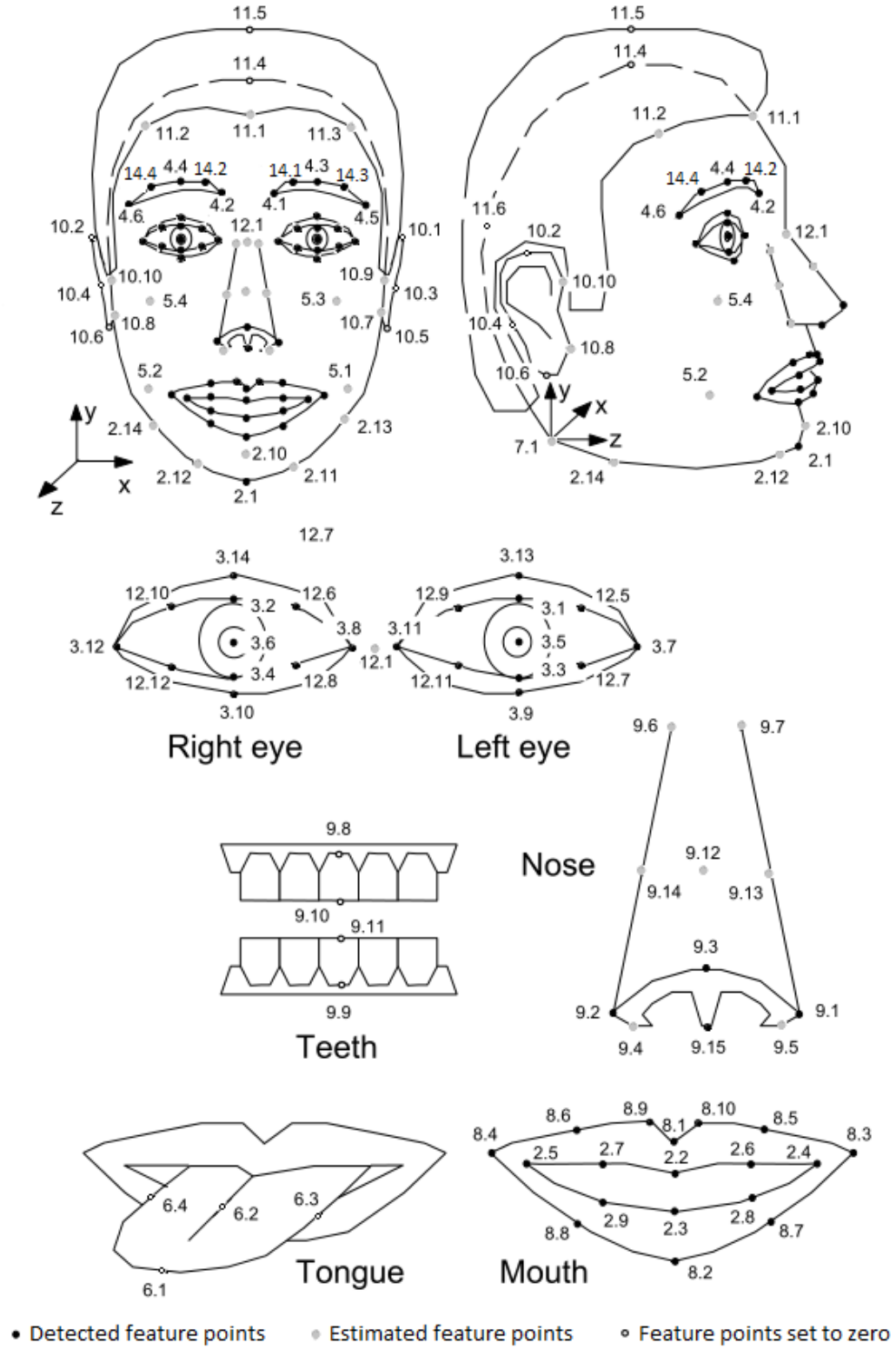
Razred *FaceData* je struktura koja služi za pohranu podataka praćenog lica dobivenig iz funkcije `track` razreda *VisageTracker*. Pri popunjavanju ove strukture neki podatci se popunjuju, dok drugi ostaju nepopunjeni ovisno o statusu koji vraća funkcija koja ju popunjuje.

Glavne skupine podataka koje se popunjuju su:

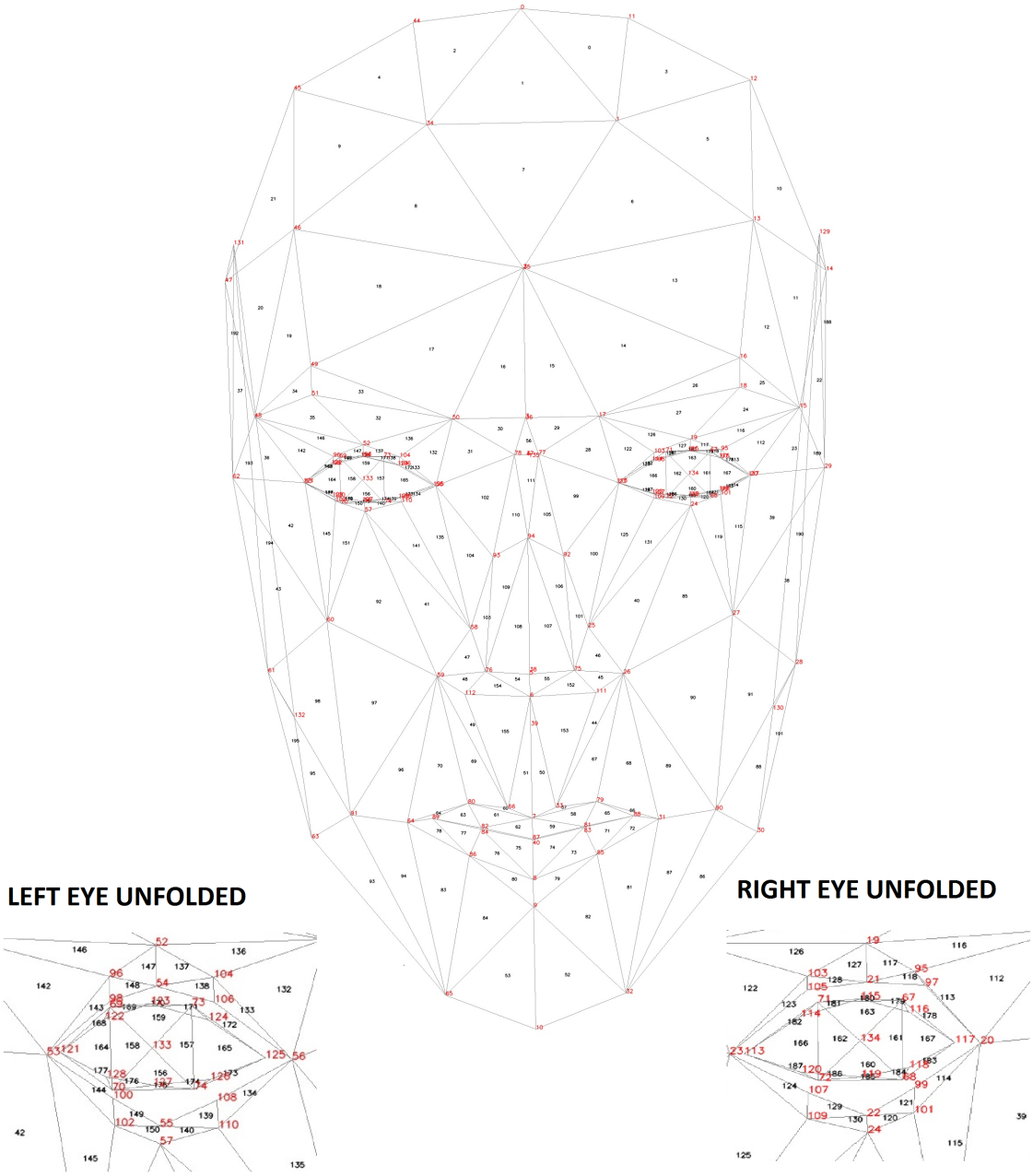
- 3D položaj glave
  opisan je translacijom i rotacijom glave u 3D prostoru. Translacija se sprema u niz od tri realna broja koji odgovaraju x, y i z koordinati središta lica (točka između očiju). Rotacija glave također se sprema u niz od tri realna broja, oni odgovaraju rotaciji u radijanima oko x, y i z osi koordinatnoga sustava.
- karakteristične točke lica
  koje se spremaju u struktire FDP(Face Definition Parameters) koje pohranjuju karakteristične točke lica prema MPEG-4 FBA (Face and Body Animation) standardu. Pojedine točke na licu indeksirane su brojem grupe točaka i indeksom u toj grupi, na primjer točka označena sa 9.3. odgovara vrhu nosa kao što je pokazano na slici 2.1.

  Varijable koje sadrže te točke su:

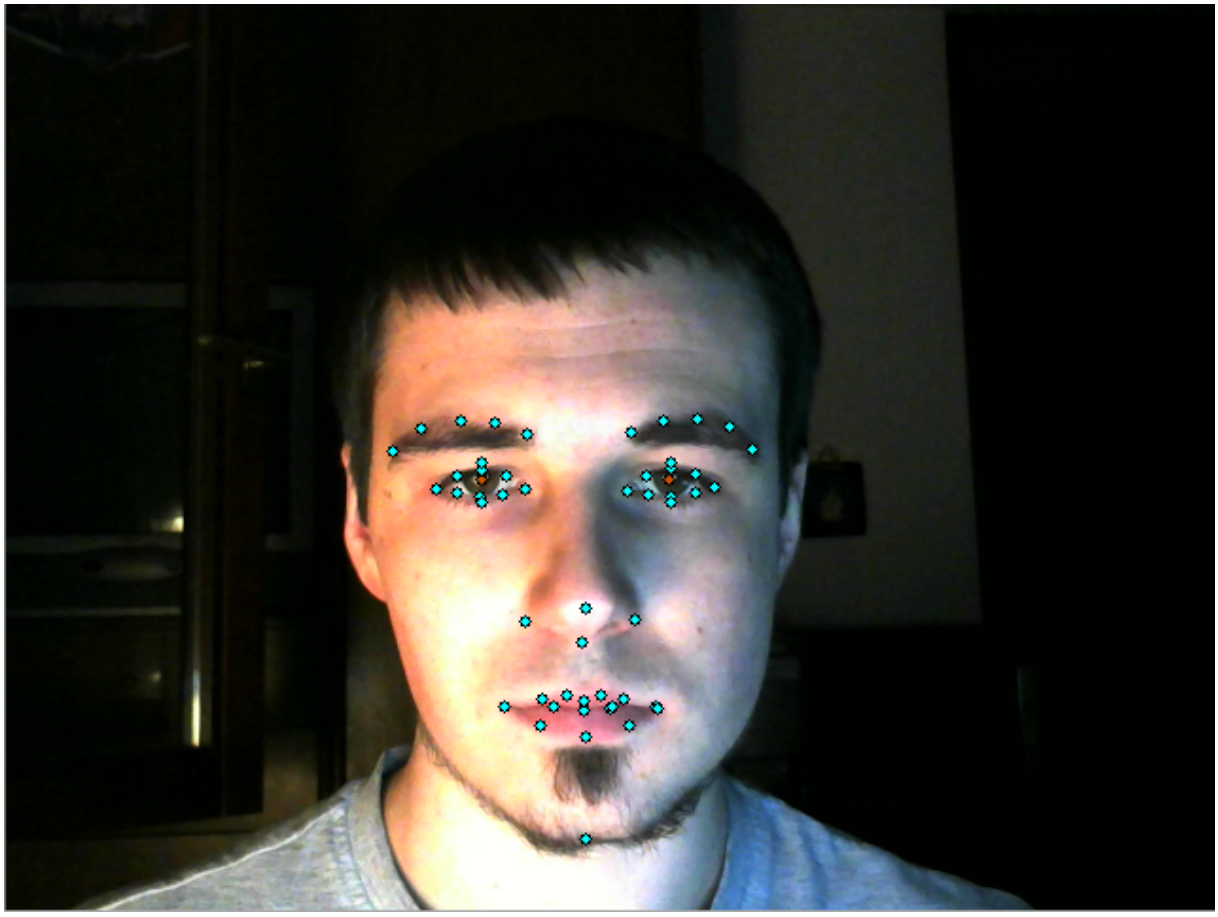
  - `featurePoints3D`
    3D koordinate karakterističnih točaka na licu u globalnom prostoru.
  - `featurePoints3DRelative`
    3D koordinate karakterističnih točaka u odnosu na centar lica (između očiju, točka označena sa 12.1. na slici 2.1.).
  - `featurePoints2D`
    2D koordinate točaka normalizirane s obzirom na sliku sa koordinatama (0,0) u doljnjem lijevom kutu, a (1,1) u gornjem desnom kutu.
- 3D model lica koji se može animirati i koristi se interno u trackeru. Model je namješten u 3D prostoru na lice u slici. Sadrži značajno više točaka od onih spremljenih iz `featurePoints` varijabli te se može koristiti za mnogo složenije zadatke. Model je prikazan na slici 2.2.



Slika 2.1: Karakteristične točke lica prema MPEG-4 FBA standardu



Slika 2.2: 3D model lica korišten u trackeru



Slika 2.3: Primjer iscrtanih rezultata trackera

3. OpenCV

OpenCV (Open Source Computer Vision Library) je open-source biblioteka koja sadrži stotine algoritama računalnoga vida. Razni moduli ove biblioteke omogućuju obradu slike i videa, estimaciju 3d oblika objekata, jednostavno korištenje raznih ulaznih i izlaznih uređaja, algoritami za ubrzavanje korištenjem grafičke kartice i drugi [2].

U ovom radu korišten je modul *highgui*. To je modul za jednostavno korištenje ulaznih i izlaznih uređaja te obradu slike. Razredi i funkcije iz modula korištene u ovom radu su:

- `IplImage`
  razred služi za spremanje i obradu slike. Uzet je izravno iz Intelove bliblioteke " *Intel Image Processing Library*". Sadrži podatke o parametrima slike kao visina, širina, broj kanala slike (sa koliko su podataka opisani pikseli slike) i drugi.
- `cvPoint`
  razred koji sadrži x i y koordinatu točke.
- `cvScalar`
  razred koji opisuje boju piksela.
- `cvCircle(CvArr*img, CVPoint center, int radius, CvScalar color, int thickness=1, int line_type=8, int shift=0)`
  funkcija koja crta kružnicu ili krug izravno na sliku. Prima pokazivač na sliku `img`, središte `center`, polumjer `radius`, boju `color` (izraženu na primjer u RGB formatu), debljinu linije `thickness` (ako se preda vrijednost -1 nacrta se puni krug).
- `cvCapture`
  razred koji služi za korištenje kamere ili videa.
- `cvLoadImage(const char* filename)`
  funkcija koja prima putanju slike, otvara ju i popunjuje njezine podatke u objekt razreda `IplImage`.

- `cvNamedWindow(const char* name)`
  funkcija koja otvara novi prozor.
- `cvQueryFrame(CvCapture *capture)`
  funkcija koja uzima sliku iz kamere ili videa i puni njene podatke u objekt razreda `IplImage`.
- `cvFlip(const CvArr* src, CvArr* dst=(CvArr*)0, int flip_mode = 0)`
  funkcija koja prima pokazivač na sliku i na temelju `flip_mode` parametra okreće sliku po x , y ili obje osi.
- `cvCloneImage`
  funkcija koja kopira jedan objekt razreda `IplImage` u drugi.
- `cvShowImage(const char* name, const CvArr* image)`
  funkcija koja crta sliku `image` u prozor imena `name`.
- `cvReleaseImage(IplImage** image)`
  funkcija koja oslobađa memoriju i briše objekt koji sadrži sliku.
- `cvReleaseCapture(CvCapture** capture)`
  funkcija koja otpušta kameru ili video i oslobađa zauzetu memoriju.

4. Implementacija algoritma

Cilj ovoga rada je napraviti alat za praćenje više lica iz kamere, videa ili slike. To će se postići stvaranjem novog razreda *MultiTracker* koji će predstavljati dodatan sloj apstrakcije između izvornog koda *VisageSDK-a* i korisničkih programa. Kako bi algoritam *MultiTrackera* bio primjenjiv, potrebno je da bude što brži (da vrijeme obrade jedne slike i nalaženja lica na istoj bude što manje). Taj zahtjev ćemo postiči iskorištavanjem današnjih procesora sa više jezgri i višedretvenošću algoritma (paralelnog izvođenja pojedinih `track` funkcija višestrukih *VisageTrackera*). Uz sam *MultiTracker* biti će napravljen i jednostavan program koji koristi *MultiTracker* i prikazuje njegovu funkcionalnost i performansu. Algoritam je pisan u jeziku C++.

4.1. ExtendedFaceData

Ovaj razred je jednostavno proširenje razreda `FaceData` iz VisageSDK. Njegovi atributi su:

- `VisageSDK::FaceData faceData`
  struktura podataka za pohranu rezultata *trackera*.
- `int id`
  identifikacijski broj koji povezuje strukturu *faceData* sa odgovarajučim *trackerom* te tako održava konzistentnost podataka u *MultiTrackeru*. Postavljen u konstruktoru i ne mijenja se.

Funkcije razreda *ExtendedFaceData* su:

- `ExtendedFaceData(int givenId)`
  konstruktor funkcija koja od *MultiTrackera* prima novi identifikacijski broj.
- `~ExtendedFaceData()`
  destruktor funkcija.
- `ExtendedFaceData::ExtendedFaceData(const ExtendedFaceData &obj)`
  copy-konstruktor.

4.2. TrackerHandler

TrackerHandler je razred koji predstavlja glavnu funkcionalnost algoritma. Upravlja ulaznim i izlaznim podatcima te ponašanjem njegovog *VisageTrackera*. Njegovi atributi su:

- `IplImage *myImage`
  služi za pohranu kopije originalne slike te za daljnju obradu slike prije `track()` funkcije njegovog *trackera*.
- `int m_Format`
  označava format slike (broj kanala), postavljen da označava korištenje RGB formata.
- `VisageSDK::VisageTracker *m_Tracker`
  pokazivač na vlastiti *tracker* kojem jedino ovaj *TrackerHandler* objekt može pristupiti.
- `int detectTimeOut`
  brojač koji pokazuje broj slika od zadnjeg poziva `track` funkcije njegovog `m_Trackera` u slucaju da taj *tracker* ima status `TRACK_STAT_INIT`. Kada brojač dođe do određene granice (postavljeno 5) funkcija `track` se pozove i brojač se postavi na 0. To osigurava da se funkcija `DetectFacialFeatures` ne zove na svaku sliku, jer bi u suprotnom brzina rada *MultiTrackera* značajno pala. Tako se u slučaju traženja novoga lica skupa funkcija `DetectFacialFeatures` zove tek svaku petu sliku.
- `int m_Id`
  identifikacijski broj koji koristi za *trackeru* da prepozna svoje izdane podatke (svoj `ExtendedFaceData` iz vektora podataka svih aktivnih trackera) sa prošle slike te za popunjavanjem rezultata na pravilno mjesto.
- `int m_Status`
  korišten za spremanje statusa svog *trackera*.
- `VisageSDK::FaceData faceData`
  struktura za privremeno spremanje rezultata *trackera* u svojoj dretvi.

Funkcije ovog razreda su:

- `TrackerHandler(int id, char* defaultConfigurationFile)`
  konsruktor funkcija. Parametri su mu njegov identifikacijski broj koji se ne mijenja i putanja konfiguracijske datoteke potrabne za stvaranje njegovog *trackera*. Poziva konstruktore *VisageTracker* i *FaceData* objekata kojima samo ova instanca razreda *TrackerHandlera* ima pristup.
- `~TrackerHandler()`
  destruktor funkcija, poziva destruktor funkcije njegovog *trackera* i *faceDate*.
- `TrackerHandler::TrackerHandler(TrackerHandler &obj)`
  copy-konstruktor.
- `grabImage(IplImage *origImage, vector<ExtendedFaceData*> PreviousFaceData)`
  funkcija koja se u *MultiTrackeru* poziva kao zasebna dretva, obrađuje dobivenu sliku i konrolira ponašanje *trackera* te instance *TrackerHandlera*. Funkcija kopira dobivanu sliku u vlastiti `myImage`, iterira po rezultatima svih *trackera* sa prošle slike te zacrnjuje lica (koristeći funkciju `cvCircle`) svih *ExtendedFaceData* objekata u vektoru `PreviousFaceData` kojima identifikacijski broj ne odgovara njegovom identifikacijskom broju. Rezultat ovoga postupka je slika koja sadrži samo lice ili lica koja nisu već praćena. Tako obrađena slika se predaje funkciji `track` od `m_Trackera` koja puni svoje rezultate u atribut `faceData`. Nakon poziva briše se slika `myImage` i oslobađa se alocirani memorijski prostor funkcijom `cvReleaseImage`.

4.3. MultiTracker

MultiTracker je glavni razred ovoga algoritma napravljen za jednostavno pokretanje i primjenu (identično kao i originalni *VisageTrackeri*). Atributi ovoga razreda su:

- `int lastId`
  zadnji izdan identifikacijski broj, povećava se pri stavaranju nove insance *TrackerHandlera* te joj se predaje. Tako se održava konzistentnost podataka i osigurava da svaki *TrackerHandler* ima različit identifikacijski broj.
- `char* defaultConfigFile`
  putanja do konfiguracijske datoteke koja se predaje konstruktorima *VisageTrackera*.
- `int numOfFaceData`
  atribut u kojem se pohranjuje broj praćenih lica na slici.
- `vector<TrackerHandler*> vectorOfTrackers`
  vektor (niz sa dodatnim ugrađenim funkcijama) pokazivača na instance *TrackerHandlera*. Vektor je struktura podataka temeljena na običnom nizu, koja ma ugrađene funkcije za jednostavno stvaranje i brisanje pojedinih instanci, kao i atribute veličine vektora i pozicioniranje.
- `vector<ExtendedFaceData*> multiFaceData`
  vektor pokazivača na instance *ExtendedFaceData*.

Funkcije razreda *MultiTracker* su:

- `MultiTracker(char *InputDefaultConfigFile)`
  konstruktor funkcija. Prima putanju do konfiguracijske datoteke potrabnu za stvaranje *trackera*. Postavlja varijablu `lastId` na 0.
- `~MultiTracker()`
  destruktor funkcija. Briše sve instance *TrackerHandlera* i *ExtendedFaceData* te oslobađa sav alocirani memorijski prostor.
- `track(IplImage *origImage, vector<ExtendedFaceData>* outFaceData, int maxFaces)`
  funkcija koja se poziva iz glavnog programa. Prima pokazivač na originalnu sliku, pokazivač na vektor instanci *ExtendedFaceData*, koji na kraju popunjava sa dobivenim rezultatima, te maksimalan dopušten broj praćenih lica. Funkcija iterira po vektoru instanci *TrackerHandlera* te stvara dretve koje izvode funkciju `grabFrame` razreda *TrackerHandler* za svaku instancu koja ispunjava jedan određenih uvjeta. Postoje dva uvjeta:
  - Instanca *TrackerHandlera* već prati jedno lice (njezin atribut `m_Status` je `TRACK_STAT_OK`).
  - Instanca *TrackerHandlera* traži novo lice (njezin atribut `m_Status` je `TRACK_STAT_INIT`) i prošlo je pet slika od kako je `track` funkcija njezinog *trackera* bila pozvana zadnji put.

  Sve stvorene dretve se spremaju u vektor pokazivača na dretve. Nakon stvaranja dretvi funkcija čeka da sve dretve završe sa svojim radom te ih uništava. Instance *TrackerHandlera* koje su pronašle (ili nastavile pratiti) svoje lice popunjavaju svoje podatke u vektor `multiFaceData`, dok instance koje su izgubile svoje lice (atribut `m_Status` je `TRACK_STAT_INIT`) se brišu i oslobađaju alocirani memorijski prostor. Ako sve instance prate neko lice (svim preostalim instancama je atribut `m_Status` je `TRACK_STAT_OK`) i broj instanci nije došao do granice `maxFaces`, stvara se nova instanca *TrackerHandlera* i stavlja se u vektor `vectorOfTrackers` koji će tražiti novo lice. Na kraju se popunjava vektor `outFaceData` dobivenim rezultatima i funkcija vraća broj praćenih lica. Tako se osigurava da uvjek postoji samo jedan *tracker* koji traži novo lice čime je konzistentnost podataka održana i uvjek postoji samo jedan više *tracker* od broja praćenih lica.

5. Primjer programa

U nastavku će biti opisan primjer programa koji koristi alat *MultiTracker*. Program je jednostavna Win32 consloe aplikacija koja ima mogućnost praćenja više lica iz kamere ili slike. Također program u konzolnom prozoru ispisuje broj praćenih lica, broj aktivnih *trackera* te vrijeme izvršavanja funkcije `track` od *MultiTrackera*. Za dobivanje slike iz kamere, otvaranje spremljene slike, potrebnu obradu slike i krajnje iscrtavanje rezultata *MultiTrackera* koristi se `highgui` modul biblioteke *OpenCV*. Tijek programa može biti objašnjen u par koraka:

1. Na početku izvršavanja program nudi korisniku opciju praćenja lica iz kamere ili sa spremljene slike.
   Ako je odabrano praćenje sa slike, datoteka slike se otvara funkcijom `cvLoadImage` te se sprema u varijablu tipa `IplImage` te se pita korisnika koliko maksimalno lica smije biti praćeno. To omogućava provjeru performansi *MultiTrackera* u ovisnosti o broju praćenih lica.
   U slučaju odabira kamere, funkcijom `cvCaptureFromCam` se dohvaća kamera i pokazivač na kameru se sprema u varijablu tipa `CvCapture`.

2. Stvara se prozor funkcijom `cvNamedWindow`. Prozor će dalje koristiti da iscrtavanje rezultata *MultiTrackera*.

3. Poziva se konstruktor razreda *MultiTracker* i sprema se pokazivač na tu instancu.

4. Početak glavne `while` petlje programa. U sličaju odabrane kamere na početku petlje se funkcijom `cvQueryFrame` dobavlja trenutna slika iz kamere, sprema u varijablu tipa `IplImage` te se funkcijom `cvFlip` ta slika zrcali u odnosu na y os.

5. Stvara se prazan vektor pokazivača na instance razreda *ExtendedFaceData*. Ovaj vektor se kasnije popunjuje rezultatima dobivenima iz *MultiTrackera*.

6. Poziva se funkcija `track` razreda *MultiTracker* pri čemu joj se predaje slika za obradu, pokazivač na prazan vektor *ExtendedFaceData* razreda te broj koji određuje maksimalan broj praćenih lica. Funkcija popunjuje vektor primljenog pokazivača svojim rezultatima (podatcima o praćenim licima) i u konzolu ispisuje broj praćenih lica i broj aktivnih *trackera*.

7. Temeljem dobivenih rezultata na sliku se crtaju karakteristične točke svih praćenih lica (koristi se funkcija `cvCircle`).

8. Slika se prikazuje u prozoru funkcijom `cvShowImage`.

9. Slika se briše i program se vraća na korak 4.

Slika 5.1: Rezultat praćenja lica sa slike

6. Performanse MultiTrackera

Jedan od ciljeva ovoga rada je *MultiTracker* napraviti prihvatljivim za opću i komercijalnu upotrebu. Taj cilj podrazumijeva dva važna uvjeta:

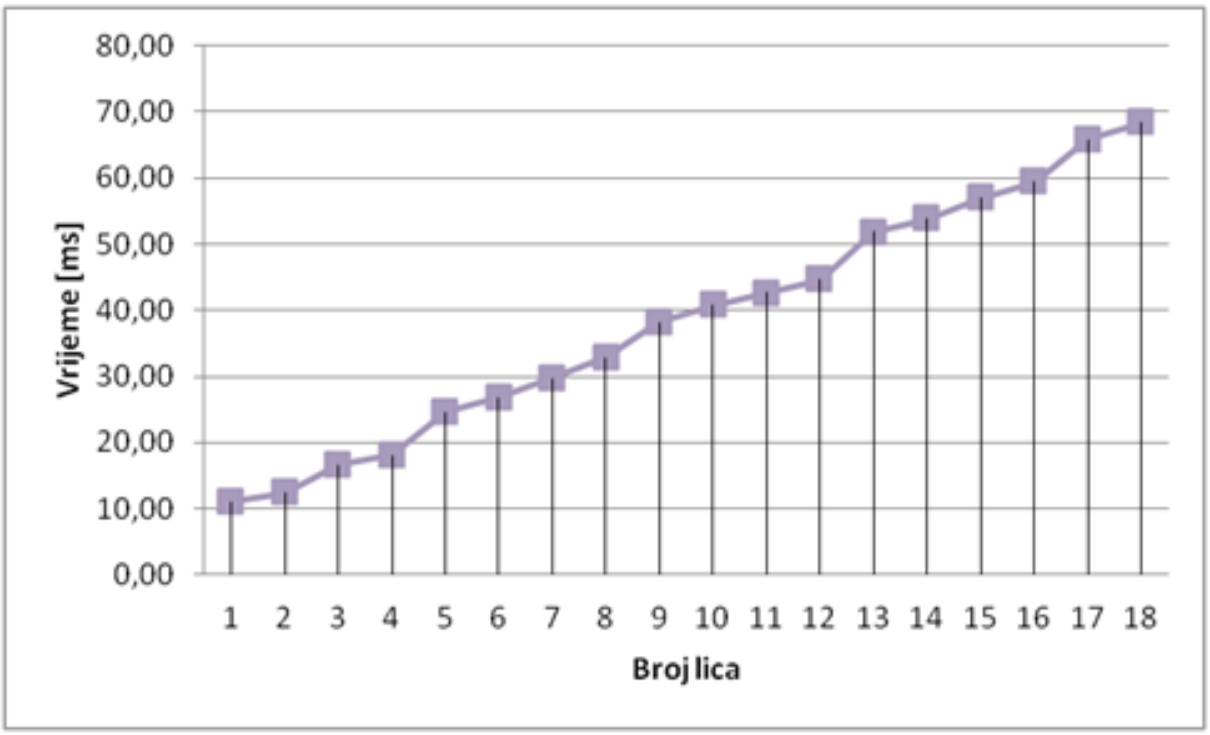
1. *MultiTracker* mora biti pracizan te točno mora detektirati i pratiti lica sa slike. Samo jedan *tracker* smije pratiti određeno lice sa slike te broj *trackera* smije biti maksimalno samo za jedan veći od broja praćenih lica.

2. Funkcija `track` *MultiTrackera* mora se izvoditi u što kračem vremenu kako nebi uzrokovala značajno kašnjenje programa koji ju pozivaju (na primjer pri praćenju lica iz kamere nesmije se primjetiti zastajkivanje izlazne slike).

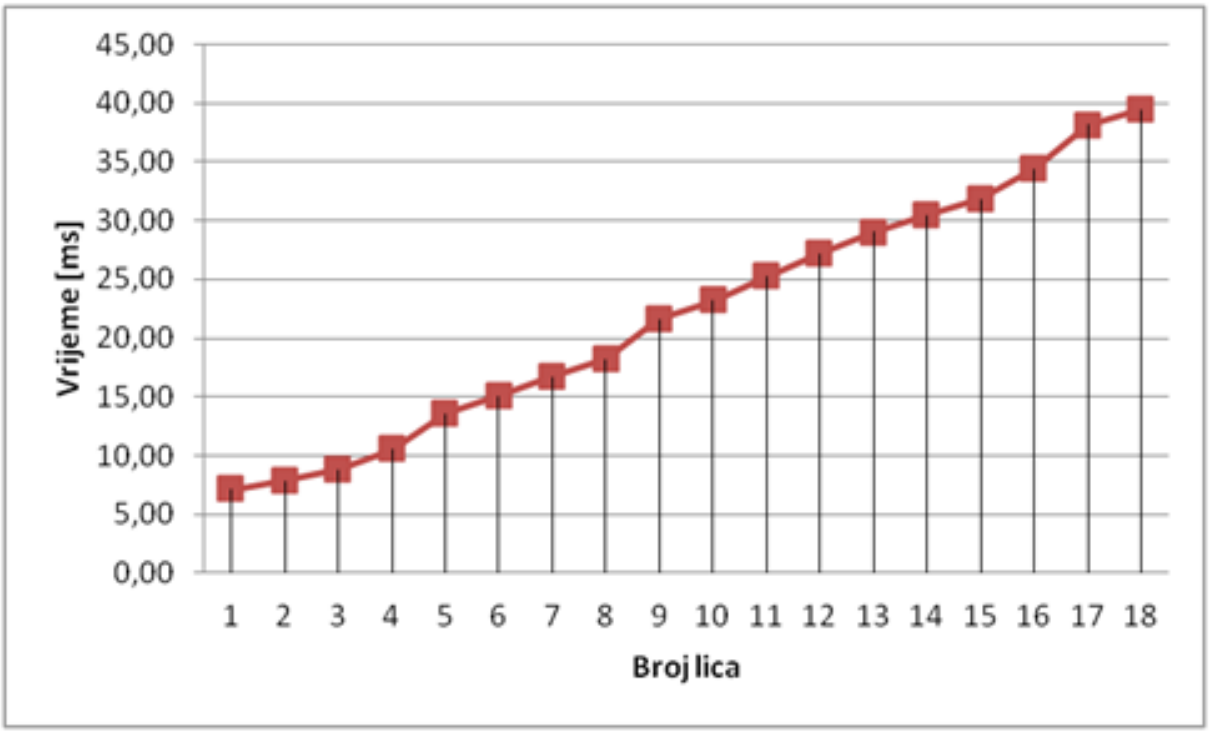
6.1. Analiza

Ispunjenje prvog uvjeta vidi se na slici 5.1. iz prošloga poglavlja. Nema višestrukog crtanja karakterističnih točaka na licu te karakteristične točke se točno crtaju na izlaznoj slici.

Ispunjenje drugog uvjeta osigurava višedretvenost *MultiTrackera* kojom se iskorištavaju višestruke jezgre na modernim procesorima. Vrijeme `track` funkcije samo blago raste dok je broj aktivnih *trackera* manji od broja jezgri procesora. Kada broj *trackera* prijeđe broj jezgri primjećuje se značajan porast izlaznog vremena (otprilike vrijeme izvođenja jednog *trackera*). To se događa zato jer se nemogu sve dretve izvesti istovremeno, dio dretvi se izvede paralelno dok dio čeka njihov završetak kako bi se izvele. Taj skok u izlaznom vremenu vidljiv je na svaki prelazak broja praćenih lica od višekratnika broja jezgri u procesoru. Ta se pojava vidi na slici 6.2 gdje se očitava značajan skok između 4. i 5. te 8. i 9. lica četverojezgrenog procesora te na slici 6.1. gdje se značajan skok očitava između svakog parnog i neparnog broja lica na dvojezgrenome procesoru.



Slika 6.1: Vremena funkcije track na dvojezgrenom procesoru (1.8GHz)



Slika 6.2: Vremena funkcije track na četverojezgrenom procesoru (3.5GHz)

6.2. Detalji mjerenja

Grafovi sa slika 6.1 i 6.2 su dobiveni mjerenjem vremena *MultiTrackera* na slici 5.1 uz promjene maksimalnog broja praćenih lica. Osim značajnih skokova vremena u prije obijašnjenim slučajevima, blagi porasti vremena pojavljeju se zbog vremena potrebnog za stvaranjem dretvi, pripremom slike za praćenje u jednoj dretvi, koja podrazumijeva zacrnjivanje ostalih praćenih lica na slici funkcijama `cvCircle` čiji broj poziva u jednoj dretvi raste brojem ostalih praćenih lica. Mora se također uvažiti da vremena na grafu imaju određena odstupanja zbog ostalih dretvi operativnoga sustava koje se izvode paralelno sa ovim programom.

7. Zaključak

Programiranje alata za praćenje više je kompleksan zadatak koji zahtijeva puno vremena i koncentracije. Paralelizacija programa zahtjeva puno pažnje na razna granične slučajeve koji na koje treba posebno obratiti pažnju. Cilj ovoga rada je uspiješno obavljen sa dosta dobrim preformansama te je spreman za opću upotrebu. Postoji mogućnost daljnje optimizacije i prilagodbe algoitma za više platformi (trenutno je ograničen na sustav Windows zbog načina stvaranja zasebnih dretvi) te izbacivanje stranih biblioteka (*OpenCV*). Također je problematično stvaranje *trackera* pošto svaki alocira zasebno memorijski prostor za potrebne podatke korištene pri praćenju lica te promjena koda *trackera* da dijele podatke koji su svakoj instanci identični bi bila poželjna.Postoji mogućnost i definiranja slučaja kada je zacrnjivanje lica doista potrebno, a kada se može izostaviti, no to otvara mogućnost za mnoge greške i neočekivano ponašanje te se traba jako dobro razmotriti i definirati.

LITERATURA

[1] Visage Technologies. VisageSDK documentation.
http://www.visagetechnologies.com
[2] OpenCV. OpenCV documentation.
http://opencv.org/documentation.html

Praćenje više lica

Sažetak

Praćenje lica iz kamere, videa ili drugog medija podrazumijeva analizu i obradu dobivene slike te određivanje karakterističnih točaka na pronađenome licu. Algoritam za praćenje više lica temeljen je na alatu koji prati jedno lice uz paralelizaciju glavnih funkcija te dodavanjem mehanizama sinkronizacije i dodatnih obrada ulaznih i izlaznih podataka. U svrhu paralelizacije programa koristi se višedretvenost algoritma gdje svaki pojedini *tracker* se odvija u zasebnoj dretvi. Kako bi se algoritam pravilno izvodio, svaka dretva mora zacrniti sva druga lica prije praćenja svoga lica. Algoritam ima vrlo dobru brzinu izvođenja te je adekvatan da primjenu u komercijalne svrhe.

Ključne riječi: praćenje lica, računalni vid, karakteristične točke, višedretvenost, dretva, obrada slike, OpenCV, FDP.

Multiple face tracking

Abstract

Face tracking from camera, video file or an other source consists of analysing and processing the given image, then estimating the feature points of the detected face. The algorithm for multiple face tracking is based on a tool for singe face tracking with added parallel processing of critical functions and synchronisation and input-output processing mechanisms. To achieve parallel processing the algorithm uses multi-threading where every *tracker* is running in his own thread. To ensure that the algorithm is executing correctly, every thread must darken every other face except it's own in the given image before tracking it's face. The algorithm has a farley good execution speed and is adequate for commercial purposes.

Keywords: face tracking, computer vision, feature points, multithreading, thread, image processing, OpenCV, FDP.