SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 521

DETEKCIJA I PRAĆENJE ZNAČAJKI LICA U WEB OKRUŽENJU

Petar Stojanac

Zagreb, Lipanj 2013.

Na drugu stranicu stavite izvornik zadatka završnog rada.

Prazna stranica ili zahvala.

Sadržaj

Uvod

Detekcija i praćenje objekata odnosno lica u realnom vremenu i u scenama bez ograničenja dobiva sve više na značaju i privlači investitore sa svojim mnogobrojnim potencijalnim primjenama. Problem je težak i računalno zahtjevan zbog različitih varijacija u sceni poput osvjetljenja, okluzije, kompleksnih pozadina ili šuma, varijacija u izgledu poput boje kože, poze te zbog same činjenice da ljudsko lice može biti vrlo različito od čovjeka do čovjeka. Raznim pristupima pokušava se riješiti problem, od pokušaja da se izradi generalni parametrizirani model, statističkih podataka pa sve do kompleksnijih tehnika učenja. Svaki od pristupa ima svoje prednosti i nedostatke.

Grana detekcije i praćenja lica je kroz razne industrijske primjene poput industrije igara i filma, usprkos tome što je relativno nova (oko 16 godina), sazrela do zavidne razine na stolnim računalima. To je djelomično zato što su stolna računala moćnija i mogu podnijeti zahtjeve za računalnom snagom. Danas, s druge strane, fokus se prebacuje više na mobilne aplikacije i Web, potaknut razvojem sklopovlja i programske podrške na prijenosnim uređajima i javlja se potreba za detekcijom objekata, lica, praćenjem značajki te praćenjem pogleda (eng. *gaze tracking*) na Webu.

Poglavlje 1 detaljnije opisuje koji alati i tehnologije su se koristili pri ostvarivanju detekcije i praćenja na Webu te se pokušavaju približiti razlozi zašto Web tehnologije privlače sve veći interes ulagača. U poglavlju 2 opisani su neki od poznatih algoritama detekcije implementirani u bibliotekama koje su se koristile. Isto tako, postavlja se kontekst u kojemu su se algoritmi razvijali kroz povijest i smjer preko kojemu razvoj teži. Također je opisana i podgrana računalnog vida i detekcije, praćenje objekata u video toku podataka. Poglavlje 3 obuhvaća opis problematike prilikom konverzije koda na Web platforme i navodi tehnike koje su se koristile kako bi se riješili problemi. Navode se postignuti rezultati te mogući smjer razvoja u budućnosti.

1. Alati i tehnologije

1.1. Zašto Web tehnologije?

Cilj programera je što efikasnije i efektivnije obaviti određeni posao, a to podrazumijeva minimizaciju pisanja istoga koda nanovo. Drugi vrlo važan cilj je dobiti proizvod koji se može isporučiti što široj populaciji i interesnim skupinama te na različitim platformama. Različite platforme, s druge strane, zahtijevaju od programera da se ista funkcionalnost implementira u različitim jezicima što se kosi s prvom premisom.

Jedan način za isporuku proizvoda širokoj interesnoj skupini su Web aplikacije. Web aplikacije se pokreću unutar Web preglednika koji se nalaze na gotovo svim platformama i operacijskim sustava. Kao jezik za izradu stranica koristi si HTML (eng. *HyperText Markup Language*), uz skriptni jezik JavaScript koji se izvršava na strani korisnika. Bitna značajka HTML i JavaScript jezika je da se pridržavaju određenih standarda (ECMAScript) što programerima daje određenu sigurnost i olakšava posao. Isto tako postoje organizacije (W3C, WHATWG) čiji posao je razvijati i definirati standarde (HTML5). Treba spomenuti i platforme Adobe Flash te Microsoft Silverlight koji u obliku dodatka Web pregledniku donose mogućnost izrade igara te aplikacija s bogatim sadržajem poput grafike, animacija te interaktivnog multimedijalnog sadržaja.

Razlog zašto je razvoj aplikacija danas više usmjeren na Web aplikacije jest razvoj tehnologije. S jedne strane, razvoj sklopovlja i pokretnih uređaja te njihova prihvatljiva cijena rezultirali su time da se danas kod većine ljudi može pronaći pametni prijenosni uređaj (eng. *smartphone*). S druge strane, usporedno sa sklopovljem razvija se i programska podrška. Primjerice HTML5 standard implementira nove elemente s kojima se omogućava reprodukcija videa, zvuka ili slike. Između ostalog uvode se novi koncepti poput paralelizacije (eng. *WebWorkers*), komunikacije (eng. *WebSockets*) te se konstruiraju programska sučelja (eng. *FileAPI*).

Poboljšanja koja dolaze i tendencija da se ljudi sve više služe svojim prenosivim pametnim uređajima, u kombinaciji sa dostupnošću bilo kada i bilo gdje, čine Web sve interesantnijom platformom za razvijatelje aplikacija, kako poslovnih tako i zabavnih.

1.2. Korišteni alati i tehnologije

Korištene tehnologije i alati vrlo su raznovrsni. Krećući s najniže razine korišteno je više različitih programskih jezika. Jedan od razloga zašto je korišten veći skup jezika je prilično jednostavan, a to je da svaki jezik ima svoju namjenu pa su tako neki jezici prikladniji za određene zadatke. Drugi razlog je što su već postojale implementacije kompleksnih segmenata u različitim programskim jezicima, a njihovo prebacivanje u unificirani jezik bilo bi vremenski neizvedivo u trajanju diplomskog rada.

Budući da ručno prebacivanje koda zahtjeva previše vremena koriste se alati za prevođenje poput Emscriptena. Jedna varijanta algoritama za praćenje i detekciju lica je već implementirana unutar razvojnog okruženja Visage u jezicima C/C++. Druga varijanta algoritma za detekciju je implementirana u CCV biblioteci za računalni vid u programskom jeziku JavaScript, a različit algoritam za detekciju u programskom jeziku C/C++.

Korišteni su i alati za kreiranje datoteka kaskada. Kaskade se koriste u procesu detekcije lica, a alati se isporučuju unutar CCV biblioteke. Vezano za treniranje kaskada potrebno je kreirati ispravne setove slika u kojima se nalaze lica u određenom formatu te setove slika bez lica. Prethodno je napravljen alat za anotaciju velikog skupa slika, a kako bi se svi podaci i slike kategorizirali koristila se SQLite baza podataka te sustav za upravljanje bazom podataka integriran u Web preglednik. Slike se obrađuju pomoću skripti napisanih u programskom jeziku Python.

1.2.1. Emscripten

Emscripten je alat za prevođenje (eng. *compiler*), napisan u JavaScriptu, licenciran pod besplatnom MIT licencom, a njegov autor je Alon Zakai, zaposlenik u Mozilli. Razvoj alata započeo je 2010 godine, a do danas su uvedene mnoge optimizacije kako bi prevedeni kod bio što brži i istoznačniji kodu iz kojega se prevodi. Primjeri, upute, podrška te zajednica koja se bavi razvojem i korištenjem alata nalazi se na stranici *GitHub* od kuda se također može i preuzeti izvorni kod[1].

Usprkos tome što je relativno nov prevoditelj, već je korišten u mnogo velikih projekata. Vezano za igre, između ostalog, prevedeni su potpuni program za upravljanje grafičkim prikazom *Unreal Engine 3* te pokretački sustav za fiziku *Bullet*. Prevedene su i uslužne biblioteke poput *zlib* i *sql* biblioteka, biblioteke za skeniranje barkoda, biblioteka za vizualizaciju, itd.. S ciljem omogućavanja naprednije grafike na Webu podržano je prevođene iz *OpenGL ES 2.0* u *WebGL*. Uz to, u velikoj mjeri, je podržana C++11 specifikacija te su uspješno prevedeni programski jezici poput Pythona, Ruby ili Perl. Prevođenje kompletnog jezika koristeći Emscripten ima prednosti u odnosu na korištenje ručno prevedenog podskupa. Ako se jezik kompletno automatski prevede onda se osigurava isto ponašanje na Webu kao i u samoj konzoli dok ručno prevedeni jezici obično predstavljaju podskup izvornog jezika te ne garantiraju istu ponašanje na svim razinama. Isto tako, ručno prevedeni jezici, ne rijetko kasne za posljednjom verzijom izvornog koda, dok se pomoću prevoditelja zadnja verzija jezika u potpunosti prevodi.

Emscripten prevodi LLVM asemblerski kod u JavaScript. LLVM (*Low Level Virtual Machine*) je također prevoditelj, najviše fokusiran na C, C++ i Objective-C kod. Za prevođenje tih jezika u asemblerski kod niske razine koristi se alat *Clang*. Bilo koji jezik koji se može prevesti u LLVM kod može se potencijalno prevesti pomoću Emscriptena u JavaScript. Druga mogućnost je prevođenje samog koda za parsiranje i izvođenje nekog jezika u LLVM kod te onda u JavaScript. Tako se omogućava korištenje tog jezika na Webu u slučaju da sam jezik nije prevediv u LLVM kod, ali postoji neka njegova implementacija pisana u jeziku koji je prevediv. To je primjerice slučaj s jezikom Python.

[1]Stranica gdje se može preuzeti izvorni kod - https://github.com/kripken/emscripten/wiki

Slika 1 Postupak prevođenja koda

Postupak prevođenja unosi određene probleme. Jedan od problema je zadržavanje kontrolne strukture koda. Prevođenjem u LLVM asemblerski kod niske razine gube se strukture poput petlji i uvjeta te se kod organizira u blokove. Da bi se postigle dobre performanse u JavaScriptu nužno je koristiti takve strukture i zato Emscripten ima zadatak rekonstruirati strukture visoke razine iz koda niske razine i to u različitom jeziku od izvornog. U tu svrhu koristi se *„Relooper“* algoritam iz kojega proizlaze kontrolne strukture optimizirane za moderne JavaScript pokretače.

Drugi problem je dobiti konzistentan kod tj. postići da prevedeni kod daje iste rezultate i ima isto ponašanje kao i kod u izvornom jeziku, a da se u tom procesu što manje žrtvuju performanse. Ovo nije trivijalno postići jer su JavaScript i, primjerice, C++ dva različita jezika s različitom semantikom koje nije lako uskladiti. Primjer su operacije s brojevima i pojava preljeva (eng. *overflow*) te zaokruživanje. Problemu se pokušava pristupiti tako da se vrši emulacija, ali zbog toga što je ona računalno zahtjevna, postoje alati koji pomažu otkriti kada je potrebna tako da bi se njeno korištenje svelo na minimum.

Emscripten pokušava što efikasnije prevesti LLVM asemblerski kod stoga se svaka linija asemblerskog koda pokušava prevesti u jednu liniju JavaScripta. To je moguće postići s operacijama poput poziva funkcija, alociranja varijabli na stogu ili s nekim operacijama poput zbrajanja. Jedan primjer gdje to nije moguće je alokacija na gomili (dinamička alokacija memorije u C++). U prevedenom JavaScriptu gomila je predstavljena jednim velikom poljem, a pristupanje tome polju preko indeksa je ekvivalentno pristupanjem gomili preko pokazivača u C++.

Upravljanje datotekama poput čitanja i pisanja je također proces koji se razlikuje u Web pretraživaču. Naime, Web pretraživač, u odnosu na operacijski sustav, nema sustav za

upravljanje datotekama tako da Emscripten implementira virtualni sustav za upravljanje datotekama i integrira ga tako da nije potrebno mijenjati izvorni kod kako bi se postigla ista funkcionalnost.

Pitanje je kakve su performanse koda nakon svih optimizacija i emulacija koje je potrebno uvesti da bi prevedeni kod bio ekvivalentan izvornom kodu. Neslužbeni testovi performansi na manjim bibliotekama pokazuju obećavajuće rezultate. Dobivene vrijednosti, prezentirane u članku o Emscriptenu, prikazuju usporenje od 2.47 do 8.46 puta u odnosu na optimizirani izvorni kod. Vrijednosti na realnim projektima prikazuju usporenje od 5 do 20 puta [1]. Treba uzeti u obzir da vrijednosti ovise o Web pregledniku.

benchmark	SM	V8	gcc	ratio
fannkuch (10)	1.158	**0.931**	0.231	4.04
fasta (2100000)	**1.115**	1.128	0.452	2.47
primes	**1.443**	3.194	0.438	3.29
raytrace (7,256)	**1.930**	2.944	0.228	8.46
dlmalloc (400,400)	5.050	**1.880**	0.315	5.97

Slika 2 Testovi performansi na manjim bibliotekama (Firefox, Chrome, Izvorni kod)[2]

Pristup Emscriptena je postići ravnotežu između brzine i strukture prevedenog koda , zbog čega se uvode neka ograničenja na što se može prevesti poput 64-bitnih objekata tipa *int* te višedretvenog koda. JavaScript ne podržava višedretveni model sa zajedničkom memorijom, a 64-bitni objekti tipa *int* su podložni zaokruživanju zbog toga što su svi brojevi u JavaScriptu implementirani kao 64-bitni objekti tipa *double*.

Budući cilj je poboljšati performanse i brzinu prevedenog koda. To se planira postići daljnjim radom na optimizaciji samog Emscriptena i načina na koji prevodi varijable i strukture. S druge strane, razvoj pretraživača i JavaScript pokretača povlači i ubrzanje prevedenog koda. U razvoju je i podskup jezika JavaScript imena „*asm.js*“ čije karakteristike su visoke razina optimizacije te potpuna kompatibilnost prema unazad. Uvode se metode detekcije tipova i novi načini koji pomažu u kontroli strukture programskog toka. Rezultati ranih testova performansi pokazuju obećavajuće rezultate u obliku usporenja od dva puta u odnosu na izvorni kod.

[2] Preuzeto iz [1]

1.2.2. Visage|SDK

Visage|SDK je programsko razvojno okruženje koje integrira niz tehnologija iz područja kompjuterskog vida i animacije virtualnih likova, a razvija ga tvrtka Visage Technologies[3]. Najznačajnije za ovaj rad su tehnike detekcije i praćenja lica sadržane u paketima *FaceTrack, HeadTrack* i *FaceDetect*.

Za parametrizaciju i modeliranje ljudskog lica Visage koristi izmjenjivi 3D model naziva Candide [2]. Geometrija modela je određena vrhovima s vrijednostima koordinata u koordinatnom sustavu modela. Candide se koristi jer je javno dostupan te jednostavan sa 190 definiranih trokuta i 17 dodatnih parametara koji se koriste kod deformacije modela.

Ciljevi algoritma praćenja su detekcija globalne pozicije i rotacije glave te značajki lica iz okvira u okvir iz video toka podataka (video datoteka ili tok podataka iz kamere). Osnovna detekcija glave i značajki lica izvršava se u prvom okviru koristeći Viola−Jones algoritam (detaljnije u poglavlju 2.1.1) implementiran u *OpenCV* biblioteci. Slijedi prilagodba 3D modela i odabir značajki lica koje će se pratiti iz okvira u okvir koristeći normaliziranu križnu korelaciju (eng. *normalized cross-correlation*). Informacije dobivene procesiraju se u proširenom informacijskom filteru kako bi se dobila procjena parametra modela lica. U slučaju da lica nije moguće pratiti pokreće se mehanizam oporavka i ponovne detekcije.

Jedan od problema s kojima se susreću algoritmi praćenja lica bazirani na estimaciji pomaka piksela iz jednog okvira u drugi, pa tako i ovaj, je tendencija da se male greške nakupljaju zbog čimbenika poput promjene u osvjetljenju, okluzije ili šuma, što rezultira sve lošijem praćenjem kako se povećava broj okvira. Kako bi se spriječilo ovakvo ponašanje Visage algoritam uvodi dodatni korekcijski korak u kojem se vrši dodatna usporedba tekstura predložaka te prošireni informacijski filter za poboljšanje estimacije parametara modela.

Nadalje, Visage uvodi par implementacijskih detalja s ciljem poboljšavanja performansi cjelokupnog sustava. Jezgra sustava u kojemu su implementirani algoritmi detekcije i praćenja pisana je u C++ koristeći standardne biblioteke što rezultira više-platformskim kodom. Nadalje, dosta pažnje posvetilo se ulaznim i izlaznim jedinicama na različitim platformama zbog njihovog značajnog utjecaja na detekciju, a time i na doživljaj korisnika.

[3] http://www.visagetechnologies.com/

Jedna od bitnijih značajki je raspodjela posla po različitim dretva i sinkronizacija dretvi. Zbog ograničenih resursa na mobilnim uređajima potrebno je efikasno sinkronizirati tri različite dretve: dretvu za pristup kameri, dretvu za praćenje i dretvu za ispis i ulazno-izlazne operacije. Za iscrtavanje se koriste *OpenGL* i *OpenGL ES* ovisno o platformi, a koristi se i programsko iscrtavanje tamo gdje se pokazalo da je takav pristup isplativiji.

Performanse sustava su ispitane na različitim uređajima i platformama pod različitim uvjetima osvjetljenja te različitim stupnjevima rotacije ulaznog skupa slika. Testovi su pokazali stabilnost algoritma za praćenje do rotacija od 40 stupnjeva po svim osima, a postignute brzine izražene u okvirima po sekundi (FPS) se mogu vidjeti na slici ispod.

–	CPU	Image	Cam
PC	2GHz Intel Core 2 Duo T7200	60	30
iPad	dual-core 1GHz Cortex-A9	30	30
iPhone 4S	dual-core 1GHz Cortex-A9	25	30
iPhone 4	1GHz Cortex-A8	15	20
HTC	1GHz Scorpion	15	12

Slika 3 Test performansi praćenja lica na različitim uređajima u FPS-u

Sadašnji algoritmi i njihove performanse su dostatne za neke aplikacije no kako bi se proširila primjena potrebno je poboljšati kvalitetu i brzinu praćenja. Jedan način za postizanje toga je implementacija nove metode za inicijalnu prilagodbu 3D modela upotrebom proširenog informacijskog filtra te korištenje grafičkih procesora za izračune računalno zahtjevnih operacija s matricama. Drugi način, povezan i sa prvim, uključuje uvođenje novog, kompleksnijeg i boljeg 3D modela.

2. Detekcija i praćenje lica

Detekcija lica je poznat, ali i kompleksan problem. Kompleksnost proizlazi iz činjenice da je ljudsko lice u slici podložno velikom broju varijacija poput varijacija u orijentaciji, položaju, veličini, značajkama te izrazu lica. Svemu tome pridonose i varijacije ne vezane za lice poput onih u osvjetljenju, sjenama te okluziji u sceni.

Zbog mnogih primjena, već implementiranih, a tako i potencijalnih, detekcija lica već duže vrijeme dobiva na značaju i sve je veći interes akademske zajednice i industrije. Problemi ovakve vrste se generalno mogu klasificirati pod područje obrade slike i računalnog vida, a sama detekcija lica pripada potkategoriji detekcije objekata određene klase u slici. Uz lica neke od poznatijih primjena su detekcija prometnih znakova ili pješaka.

Razvoj detekcije lica povlači inovaciju u smislu razvoja novih metoda kojima se pokušavaju riješiti navedeni problemi. Cilj je postići bržu detekciju lica u slici, a pritom zadržati ili povećati pouzdanost te minimizirati utjecaj vanjskih čimbenika scene na detekciju.

Praćenje lica i praćenje značajki lica je srodna problematika, a podrazumijeva detekciju istih u video datoteci iz okvira u okvir. Problemi s kojima se susreće su povezani s onima kod detekcije lica. Oni uključuju promjene uvjeta iz okvira u okvir poput promjene razine osvjetljenja, promjene poze ili rotacije samog lica ili značajke koja se prati na licu te rješavanje problema šuma slike kod određenih kamera ili videa.

2.1. Detekcija lica

Postoji više pristupa detekciji lica od kojih neki postižu bolje rezultate, a neki lošije ovisno o postavljenim uvjetima. Ako se u sceni postave neka ograničenja poput ograničenja na pozadinu slike, ograničenja na slike u boji ili ako se vodi pretpostavkom da će lice u sceni biti u pokretu mogu se implementirati metode koje daju dobre rezultate no takav pristup ima i svoje nedostatke. Ograničenja na samu scenu su očit nedostatak no svaka od navedenih metoda uvodi i dodatne specifične probleme.

Primjerice ako se koristi boja kože za detekciju lica znatno se smanjuje otpornost metode na razlike u osvjetljenju, a mogućnost detekcije lica ljudi različite boje kože postaje upitna.

Ako se detekcija vodi pretpostavkom da će se objekt koji se prati kretati dolazi do problema kada u sceni postoje i drugi objekti koji se kreću. Kombinacijom različitih metoda može se postići određena otpornost na ovakve nedostatke.

Interesantniji i kompleksniji problem je detekcija lica i značajki lica u scenama bez zadanih ograničenja. Zbog već spomenutih varijacija lica ovaj problem se smatra možda i najtežim u detekciji objekata u slici. Problemu se može pristupiti sa različitih strana, a zajedničko svim metodama je da kao ulaz većinom primaju crno-bijelu sliku.

Uzimajući u obzir broj izdanih znanstvenih članaka može se reći da je zanimanje za ovu temu poraslo polovicom 90-ih godina prošlog stoljeća. Među prvim metodama su bile one koje su kao ulazni parameter primale slike crno-bijelu slika lica u frontalnoj ili skoro-frontalnoj pozi.

Među jednostavnijim pristupima su oni koji se temelje na spoznajama o ljudskom licu i pravilima koja iz njih proizlaze. Takve metode se nazivaju *metode zasnovane na znanju*. Karakterizira ih skup pravila koji vrijedi za svako lice poput broja očiju, udaljenosti između očiju ili razlike u kontrastu i intenzitetu između pojedinih dijelova ljudskog lica [3]. Nažalost, takve metode su se pojedinačno pokazale neadekvatne zbog samog procesa konverzije znanja o ljudskom licu u sveobuhvatan set pravila. Zbog toga se postiže loša detekcija kod neuobičajenih lica ili kod slika s kompleksnom pozadinom [4].

Neki od ranijih složenijih pristupa detekciji lica koriste učenje i umjetnu inteligenciju poput neuronskih mreža [5] ili kombinaciju istoga sa statističkim podacima i zasnivaju se na usporedbi ulaznih slika sa već definiranim predlošcima. Predlošci mogu biti unaprijed definirani tako da opisuju lice kao funkciju čime se pokušava dobiti standardni predložak s kojim će se detektirati svako lice i takve metode se nazivaju *metode zasnovane na predlošku* [6][7]. Alternativa zadanim predlošcima su predlošci dobiveni procesom treniranja. Takve metode se nazivaju *metode zasnovane na izgledu* i kod njih, uz samu detekciju, veliku važnost ima trening. Svrha treninga je dobiti dovoljno dobru funkciju koja može klasificirati objekte u dvije skupine: lica i ne-lica, a to se najčešće izvodi tako da se kao ulazni parametri daje skup slika koji predstavlja lica te skup slika koji predstavlja ne-lica. Izazov je napraviti kvalitetan skup slika, takav da je dobiveni predložak dovoljno precizan u detekciji te efikasno implementirati proces treniranja kako bi se postigla dovoljno velika brzina.

Paul Viola i Michael Jones [8] 2001 razvili su metodu detekcije koja je zbog vrlo dobrih rezultata ušla u široku uporabu i povećala zanimanje za ovu granu obrade slike i računalnog vida. Metoda se temelji na algoritmu treniranja zasnovanom na *AdaBoost*, novom načinu reprezentacije slike u memoriji koristeći integralnu sliku te hijerarhijskom skupu klasifikatora koji koriste *Haar*-ove značajke. Implementacija metode se nalazi unutar *OpenCV* biblioteke.

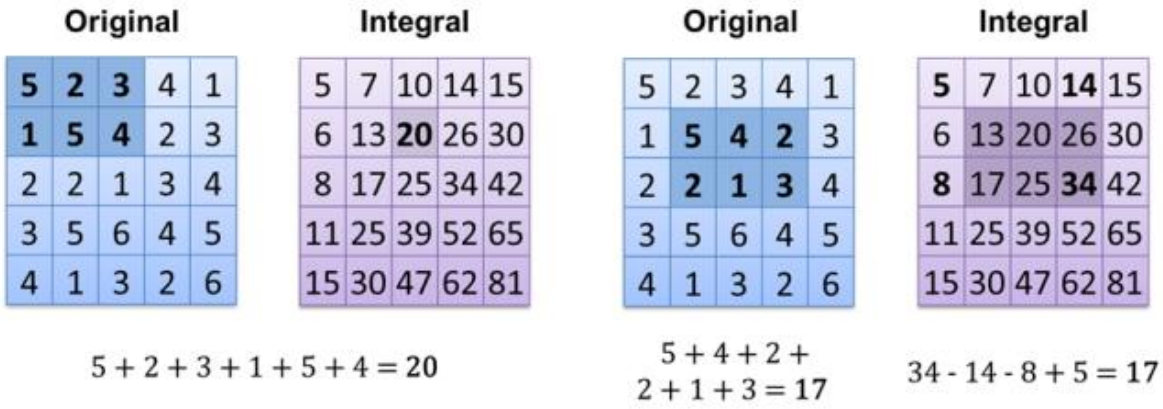
Razvoj metoda za detekciju nakon revolucionarne metode Viole i Jonesa najviše se nastavio prema metodama temeljenim na izgledu i učenju [9][10]. Primjerice 2005 Ambramson i Steux [11] nastavljaju razvoj pokušavajući poboljšati detektor Viole i Jonesa uvodeći novu vrstu značajki čime se eliminira priprema integralne slike i postiže značajno ubrzanje pogotovo za detekciju u realnom vremenu. Na taj rad se nadovezuju Huang, Ai, Li and Lao [12] s detekcijom visokih performansi neovisnoj o rotaciji.

Ako se govori o potkategoriji detekcije lica, detekciji značajki lica i poravnanju lica, među najnovijima je perspektivna metoda Microsoftovog azijskog razvojnog tima [13] imena Eksplicitna regresija oblika (eng. *Explicit Shape Regression*) iz 2012 godine. Karakteristika metode je brži proces treniranja (20 minuta za skup od 2000 slika) iz kojega proizlazi vektorska regresijska funkcija koja se direktno mapira na izgled slike.

2.1.1. Metoda 1 - Viola−Jones Object Detection

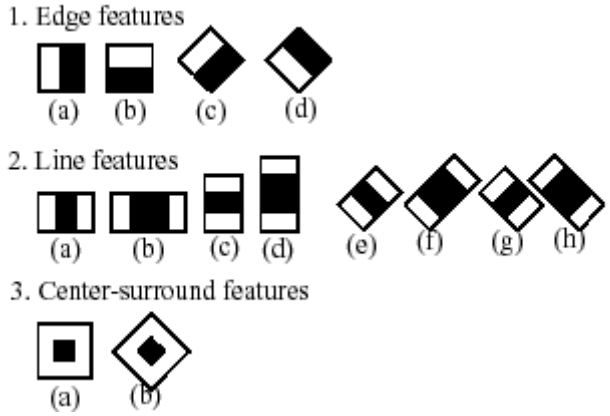
Prošlo je 12 godina od kada je metoda razvijena no i danas je jedna od najčešće korištenih metoda u detekciji lica. To proizlazi iz činjenice da je njena implementacija integrirana u besplatnu biblioteku za računalni vid – *OpenCV* u obliku crne kutije što je čini vrlo pristupačnom korisniku kojemu su potrebne metode detekcije lice no ne želi ulaziti u detalje implementacije. Uz sam algoritam korisnik ima pristup i općenitim kaskadama za detekciju lica te značajki lica, dovoljno dobrim da se izbjegava cijeli proces stvaranja skupa podataka za treniranje te sami proces treniranja. Zbog načina na koji je algoritam osmišljen postiže se precizna detekcija lica s malim brojem lažnih detekcija, a ujedno se i postiže i brzina dostatna za detekciju u realnom vremenu što ga izdvaja od prijašnjih metoda.

Algoritam Viole – Jonesa može detektirati na crno-bijelim ulaznim slikama i zasniva se na 3 temeljna obilježja. Prvo je integralna reprezentacija slika (eng. *integral image)* koja značajno pridonosi brzini algoritama.



Slika 4 Reprezentacija slike u integralnom obliku[4]

Naime, algoritmi detekcije lica često koriste „prozor“ koji se pomiče po slici i unutar kojega se dio slike obrađuje. U ovom algoritmu ta se obrada sastoji od računanja značajki nalik na Haarove bazne funkcije u mnogo različitih veličina. Integralnu reprezentaciju slike je moguće izračunati iz slike koristeći mali broj operacija po pixelu. Potrebno je jednom izvršiti konverziju slike i nakon toga je moguće svaku Haarovu značajku, neovisno o veličini i lokaciji, izračunati u konstantom vremenu - O(*1*).

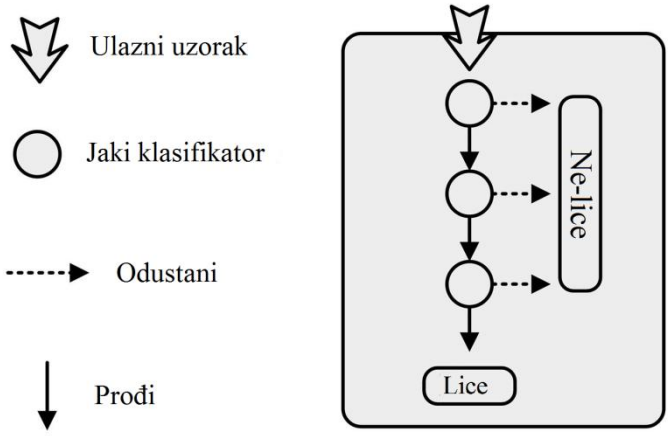


Slika 5 Haarove značajke u *OpenCV*-u[5]

[4] Slika preuzeta sa http://www.codeproject.com/Articles/441226/Haar-feature-Object-Detection-in-Csharp

Drugo obilježje je konstrukcija slabih klasifikatora (eng. *weak classifiers*) koristeći *Adaboost*. Temelji se na načinu izbora malog skupa kritičnih značajki iz velikog skupa značajki koje se nalazu unutar jednog potprozora. Pojednostavljeno, algoritam trenira klasifikatore slabih performansi tj. slabe klasifikatore te onda njihovom kombinacijom kreira jači klasifikator sa puno boljim performansama. Klasifikatori se dodaju na taj način da svaki sljedeći slabi klasifikator koji se dodaje pokušava poboljšati pogreške onog prijašnjeg. Svaki klasifikator asocira se s jednom od Haarovih značajki.

Svrha trećeg obilježja algoritma je također poboljšati brzinu detekcije, a to se postiže grupiranjem klasifikatora u kaskadnu strukturu čime se obrada koncentrira samo na one regije slike gdje je veća vjerojatnost da će se detektirati lice. Ostale regije, poput pozadine, se zbog takve kaskadne strukture vrlo brzo odbacuju i na njih se ne gubi procesorska moć.



Slika 6 Klasifikatori uređeni u kaskade[6]

Postupak detekcije započinje konverzijom ulaznog uzorka tj. slike u integralnu sliku. Ovaj postupak je potrebno izvršiti samo jednom, a brži je od standardnog postupka kreiranja piramide slika različitih veličina. Razlog tome je set značajki koje se specifične u smislu da se svaka može izračunati u bilo kojoj veličini i lokaciji. Nakon toga se postavlja „prozor“

[5] Slika preuzeta sa http://opencv.willowgarage.com/documentation/object_detection.html

[6] Slika preuzeta i prevedena iz [9]

za skeniranje koji se pomiče po slici za neki pomak piksela. Pomak piksela tj. Δ piksela je varijabilan te o njemu ovisi brzina samog detektora, a obrnuto-proporcionalno tome i stopa detekcije i broj lažno detektiranih objekata. Pomak također ovisi i o veličini „prozora“ za skeniranje. Naime, zbog već spomenutih karakteristika Haarovih značajki, nije potrebno skalirati sliku već se skalira sam detektor mijenjajući veličinu „prozora“ za skeniranje. Unutar svakog potprozora vrši se kalkulacija značajki svakog sloja kaskada. Same kaskade su definirane procesom treninga i sadržane su u datoteci koja se predaje algoritmu. Prvi sloj kaskada je najopćenitiji, koristi najjednostavnije značajke te je najbrži. Ima 100-postotnu detekciju lica u slici, ali zbog svojih obilježja, stopa lažno detektiranih lica je 40%. Drugi sloj je složeniji, ima veći broj značajki te je vremenski zahtjevniji, ali kod njega je stopa lažno detektiranih slika 20%. Svaki sljedeći sloj sve je složeniji. Detekcija se vrši tako da se u svakom potprozoru prolazi redom kroz slojeve, kojih ukupno može biti do 32, te nakon što svaki sloj vrati pozitivan rezultat, može se reći da je lice detektirano. Ako neki sloj vrati negativan rezultat postupak se prekida, daljnji slojevi se ne obrađuju te se sam postupak značajno ubrzava. Time se efektivno fokusira na potprozore koji „obećavaju“, a izbjegava se procesiranje onih u kojima nije prisutno lice.

Zbog toga što se „prozor“ za skeniranje pomiče za mali Δ piksela (1 piksel) dolazi do većeg broja detekcija za isto lice ili za lažno detektirana lica i moguće je dodatno optimizirati algoritam. Optimizacija se provodi kroz dodatnu obradu seta „prozora“, u kojima je dojavljena pozitivna detekcija. Vrši se proračun nad svim granicama regija, a kao rezultat proizađe finalna regija čije su granice prosjek onih u setu. Time se također i smanjuje broj lažno detektiranih lica.

Proces treninga tj. način na koji se odabiru najbolje značajke iz seta od 45,396 značajki te koje od tih značajki su sadržane u kojem sloju kaskade je nešto kompliciraniji od samog algoritma detekcije te neće biti pobliže objašnjen. Koristi se varijanta *AdaBoost* algoritma za učenje te je moguće podešavati parametre poput pragova, broja značajki i dr.. Više o samom algoritmu treninga se može naći u radu Viole−Jones [8].

Nije nužno da ovaj postupak daje najbolji postotak detekcije lica u odnosu na druge algoritme, ali ono što ga je učinilo jednim od poznatijih i najkorištenijih algoritama je njegov visok postotak detekcije lica uz postizanje visoke brzine detekcije, svojevremeno 15 puta brže od bilo kojeg pristupa do tada [8].

Zbog potrebe za brzom detekcijom u realnom vremenu implementacija algoritma se koristi i unutar Visage|SDK kroz *OpenCV* biblioteku.

2.1.2. Metoda 2 - YEF[7] Real-Time Object Detection

Varijanta algoritma za detekciju lica koja je implementirana unutar CCV biblioteke za kompjuterski vid inspirirana je i temelji se djelomično na ovom algoritmu. CCV biblioteka je relativno nova, njen razvoj započeo je 2010 godine, a interesantna je zato što nudi implementaciju algoritma za detekciju pisanog u JavaScriptu te mogućnost vrlo jednostavne integracije.

YEF algoritam je razvijen tako da se nadovezuje na algoritam Viole−Jones i različitim preinakama ga se pokušava poboljšati. Poboljšanje se temelji na novoj vrsti značajki različitoj od Haarovih značajki. Uz to su uvedene i promjene u pripremi slike tj. umjesto pripreme integralne slike i normalizacije svakog potprozora, izračun se temelji na vrijednostima individualnih piksela.

Haarove značajke iz Viola−Jones algoritma baziraju se na kalkulaciji razlike suma piksela kvadratičnih regija i potrebi da se prije izračuna vrši normalizacija potprozora. Transformacija u integralnu sliku u velikoj mjeri rješava te probleme no takvi izračuni su i dalje vremenski zahtjevni te zauzimaju resurse. U YEF algoritmu se koriste drugačije značajke koje se temelje na statističkoj analizi individualnih vrijednosti piksela tj. rezultat se dobiva usporedbom između piksela (veće, manje) umjesto izračuna određenog praga. Iz tog razloga nije nužno vršiti normalizaciju slike.

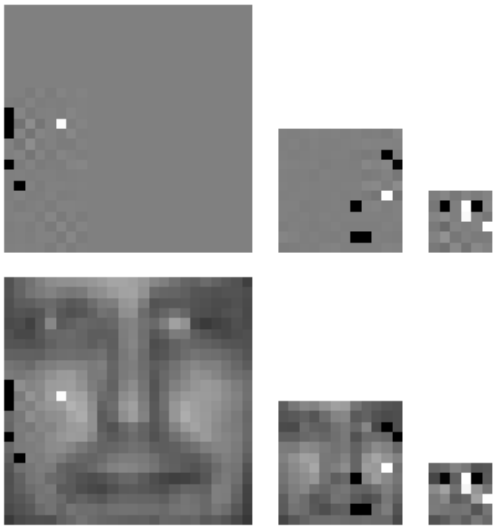
Jedna značajka se sastoji od tzv. kontrolnih točaka (eng. *Control-Points*). Ako je širina slike ili potprozora *W,* a duljina *H*, onda je kontrolna točka definirana kao dio slike oblika <i,j> gdje je $0 \leq i < H$ i $0 \leq j < W$, a uzimajući lokaciju na slici z, vrijednost piksela na toj lokaciji je *val(z)*. Jedna značajka definirana je u obliku dva seta kontrolnih točaka:

$$x_1, \dots, x_n \; ; \; y_1, \dots, y_m \; ; n, m \leq K$$

K je gornja granica koja je varijabilna, a njena vrijednost direktno utječe na performanse detektora. Svaka značajka radi na slici originalnih dimenzija, a ostale dimenzije (1/2,1/4,...) se moraju pripremiti unaprijed skaliranjem originalne slike. Značajka odgovara potvrdno ako vrijedi:

$$\forall x \in \; x_1, \dots, x_n \; ; \forall y \in \; y_1, \dots, y_m \; ; val(x) > val(y)$$

[7] Yet Even Faster



Slika 7 Značajke koje se koriste u YEF algoritmu[8]

Implementaciji algoritma detekcije lice može se pristupiti na dva načina, a to su gradeći piramidu slika ili ne gradeći piramidu sliku. Piramida slika obično se gradi smanjujući rezoluciju slika za 25% sve dok se ne dostigne veličina „prozora“ za skeniranje. Nakon tog koraka provodi se sličan postupak kao i kod Viole−Jones. Za pomak Δ se pomiče prozor za skeniranje po slikama različitih veličina i na kraju svake faze se ujedinjuju slične detekcije.

Ako se ne gradi piramida slika (vodeći se postupcima Viole−Jones) onda se algoritam malo komplicira. Naime, kod Viole−Jones izbjegava se skaliranje slike skalirajući Haarove značajke, što je zbog njihove kvadratičaste prirode moguće iako nije ekvivalentno dok skaliranje kontrolnih točaka nije uopće ekvivalentno jer one predstavljaju sam piksel i zapravo se skalira udaljenost između piksela. Tome problemu se pristupa tako da se ponavlja proces treninga za različite rezolucije slike čime se dobiva više različitih kaskada. Slijedi izgradnja znatno jednostavnije piramide, gdje je svaki sloj duplo manje rezolucije od prijašnjeg. Nakon konstrukcije slijedi proces pomicanja „prozora“ za skeniranje za svaki od različitih detektora, a rezultat je unija svih rezultata iz različitih detektora za svaki nivo piramide.

[8] Slika preuzeta iz [11]

Trening kaskada koristi varijantu *AdaBoost* algoritma te implementira svojevrsni genetski algoritam. Prva generacija sadrži 100 slučajno odabranih značajki. Svaka sljedeće generacija se sastoji od 10 najboljih iz prijašnje (onih s najmanjom pogreškom), 50 kojih rezultiraju mutacijom najboljih 10 te 40 novih slučajno odabranih. Proces treninga slijedi pravila koje su uveli Viola−Jones [8][11]. Vrijednost K varijable je eksperimentalno odabrana proučavajući broj operacija u odnosu na broj lažno detektiranih lica i iznosi 6.

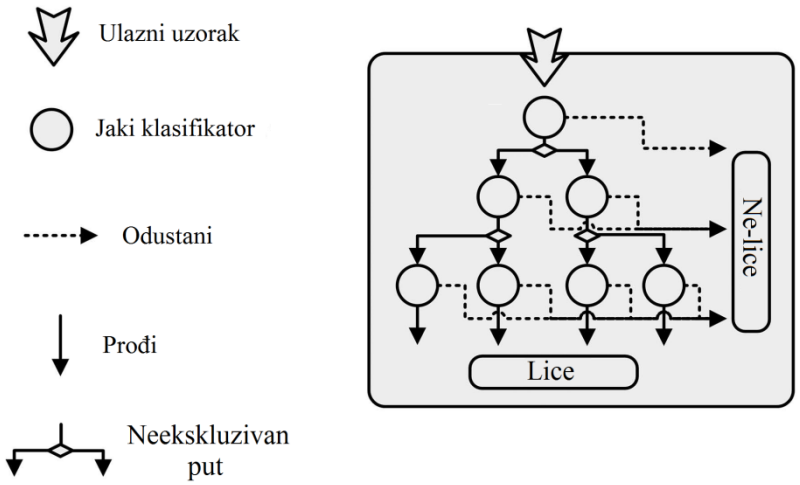
Implementacija u CCV biblioteci koristi pristup izgradnje piramide slika. Izgradnja piramide je ujedno i najzahtjevniji dio algoritma no dobra vijest je što JavaScript implementacija pruža način za rješavanje tog problema preko HTML5 *canvas* elementa.

2.1.3. Metoda 3 - Rotation Invariant Multiview Face Detection

Ova metoda donosi novitete u detekciji lica pronalaženjem novih načina detekcije lica s rotacijom u različitim ravninama, a nadograđuje se na Viola−Jones i Abramson−Steux YEF algoritme. Ideje metode su, uz YEF algoritam, implementirane unutar CCV biblioteke za računalni vid.

Problem koji se pokušava riješiti je problem detekcije lica s rotacijom što je u realnim sustavim vrlo česta pojava. Lica gledana iz profila sadrže manje informacija nego ona frontalna, a utjecaj vanjskih čimbenika poput osvjetljenja i okluzije imaju na njih veći učinak. Uz to, kombinacija rotacija različitih ravnina znatno pridonosi broju varijacija za koje je potrebno istrenirati detektor. Problemu se pristupa sa tri razine i svaka će biti ukratko opisana.

Prva razina s koje se pristupa problemu je razina strukture detektora. Primjerice kod Viola−Jones ili YEF to je kaskadna struktura, dok se ovdje koristi struktura WFS[9] stabla kako bi se konsolidirala dva zadatka: detekcija lica i procjena poze lica.

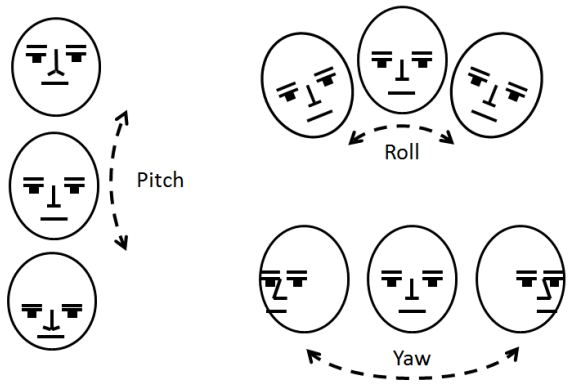


Slika 8 Klasifikatori uređeni u WFS stablo[10]

[9] WFS – pretraživanje po širini – eng. *Width-First-Search*

[10] Slika preuzeta i uređena iz [12]

Grade se četiri različita detektora trenirana na licima rotiranim oko različitih osi (eng. *Roll, Yaw*). Lica svakog detektora se dalje kategoriziraju i organiziraju u obliku WFS stabla. Valja primijetiti da se iz jednog čvora stabla može doći do više čvorova/listova (Slika 5). Ta neekskluzivnost je jedan od razloga zašto je WFS stablo fleksibilno što rezultira odlukama koji nisu niti preagresivne, a ni preneodređene.

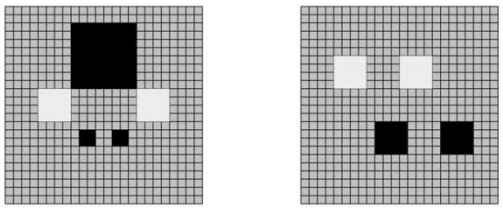


Slika 9 Prikaz nagiba glave po različitim osima[11]

Druga razina s koje se pristupa je kompleksna, a uključuje uvođenje novog algoritma za odabir kritičnih značajki koji može raditi sa WFS stablom. Ime algoritma je „Algoritam vektorskog ubrzanja“ tj. *Vector Boosting Algorithm* za razliku od *AdaBoost* algoritma koji se koristi u ranije navedenim algoritmima detekcije. Više o algoritmu se može naći u radu Huang et al [12].

Raštrkane granularne značajke (eng. *Sparse Granular features*) su treća razina pristupa detekciji lica s rotacijama. Viola−Jones koriste Haarove značajke koje zbog svoje strukture nisu u mogućnosti opisati kompleksne uzorke profilnih lica. Abramson−Steux rješavaju taj problem sa značajkama baziranim na pikselima, ali ni one se nisu pokazale dovoljno ekspresivne. Zbog toga je razvijena nova vrsta značajki zajedno sa algoritmom za odabir dovoljno dobrih koji koristi heurističke metode pretrage.

[11] Slika preuzeta sa http://msdn.microsoft.com/en-us/library/jj130970.aspx



Slika 10 Primjer raštrkanih granularnih značajki[12]

Kao rezultat proizlazi jedan detektor, sačinjen od 4 različita, s mogućnošću detekcije lica sa značajnim nagibom. Kod procesa detekcije velik dio računalnog vremena zauzima faza pripreme tj. skaliranje slike kako bi se omogućila detekcija lica različitih veličina te priprema i izračun integralne slike no sam izračun značajki to nadoknađuje jer je, pogotovo kod većeg broja klasifikatora, efikasniji.

[12] Slika preuzeta iz [12]

2.2. Praćenje lica

Postupak praćenja lica najčešće se izvršava nakon što postupak detekcije lica vrati pozitivan rezultat. Zatim se koriste razne metode kako bi se odredio položaj lica u svakom sljedećem okviru video datoteke ili toka podataka iz kamere. Pokušava se izbjeći ponovna detekcija lica u svakom sljedećem okviru zbog toga što je metoda detekcije računalno i vremenski zahtjevna.

Praćenje lica je kritičan dio sustava poput sustava za prepoznavanje lica, sustava za video nadzor ili za usporedbu lica i značajki lica. Koristi se u velikoj mjeri kod postupaka 3D modeliranja u različitim industrijama od kojih su najznačajnija industrija video igara i filmska industrija.

U ovisnosti o namjeni praćenje se može usmjeriti prema praćenju lica ili primjerice praćenju značajki lica. Također se razlikuju u vrsti informacija koje proizlaze iz metode praćenja poput 2D položaja lica iz okvira u okvir, 3D pozicije lice ili lokacije značajki na licu.

Problemi koji se pojavljuju su ili isti ili povezani s onima koji se pojavljuju kod metoda detekcije lica. Potrebno je riješiti ili minimizirati utjecaj okluzije, osvjetljenja te naglih promjena u značajkama lica ili poze lice.

Uz detekciju lica, praćenje lica uključuje i prilagodbu lica (eng. *face alignment*) lociranjem semantičkih značajki poput očiju, nosa, usta i brade. Jedan od načina je minimizacija pogreške poravnanja (eng. *alignment error*), a o ovisnosti o tome kako se problemu pristupa metode se dijele na metode koje se temelje na optimizaciji i metode koje se temelje na regresiji. Primjer je metoda imena Eksplicitna regresija oblika (eng. *Explicit Shape Regression*) [13] u kojoj se pogreška minimizira uz pomoć regresijske funkcije. Metoda se razlikuje od ostalih jer iskorištava korelaciju između značajki lica i time se postiže otpornost na djelomičnu okluziju te varijacije u pozi. Poput drugih metoda i ovdje se uvodi ograničenje na oblik, ali ono nije realizirano parametriziranim modelom već kroz metode treninga koje uvode fleksibilnost. Finalni oblik je linearna kombinacija oblika iz procesa treninga.

Unutar Visage programskog razvojnog okruženja praćenje koristi metode prilagodbe parametriziranog 3D modela u kombinaciji sa informacijskim filterom za bolju procjenu parametara modela. Nakon detekcije lica odabiru se karakteristične značajke tj. uzorci

(uzorci moraju biti strukturirani – rubovi ili specifične teksture) koji se prate iz okvira u okvir predstavljajući estimacijski problem koji se rješava normaliziranom križnom korelacijom. Nakon što prošireni informacijski filtar obradi te informacije dobiju se podaci poput rotacije i translacije te položaja značajki lica. Dobivenim informacijama dobavljaju se dijelovi slike iz prethodno okvira koji se dalje koriste za pretragu u trenutnom okviru. Broj značajki u teoriji poboljšava kvalitetu praćenja no isto tako povećava i kompleksnost izračuna.

3. Metodologija

3.1. Opis i definicija problematike

Zadatak je postići dovoljno brzu te dovoljno preciznu detekciju lica odnosno značajki lica te pratiti lice i značajke iz okvira u okvir. Za potrebe rada opseg je postavljen na detekciju i praćenje iz kamere.

Razmatrana su dva pristupa problemu. Prvi je ručno prebacivanje tj. prevođenje algoritma za praćenje iz jezika C/C++ u JavaScript i HTML5 uz korištenje već postojeće implementacije algoritma za detekciju iz biblioteke za računalni vid CCV (opis algoritma se može pronaći u poglavlju 2.1.2 i 2.1.3). Tada bi bilo potrebno proučiti algoritam i JavaScript kod detektora te pronaći mjesto u kodu gdje se može ubaciti daljnja obrada na temelju dobivene detekcije.

Drugi pristup podrazumijeva upotrebu prevoditelja (poput alata Emscripten 1.2.1) kako bi se postojeći kod u C/C++ preveo u JavaScript automatski. U tom slučaju potrebno je proučiti dokumentaciju prevoditelja i primjere te postaviti putove da različitih alata kako bi prevoditelj ispravno funkcionirao. Također je potrebno, u znatno manjoj količini, modificirati postojeći kod ako u njemu postoje dijelovi koji se ne mogu prevesti.

Oba pristupa imaju svoje mane i prednosti, a odabir jednoga ovisi o zadanom cilju i svrsi. Prvi pristup je vrlo vremenski zahtjevan i uz to zahtjeva prilično dobro poznavanje jezika u koji se prevodi, a isto tako i jezika iz kojega se prevodi. Ako su ispunjeni ovi uvjeti kod koji proizlazi iz procesa je optimiziran i programer ima potpunu kontrolu nad svim aspektima procesa konverzije, a cijeli projekt se može promatrati kao dugoročan projekt.

Drugi pristup je u odnosu na prvi manje vremenski zahtjevan i rezultati su puno brže vidljivi. Zbog toga što je konverzija automatska programer ne mora znati jezik u koji prevodi u detalje, ali potrebno je znati barem paradigme jezika. Većina vremena se ulaže u rješavanje specifičnih problema koji se pojavljuju u procesu konverzije poput problema uzrokovanih različitim paradigmama i razlikom u semantici između dva jezika te potencijalno problema s lošim postavkama alata i slično. Kod koji proizlazi iz ovakvog procesa u pravilu je manje optimiziran te programer nema potpunu kontrolu. Dodatno je

potrebno proučiti sučelja za komunikaciju između prevedenog koda, koji je ne rijetko enkapsuliran, i dodatnog koda koji se nadovezuje na prevedeni.

Budući da diplomski rad više odgovara formatu kraćeg projekta odabran je drugi pristup te se za prevođenje postojećih biblioteka iz C/C++ koristio prevoditelj Emscripten.

3.2. Opis postupka

Rad se može podijeliti u 3 osnovne faze. Početna faza pripreme podataka i treniranja kaskada, druga faza u kojoj se pomoću Emscriptena prevodi algoritam za praćenje (metoda regresije oblika) te se povezuje sa algoritmom za detekciju te finalna faza koja uključuje: prevođenje *OpenCV* biblioteke (algoritam detekcije), prevođenje biblioteka Visage programskog razvojnog okruženja (algoritam praćenja), prevođenje jednostavnog pokaznog projekta koji koristi navedene biblioteke te povezivanje tog koda sa ručno pisanim JavaScript kodom (dohvaćanje slike iz kamere).

1) **Faza I – Priprema podatka i treniranje**

Cilj prve faze jest dokučiti da li je moguće istrenirati kaskade značajki za algoritam detekcije implementiran u CCV biblioteci. Korištena je baza podataka slika ljudskih lica različitih rotacija, sakupljenih uglavnom s Weba. Baza također sadrži anotirane vrijednosti za svaku sliku poput vrijednosti položaja značajki lica (oči, brada, uši), dobi, spola, i drugog.

path_na...	Age	Gender	Race	Eye_open	Mouth_...	Gaze_di...	Gaze_di...	Face_Bound_Box_Position_value_x	Face_Bound_Box_Position_value_y	Face_Bound_Box_Dimension_value_x
\SLICKE...	25	Female	Caucasian	0	0	0	0	4	30	58
\face1.jpg	28	Female	NULL	75	50	0	0	5	21	92
\m17v1e...	26	Male	NULL	77	0	0	0	5	16	87
\face_su...	28	Female	NULL	73	27	0	0	6	25	85

Face_Bound_Box_Dimension_value_y	Head_Rotation_value_x	Head_Rotation_value_y	Head_Rotation_value_z	Expression_Anger	Expression_Digust	Expression_Fear	Expression_Happy
57	3	-2	0	0	0	0	30
69	0	0	0	0	0	0	50
70	0	0	-3	0	0	0	33
67	0	0	3	0	0	0	24

p9_1_y	p9_2_x	p9_2_y	p9_3_x	p9_3_y	p9_15_x	p9_15_y	p10_1_x	p10_1_y	p10_2_x	p10_2_y	p10_5_x	p10_5_y	p10_6_x	p10_6_y	p10_7_x	p10_7_y	p10_8_x
0.733427	0.27379	0.748942	0.355593	0.74048	0.357262	0.770099	0.617696	0.496474	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
0.65125	0.384749	0.65	0.495514	0.625953	0.493934	0.6725	0.956672	0.49875	0.0658579	0.4975	0.883882	0.7225	0.124783	0.73375	0.87127	0.670818	0.129983
0.60125	0.429936	0.59875	0.514815	0.577955	0.520701	0.6275	0.903238	0.451808	0.0780255	0.42875	0.867834	0.645	0.0955414	0.6275	0.829618	0.6475	0.157643
0.661459	0.425698	0.669291	0.512898	0.6725	0.515933	0.71625	0.860395	0.42	0.0971168	0.42625	0.831563	0.59625	0.150228	0.61125	0.816388	0.61125	0.168437

Slika 11 Anotirane vrijednosti za slike u bazi

Napisana je skripta za filtriranje u Pythonu kako bi se dobile samo podobne slike. Pod podobne slike spadaju one gdje je kut rotacije glave u bilo kojoj osi veći od -5 i manji od 5 i gdje su ispravno definirani parametri pravokutnika koji opisuje lice. Dobivene

slike se dalje obrađuju izrezujući lica te promjenom veličine na 24x24 piksela. Rezultat je set od 3466 slika. Te slike uz set negativnih slika, također preuzetih s Weba, predaju se kao ulaz u alat za treniranje u sklopu CCV biblioteke.

Proces treniranja ima varijabilno trajanje u ovisnosti o željenom broju razina kaskada. Na računalu konfiguracije: Intel(R) Core(TM) i7-2600 i 16 GB RAM proces treninga je trajao 1 dan iz čega je proizašla datoteka kaskada od 20 razina. Zamijećeno je da je potrebno puno više vremena za pripremu negativnih primjera u kompleksnijim kaskadama s više značajki.

2) Faza II – prevođenje metode regresije oblika

Nakon što je uspješno izvršen proces treniranja te se time dokazalo da je moguće proizvesti vlastitu datoteku kaskada i koristiti ju dalje s CCV algoritmom detekcije lica potrebno je prevesti algoritam praćenja lica u JavaScript.

Prvi i osnovni korak je postavljanje Emscripten okruženja na operacijskom sustavu koji se koristi za što postoji vodič na njihovoj stranici. Prevođenje primjera koji dolaze s Emscripten projektom služi kao provjera da su okruženje i sve pripadne sistemske varijable dobro postavljene za trenutni operacijski sustav.

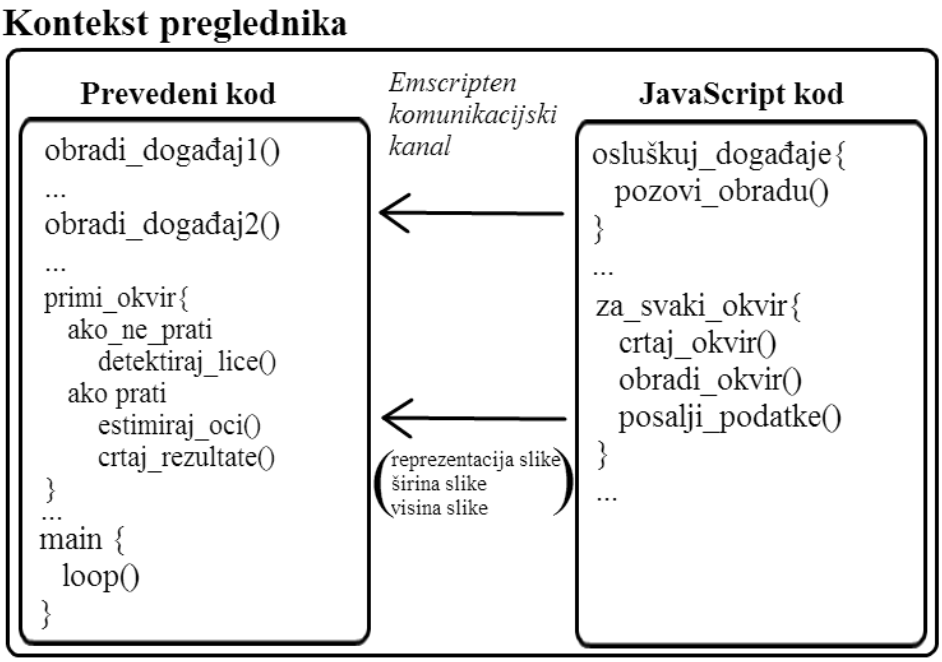
U fazi dva nije korištena biblioteka *OpenCV* nego se umjesto toga sučelje za pristup kameri, za manipulaciju slike te detekciju implementira u zasebnoj JavaScript datoteci. Algoritam praćenja se zatim prevodi u JavaScript i povezuje se sa ručno napisanim sučeljem. Prevedeni i ručno pisanog kod povezuje se preko mehanizama koje implementira Emscripten.

Pristupanje kameri preko sučelja operacijskog sustava razlikuje se od onoga u Web pregledniku. Unutar Web preglednika korisnik prethodno treba dati dopuštenje za korištenje kamere. Nakon toga se na svaku promjenu sadržaja okvira aktivira metoda „*drawToCanvas“* (koristi se *Request Animation Frame* mehanizam). Unutar metode poziva se algoritam za detekciju iz CCV biblioteke i nastavlja se pozivati sve dok se ne detektira lice. Ako je lice detektirano, detekcija se više ne poziva, već se preko mehanizama komunikacije poziva prevedeni algoritam praćenja sa sljedećim argumentima: lista vrijednosti piksela slike, širina slike, visina slike i koordinate pravokutnika koji opisuje lice. Vrijednosti piksela slike se prije slanja obrađuju tako da se iz slike u boji dobije crno-bijela slika.

Crtanje slike na HTML5 *canvas* element izvršava se unutar „*drawToCanvas"* metode, a pozivima iz prevedenog koda crtaju se dodatni elemenata na sliku (točke zjenice). Uz osnovnu funkcionalnost implementirano je i osnovno korisničko sučelje koje se sastoji od mogućnosti unosa sa tipkovnice te pomičnih kliznika (eng. *slider*) preko kojih se mogu podešavati neke vrijednosti (razmak između očiju, veličina i dr.).

Primjerice:

- „t" - reset praćenja i ponovna detekcije
- „d" - uključi/isključi detaljno iscrtavanje
- „l" - uključi/isključi računa za lijevo oko



Slika 12 Dijagram komunikacije između prevedenog koda i ručno pisanog JavaScript koda

3) Faza III – prevođenje dijelova Visage|SDK

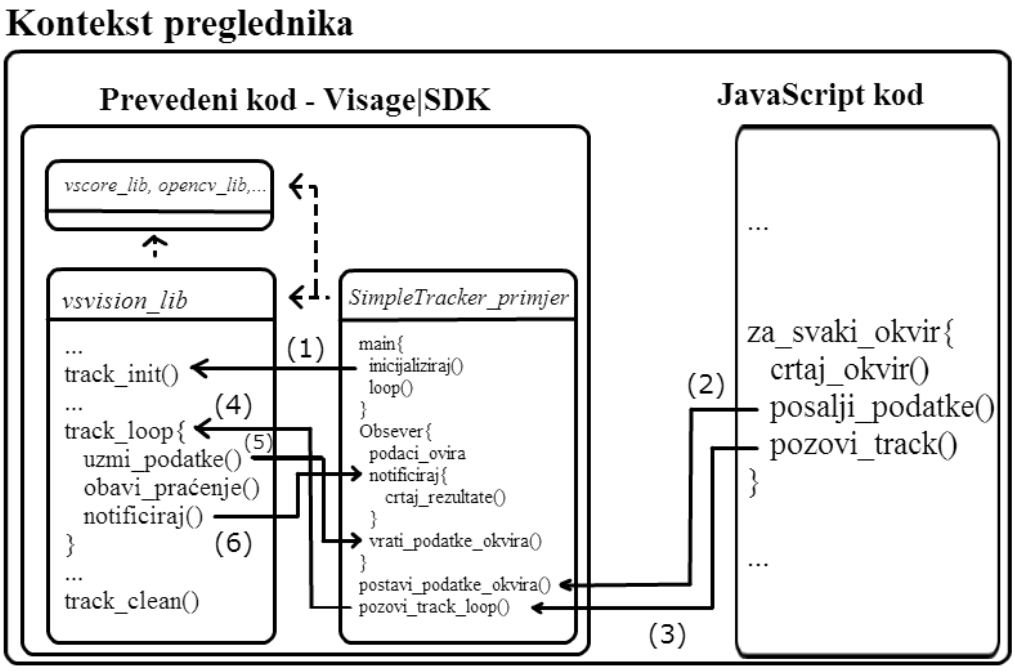
Cilj posljednje faze je prevođenje osnovnih dijelova Visage sustava za praćenje zajedno sa svim pripadnim bibliotekama (*OpenCV*).

Najprije je potrebno prevesti *OpenCV* biblioteku koristeći Emscripten. To se može postići modificiranjem *make* skripti tako da se umjesto uobičajenog prevoditelja koristi

Emscripten. Uspješnost prevođenja testirala se integracijom prevedene *OpenCV* biblioteke s projektom iz faze II. Rezultat je detekcija Viola−Jones algoritmom i praćenje metodom regresije oblika unutar Web preglednika.

Sljedeći korak je prevođenje Visage biblioteka *vscore* i *vsvision*. U ovom koraku bilo je potrebno modificirati izvorni kod i izbaciti neke stvari djelomično zbog načina na koji Emscripten rješava beskonačne petlje (eng. *game loops*), a djelomično zato što neke dijelove koda jednostavno nije bilo moguće uspješno prevesti.

Visage sustav za praćenje, točnije, sustav za dojavu rezultata implementiran je u arhitekturi subjekt-promatrač zbog čega je način na koji prevedeni kod te ručno pisani JavaScript kod komuniciraju različit od onoga u Fazi II.



Slika 13 Dijagram komunikacije, faza III

Petlja za praćenje je podijeljena na 3 dijela po funkcionalnosti. Inicijalizacijski dio, dio za iscrtavanje u kojemu se također vrše i izračuni za praćenje te finalni dio u kojemu se čiste inicijalizirani objekti tj. memorija.

Korisnik započinje proces praćenja, opisan na Slika 13, klikom na gumb u Web pregledniku. Postavlja se inicijalni okvir te se inicijaliziraju varijable (1). Pomoću

zasebne JavaScript datoteke, u kojoj je ručno pisani kod, iscrtava se okvir iz kamere na HTML5 *canvas* element. Za razliku od faze II nema pripreme slike jer je ona već implementirana preko *OpenCV* biblioteke unutar Visage sustava. Umjesto toga poziva se metoda koja postavlja podatke o slici (vrijednosti piksela) u varijablu promatrača (2). Nakon što su podaci o okviru postavljeni poziva se glavna petlja u kojoj se vrši praćenje ili detekcija (u prvom okviru ili u reinicijalizaciji)(3,4). Unutar glavne petlje dohvaćaju se podaci o okviru (5) te se na temelju podataka vrši obrada slike odnosno praćenje značajki lica i glave. Budući da je promatrač pretplaćen na objekt u kojemu je implementirana petlja za praćenje preko tog mehanizma dojavljuju se podaci promatraču i na okvir iz kamere dodatno se iscrtavaju željene značajke (6).

3.2.1. Opis glavnih problema i njihova rješenja

U procesu konverzije često se znaju pojaviti problemi uzrokovani neispravno postavljenim sustavom za prevođenje ili oni uzrokovani razlikom semantika izvornog i ciljnog jezika.

Kod postavljanja okruženja za prevođenje poput Emscriptena treba biti pažljiv da su instalirane prave verzije potrebnih alata. Emscripten je relativno nov prevoditelj, a verzije alata koje se koriste također imaju česte izmjene i nadogradnje. Ne garantira se da će Emscripten funkcionirati s novom verzijom alata isto kao i sa službeno podržanom tako da treba pratiti instrukcije[13] doslovno. Također, postavke ovise o operacijskom sustavu (Windows, OS X, Linux).

1) Problemi u Fazi I

Proces treniranja datoteke kaskada nije trivialan i zahtjeva dobro definiran set ulaznih podataka. Ako to nije slučaj algoritam ne radi dobro i dolazi do beskonačnih petlji. Tijekom procesa treniranja otkrilo se da određene slike nisu dobro anotirane. Takve slike su izolirane te izbačene iz seta pozitivnih slika. Drugi problem je povezan sa negativnim setom slika. Manji dio problema predstavlja pronalaženje takvih slika na Webu te priprema, a veći dio proizlazi iz činjenica da je priprema negativnih primjera u

[13] Stranica na kojoj se mogu naći upte - https://github.com/kripken/emscripten/wiki/Tutorial

većim slojevima kaskada unutar procesa treniranja posao koji zahtjeva puno vremena (1 dan na više). Taj problem nije riješen jer to uključuje modifikaciju algoritma za treniranje što premašuje doseg diplomskog rada.

2) Problemi u Fazi II

Druga faza uključuje prevođenje jednog konkretnijeg projekta pa je tu bila veća mogućnost pojava raznih poteškoća. Kroz postupak prevođenja bolje je upoznat alat Emscripten i način na koji se potrebno nositi s razlikama između izvornog i ciljnog jezika.

Prvi primjer su operacije sa binarnim i tekstualnim datotekama. Iz sigurnosnih razloga nije lako pristupati lokalnim datotekama korisnika, a ovisno o postavkama Web preglednika može se dogoditi da je ta funkcionalnost u potpunosti isključena. Kod algoritma detekcije nužno je imati tu mogućnost, primjerice zbog datoteke kaskada. Emscripten dopušta da se datoteke zapakiraju zajedno sa izvršnom verzijom programa te da se onda koristi virtualni sustav za upravljanje datotekama. Pritom nije potrebno modificirati kod (fopen, fread).

```
--preload-file haarcascade_frontalface_default.xml
```

Specifičnost Emscriptena je da se prevedeni kod enkapsulira u modul. U modulu su implementirane metode za inicijalizaciju varijabli, a isto tako i cijeli konteksta izvornog koda. Jednom kada se pokrene *main* funkcija, obave se sve zadane operacije, *main* funkcija vrati vrijednost i kontekst se „čisti“. Kako bi se spriječilo brisanje konteksta, a tako i svih varijabli i vrijednosti na koje one pokazuju, potrebno je modificirati izvorni kod tako da *main* funkcija nikad ne završi. Ako se to proba postići pomoću *while* petlje dolazi do blokiranja skripte jer se JavaScript izvršava u jednoj dretvi. Umjesto toga koristi se posebni Emscriptenov mehanizam *emscripten_set_main_loop* kojemu se predaje pokazivač na metodu koja će se pokretat u petlji te vrijednost koja označuje broj okvira u sekundi. Uz ovu metodu postoji još i *emscripten_pause_main_loop* tako da programer ima određenu kontrolu.

Zbog pretvorbe koda iz više razine u asemblerski kod pa zatim opet na višu razinu funkcije se preimenuju i gube svoja stara imena. Ako postoji potreba da se prevedene metode pozivaju naknadno tada se treba spriječiti njihovo preimenovanje. Za to također

postoji Emscriptenov mehanizam. Prilikom prevođenja valja navesti za koje metode se želi zadržati njihovo originalno ime.

```
-s EXPORTED_FUNCTIONS="['_main','_callTracking']"
```

3) Problemi u Fazi III

Posljednja faza je najzahtjevnija jer je Visage sustav kompleksan čak i ako se prevedi njegov najosnovniji dio. Dio kompleksnosti Visage sustava jest implementacija u više dretvi. Budući da u JavaScriptu ne postoje dretve u istom obliku u kakvom postoje u C/C++ kod se ne može prevesti bez da se modificira. Izvorni kod je izmijenjen, iz njega su izbačena stvaranja novih dretvi te je metoda u kojoj je implementirana glavna petlja podijeljena u tri nove. Metode se pozivaju odvojeno, a glavni dio se poziva za svaki okvir iz ručno pisanog JavaScripta kao što je opisano na Slika 13. Također je modificiran i način iscrtavanja. Umjesto iscrtavanja pomoću OpenGL-a koriste se pozivi JavaScript metoda za iscrtavanje na *canvas* element preko Emscriptenovog mehanizma *emscripten_run_script*. Mehanizam dozvoljava pokretanje bilo kakvog JavaScript koda direktno iz izvornog koda.

```
char inputString[500];

sprintf(inputString,"canvas.getContext('2d').arc(%f,%f,1,0,2*Math.PI,true);",x,y);

emscripten_run_script(inputString);
```

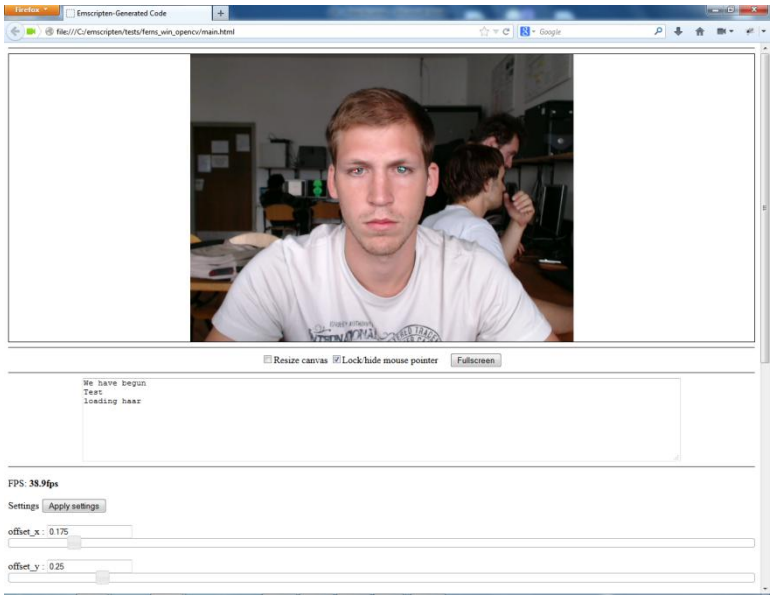
3.3. Rezultati

Rezultat prve faze diplomskog rada jest datoteka kaskada koja je proizašla iz procesa treniranja. Ne može se reći da ima bolje performanse od standardne jer je trenirana na inferiornijem setu slika no postignuta je uspješna detekcija lica unutar okvira što je i bio zadani cilj. Time se dokazalo da je algoritam iskoristiv, a stvaranje kvalitetnije datoteke kaskada bi bio potencijalno zaseban projekt u budućnosti.

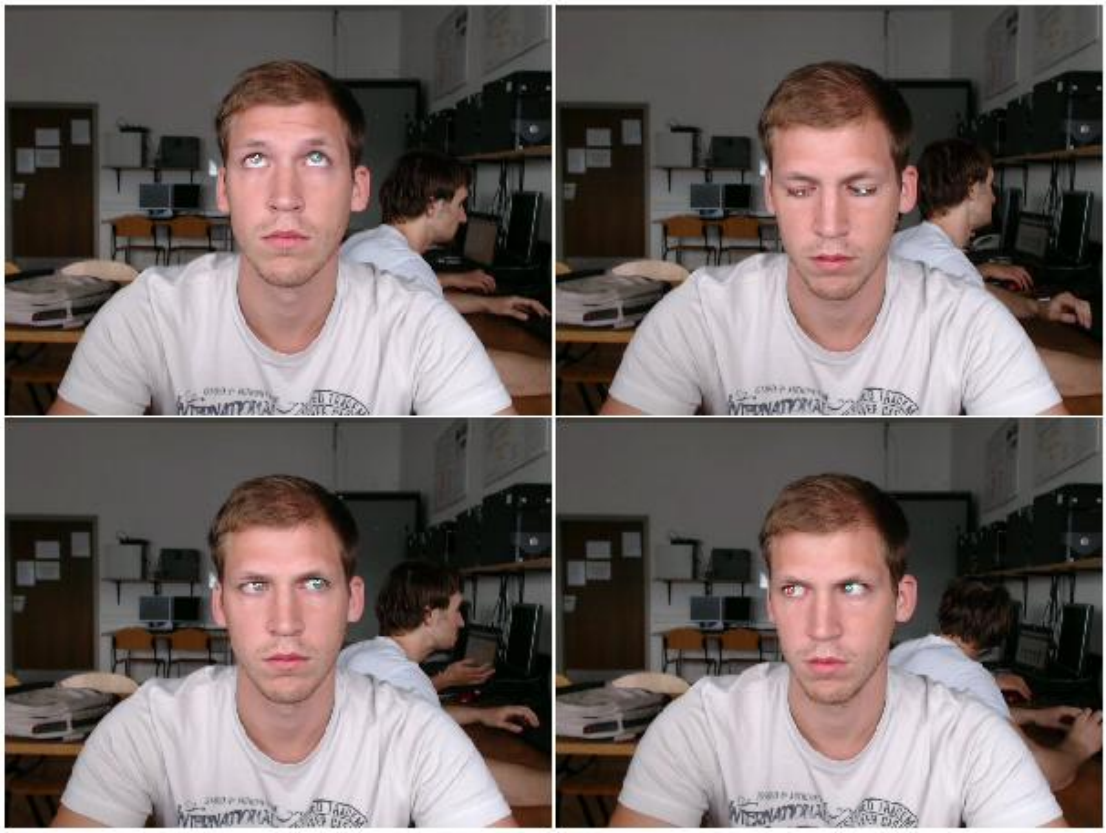
Nadalje, uspješno je prevedena metoda regresije oblika u JavaScript te ju je moguće pokrenuti i koristiti iz Web preglednika. Za sada je moguće detektirati lice i pratiti zjenice

očiju. Detekcija očiju moguća je pomoću prevedene *OpenCV* biblioteke ili bez nje korištenjem CCV JavaScript biblioteke.

Slika 14 Primjer praćenja metodom eksplicitne regresije oblika



Slika 15 Primjer izvođenja u Firefox Web pregledniku



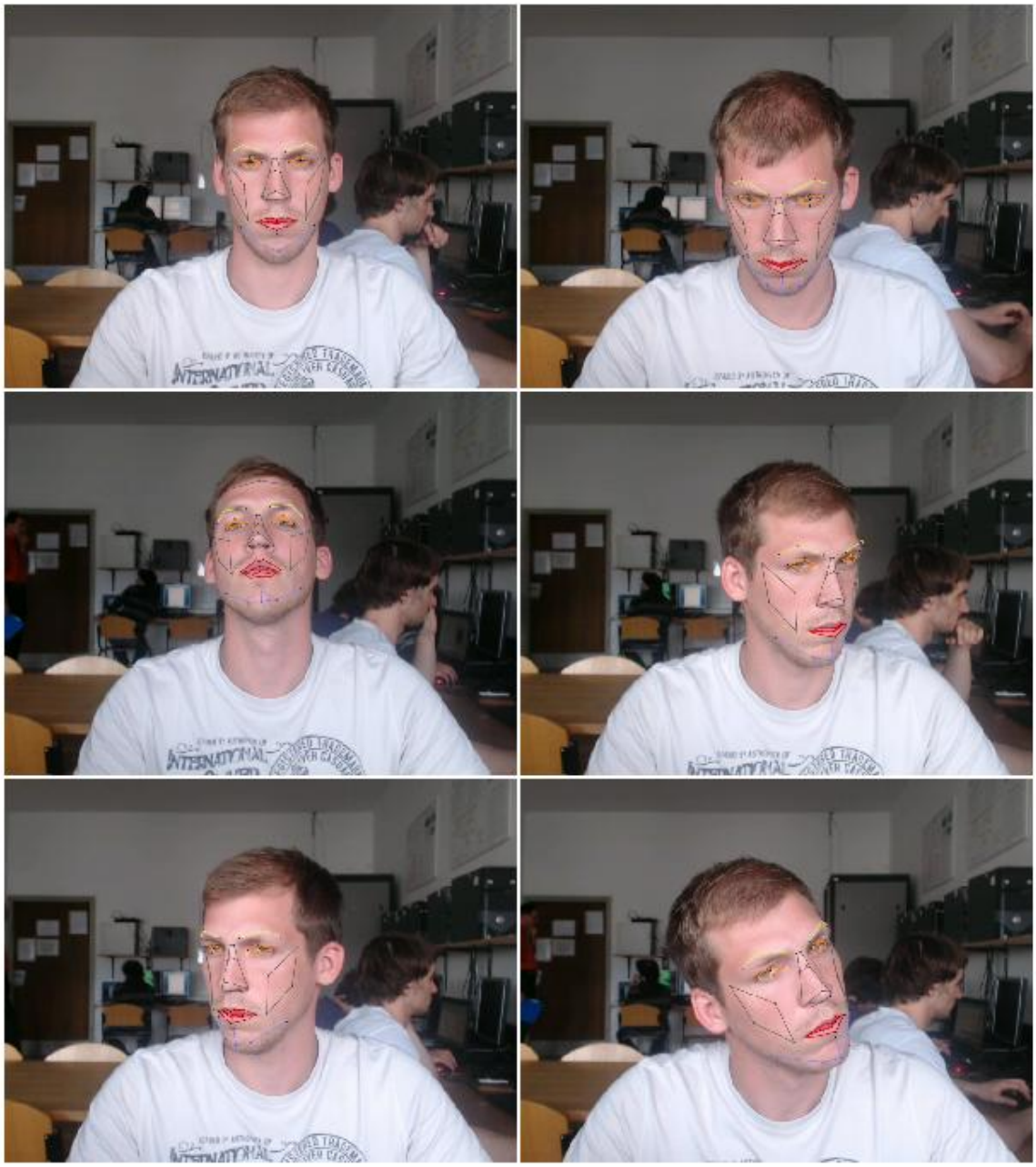
Slika 16 Primjer praćenja zjenica metodom eksplicitne regresije oblika

Za metodu su također obavljeni testovi performansi na računalima različitih specifikacija. Svi testovi izvođeni su u Web pregledniku Firefox verzija 21.0 s kamerom Logitech HD Pro Webcam C920. Prezentirana je, u obliku tablice, skraćena verzija koja prikazuje vrijednosti za konfiguraciju gdje su oba oka uključena i sve točke se iscrtavaju (najkompleksnija verzija).

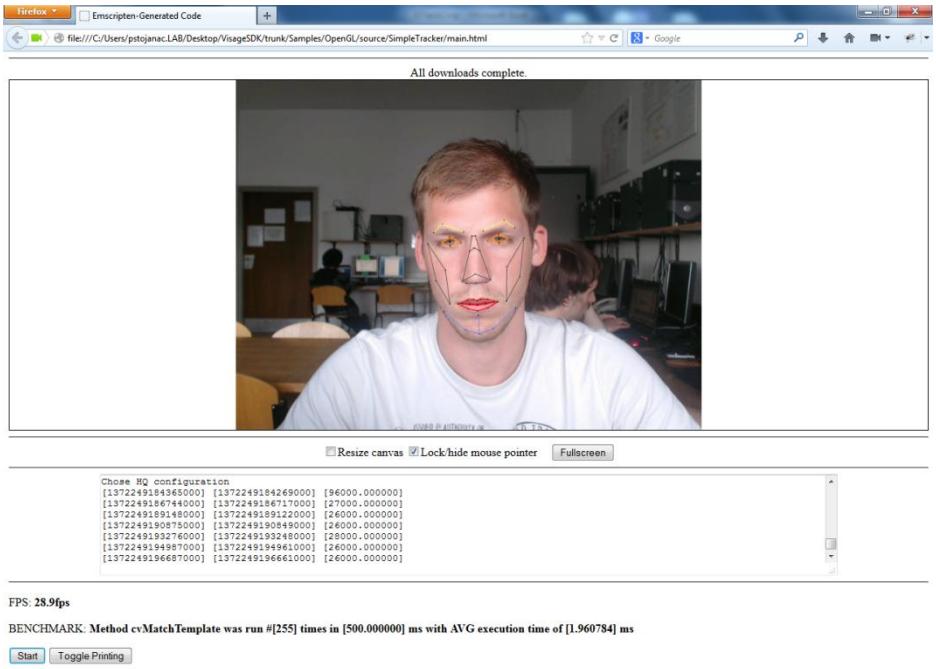
Tabela 1 Performanse metode u okvirima po sekundi na različitim konfiguracijama

CPU	Kamera (FPS)
Intel(R) Core(TM) i7-2600	36.5
Intel(R) Core(TM) i5-3210M	32.8
Intel(R) Pentium(R) 4	7.2

Prevedene u JavaScript su osnovne biblioteke Visage programskog razvojnog okruženja, *vscore* i *vsvision* gdje se nalazi implementacija algoritma za praćenje. Dakako, uz njih je prevedena *OpenCV* biblioteka i algoritam detekcije o kojoj ovisi Visage sustav. Modificirana i prevedena je i testna aplikacija koja na okvirima iz kamere detektira i prati značajke lica te prethodno pokreće grubi test performansi te tako ovisno o snazi računala bira odgovarajuću konfiguraciju.



Slika 17 Primjer praćenja značajki lica prevedenog Visage sustava



Slika 18 Primjer izvođenja u Firefox Web pregledniku

Tabela 2

CPU	Kamera (FPS)
Intel(R) Core(TM) i7-2600	27.6
Intel(R) Core(TM) i5-3210M	21.5
Intel(R) Pentium(R) 4	5.0

Stečeno je znanje o procesu prevođenja na različitim platformama i operacijskim sustavima. Dobro je upoznat alat Emscripten te njegove prednosti i nedostatci. Sve to postavlja dobre temelje za implementaciju detekcije i praćenja i na drugim platformama poput Flash-a. Mjesta za unaprjeđenje koda ima mnogo počevši od optimizacije do osiguravanja da funkcionira jednako na svakom pregledniku. Drugi način za ubrzavanje jest prevođenje u već spomenuti (poglavlje 1.2.1) podskup JavaScripta *asm.js*. Napravljeni su već inicijalni testovi no u trenutku pisanja ovog rada *asm.js* nije bio podržan u službenoj verziji Firefox-a. Pozitivna stvar je da je razlika u prevođenju u *asm.js* u odnosu na JavaScript jedna linija u skripti dok rezultati preliminarnih testova provedenih na različitim

bibliotekama u Mozilli pokazuju velika ubrzanja. To je još jedna prednost prevođenja i definitivno put kojim bi projekt mogao krenuti u budućnosti.

Zaključak

Prevođenje postojećeg koda pomoću prevoditelja nije jedini, a možda ni najoptimalniji način da se postojeći kod prebaci na drugu platformu, ali definitivno ima svoje prednosti. Prije donošenja odluke koja metoda će se koristiti potrebno je odrediti doseg i svrhu projekta. Ako je cilj rezultat u kratkom roku koji ima solidne performanse, prevoditelj je put koji ima najviše smisla.

Proces prevođenja, pogotovo ako se koriste noviji prevoditelji i biblioteke koje dosad nisu prevođene na taj način, može biti kompleksan i zahtjeva snalažljivost od programera. To je također proces kroz koji se moguće upoznati s različitim platformama te programskim jezicima. U konačnici, iz procesa je proizašao sustav s kojim je u realnom vremenu moguće iz Web kamere detektirati lice te pratiti značajke lica iz okvira u okvir. Performanse takvog sustava na modernom računalu (procesor Intel(R) i5/i7) dostatne su za ugodno korisničko iskustvo (3.3).

Valja naglasiti kako sustav dobiven iz ovog rada za sada još nije u potpunosti spreman za upotrebu na mobilnim uređajima, djelomično zato što nije optimiziran, a djelomično i zbog čimbenika na koje nije moguće utjecati poput razvoja Web preglednika te JavaScript pokretača. Sustavi koji su razvijeni za Web pate od problema nestandardizacije koji je još uvijek prisutan u razvoju Web aplikacija. U ovom slučaju se to ukazuje u velikim razlikama u performansi koje se pokazuje od preglednika do preglednika. Manjak standarda također otežava pisanje programa jer je potrebno uložiti dodatni trud kako bi se podržali svi pretraživači.

Ohrabruje činjenica da se trend kreće prema ulaganju u Web tehnologije i kako se JavaScript pokretači obogaćuju te optimiziraju na mjesečnoj bazi. Zajedno s time razvija se i sklopovlje mobilnih uređaja i definiraju se novi jezici poput *asm.js*. Uz nadolazeće nadogradnje HTML5 standarda vanjski dodaci poput Flasha gube na važnosti te izgleda da će se razvoj aplikacija detekcije i praćenja za Web nastaviti u smjeru HTML i JavaScripta.

Literatura

[1] ALON ZAKAI: "Emscripten: an LLVM-to-JavaScript compiler,", *SPLASH '11 Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 301-312, 2011.

[2] J. AHLBERG: "CANDIDE-3 -- an updated parameterized face," *Report No. LiTH-ISY-R-2326*, Dept. of Electrical Engineering, Linköping University, Sweden, 2001.

[3] C.-C. HAN, H.-Y. M. LIAO, G.-J. YU, AND L.-H. CHEN: "Fast face detection via morphology-based pre-processing," *Lecture Notes in Computer Science*, vol. 1311, pp. 469-476, 1997.

[4] G. YANG AND T. S. HUANG: "Human face detection in a complex background," *Pattern Recognition*, 27(1): pp. 53–63, 1994.

[5] H. ROWLEY, S. BALUJA, T. KANADE: "Rotation Invariant Neural Network-Based Face Detection," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 38-44, 1998.

[6] I. CRAW, D. TOCK, AND A. BENNETT: "Finding face features," *Second European Conference on Computer Vision*, pages 92-96, 1992.

[7] A. LANITIS, C. J. TAYLOR, AND T. F. COOTES: "An automatic face identification system using flexible appearance models," *Image and Vision Computing*, vol.13, no.5, pp.393-401, 1995.

[8] P. VIOLA,M. JONES: "Rapid Object Detection Using a Boosted Cascade of Simple Features", *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2001.

[9] S. K. PAKAZAD, K. FAEZ, F. HAJATI: "Face detection based on central geometrical moments of face components," *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference*, vol. 5, pp. 4225-4230, 2006.

[10] C. A. WARING, X. LIU: "Face Detection Using Spectral Histograms and SVMs," *IEEE Trans Systems, Man, and Cybernetics-Part B: C Cybernetics. 2005*, 2005.

[11] Y. ABRAMSON AND B. STEUX: "YEF* Real-Time Object Detection," *Proc. Int'l Workshop Automatic Learning and Real-Time*, 2005.

[12] C. HUANG, H. AI, Y. LI, S. LAO: "High-Performance Rotation Invariant Multiview Face Detection," *Proc. Sixth Int'l Conf. Automatic Face and Gesture Recognition*, pp. 79-84, 2004.

[13] X. CAO, Y. WEI, F. WEN, J. SUN, "Face alignment by Explicit Shape Regression," *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* , pp. 2887-2894, 2012.

Naslov, sažetak i ključne riječi (HRV i ENG)

Naslov: Detekcija i praćenje značajki lica u Web okruženju

Sažetak:

Rad prezentira rješenja za problem implementacije sustava za detekciju i praćenje lica u Web okruženju. Opisana su dva pristupa problemu, automatsko prevođenje korištenjem alata i prevođenje bez alata te je objašnjeno zašto je za ovaj rad odabrana automatska metoda. Nacrtani su dijagrami koji objašnjavaju komunikaciju i logiku sustava na Webu i opisan je postupak konverzije. S ciljem postavljanja konteksta problema proučena je teorijska pozadina te najpoznatiji algoritmi i detaljno je opisan alat za prevođenje koji se koristio – Emscripten i sustav Visage. Rezultati su prezentirani u obliku slika te grubih testova performansi koji prikazuju da je detekcija i praćenje u Webu dostižan cilj.

Ključne riječi:

Emscripten, Visage, značajke lica, Viola – Jones, HTML5, OpenCV, detekcija, praćenje

Title: Facial features detection and tracking in Web environment

Abstract:

Diploma thesis presents solutions for implementing a detection and tracking system in Web environment. Two approaches to the problem are described, automatic compilation using a compiler tool and manual compilation by rewriting the code. Reasons for choosing automatic compilation for this specfic project are also presented. Furthermore, system communication and logic are described via diagrams along with a method of compilation described in more detail. To put the problem in context, theoretical approach was also explored and some of the better known algorithms are explained. More information along with the in-depth analasys of the compiler tool Emscripten and detection and tracking system Visage are provided. Results are presented showing images of the system in action in Web environment along with a rough benchmark which together demonstrate that detection and tracking is indeed viable in Web environments.

Keywords:

Emscripten, Visage, facial features, Viola – Jones, HTML5, OpenCV, detection, tracking