iaSVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. **857**

Detekcija zadanog lica u grupnim slikama

Domagoj Rukavina

Zagreb, lipanj 2009.

Sadržaj

1. Uvod u računalni vid

Računalni vid je grana **umjetne inteligencije** u području **računarske znanosti**, koja se bavi izgradnjom računarskih automata ili modela koji imaju sposobnost izvlačenja informacije iz slika. Pri tome se smatra da je slika bilo koji format koji sadrži vizualne podatke, statička slika, video sekvenca, snimak iz više kamera itd.

Neki primjeri primjene sustava računalnog vida su:

- Kontroliranje procesa (npr. industrijski robot ili autonomno vozilo).
- Praćenje događaja (npr. video nadzor ili brojanje ljudi).
- Organiziranje informacija (npr. indeksiranje baza podataka koje sadrže slike).
- Modeliranje objekata i okoline (npr. industrijska inspekcija, analiza medicinskih slika).
- Interakcija (npr. interakcija korisnika i računala, nove generacije konzola najavile su mogućnost upravljanja gestama tijela).

Računalni vid može se smatrati komplementarnom disciplinom biološkom vidu. U biološkom vidu, proučavaju se vizualna percepcija ljudi i drugih životinja, što rezultira modelima koji objašnjavaju kako ti sustavi rade u terminima psiholoških procesa. Za razliku od toga, računalni vid proučava i opisuje umjetne vizualne sustave koji se implementiraju u softveru ili hardveru. Interdisciplinarna razmjena između biološkog i računalnog vida pokazala se veoma plodnom za oba područja istraživanja.

Poddomene računalnog vida uključuju: rekonstrukcije scene, praćenje procesa, praćenje predmeta, prepoznavanje predmeta ili ljudi, učenje, indeksiranje baza podataka, predviđanje kretanja i obnavljanje uništenih slika. U kontekstu svega rečenog, ovaj rad spada u područje učenja i prepoznavanja ljudi.

2. Detekcija lica

2.1. Uvod u detekciju lica

Detekcija lica je jedno od vrlo aktivnih područja istraživanja unutar domene računalnog vida. Ljudima je prepoznavanje lica urođena sposobnost i gotovo je nevjerojatno precizna. U stanju smo prepoznati ista lica prolaskom vremena, promjenama u izgledu, pa čak i djelomičnim prekrivanjem lica. Cilj istraživanja u ovom području je izgraditi računalni model koji će sa jednakom ili boljom preciznošću moći obavljati detekciju i prepoznavanje lica. Dakle, želja nam je izgraditi takav računalni sustav kojem bi ulaz bila slika koja sadrži lica, a izlaz identifikacija ljudi na slici. Od takvog rješenja koristi bi imala mnoga područja u kojima je potrebna identifikacija ljudi, poput banaka ili zračnih luka gdje bi se mogla dramatično pojačati sigurnost i smanjiti mogućnost krađa na vrlo efikasan način.

Posljednjih godina razvijeno je mnoštvo modela koji se hvataju u koštac sa ovim problemom, više ili manje uspješno. U ovom radu razmatra se jedan od njih, a to je Viola-Jones algoritam koji se pokazao iznimno brzim u detekciji lica i de facto postavio standarde u daljnjem razvoju na ovom području.

2.2. Viola – Jones algoritam

Viola-Jones algoritam objavljen je 2004. godine u časopisu Računalni vid. To je bio prvi detektor lica koji je detektirao lica u realnom vremenu, te se danas smatra ključnim napretkom u istraživanjima detekcije lica.

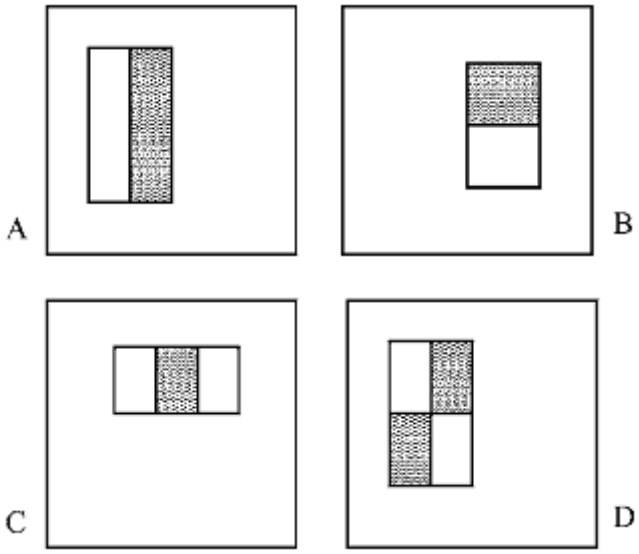
Prije Viole-Jones algoritma, osnovni princip detekcije lica bio je preskalirati originalnu sliku na veličinu detektora. Iako se čitatelju naizgled može učiniti da je vremenski jednako mijenjati veličinu slike ili detektora, Viola-Jones pokazali su da je daleko efikasnije mijenjati veličinu detektora nego slike. U daljnjem tekstu opisuju se osnovni koncepti ključni za razumijevanje rada algoritma.

2.2.1. Značajke

Viola-Jones algoritam klasificira slike pomoću izračuna jednostavnih značajki(engl. *features*). Postavlja se pitanje, zašto računati značajke umjesto izravnog korištenja piksela. Odgovor na to pitanje dali su sami autori. Jedan od razloga je

lakše učenje sustava koji koristi značajke, te da je takav sustav daleko brži od sustava temeljenog na računanju nad pikselima.

Značajke podsjećaju na Haarove funkcije[3]. Koriste se tri vrste značajki. Vrijednost *dvodijelne pravokutne* značajke određuje se razlikom između suma piksela unutar pravokutne regije cijele značajke. Pravokutne regije su susjedne (vertikalno ili horizontalno) i iste veličine. *Trodijelna pravokutna* značajka računa se kao suma dva vanjska pravokutnika od čega se oduzima suma srednjeg prvokutnika. Posljednja, *četverodijelna pravokutna* značaka, je značajka koja se sastoji od četiri pravokutnika kod koje se od sume dijagonalnih pravokutnika oduzima suma druga dva dijagonalna pravokutnika.



Slika 1 Prikaz Haarovih značajki. Značajke sa dva pravokutnika (A) i (B). Značajka sa tri pravokutnika(C). Značajka sa četiri pravokutnika (D).

Osnovna rezolucija detektora koju su autori koristili iznosila je 24x24 piksela. Time su nad slikom od 384x288 piksela generirali oko 160 000 značajki.

2.3. Integralna slika

Koncentrirani čitatelj lako će zaključiti da ovako opisano računanje značajki može konzumirati puno procesorskog vremena. Uistinu kad bismo računali značajke ovako kako je opisano, čak i na računalima u vrijeme pisanja ovog rada, nepotrebno bi trošili procesorsko vrijeme. Pokazalo se da je moguće izračunati npr. sumu značajke sa dva pravokutnika korištenjem samo 8 referenci na

integralnu sliku. Stoga se uvodi novi pojam – *integralna slika.* Integralna slika definira se sljedećom formulom:

$$ii(x,y) = \sum_{x' \le x, ' \le y} i(x', y') \qquad (1.1)$$

gdje je *ii(x,y)* integralna slika, a *i(x,y)* originalna slika. Intuitivno, integralna slika na mjestu x,y jednaka je sumi svih piksela lijevo i iznad(Slika 2.).

1	1	1
1	1	1
1	1	1

1	2	3
2	4	6
3	6	9

Slika 2 Vrijednost integralne slike na mjestu x,y suma je piksela svih piksela lijevo i iznad

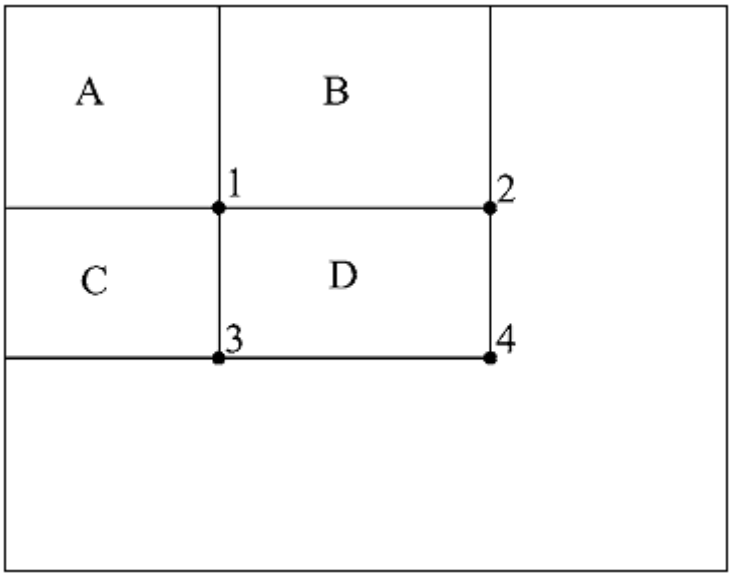
Korištenjem sljedeće dvije formule integralna slika može se izračunati u jednom prolazu:

$$s(x,y) = s(x,y-1) + i(x,y) \qquad (1.2)$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \qquad (1.3)$$

gdje s(x,y) predstavlja sumu reda integralne slike. Pri tom vrijedi da je *s(x,-1)=0*, te također *ii(-1,y)=0*.

Korištenjem integralne slike bilo koja pravokutna suma može se dobiti korištenjem samo četiri reference (Slika 3.). Očito da bi se dobila razlika između dvije pravokutne sume potrebno nam je osam referenci. Budući da su pravokutnici susjedni, moguće ih je računati poznavanjem samo šest referenci, tj. osam u slučaju trodijelnih pravokutnih značajki i devet u slučaju četverodijelnih značajki.



Slika 3 Suma piksela unutar pravokutnika D može se izračunati preko samo četiri reference. Vrijednost integralne slike kod točke 1 jednaka je sumi piksela pravokutnika A, kod točke 2 je suma A+B, kod točke 3 je suma A+C i kod 4 je A+B+C+D. D se potom racuna kao 4+1-(2+3).

Ovakvim računanjem suma broj korištenih UI operacija procesora smanjuje se za otprilike sto puta.

Ovim postupkom dobiva se veliki broj značajki. Za sliku veličine 384x288 piksela oko 160 000 značajki. Empirijskim zaključivanjem Viola i Jones došli su do spoznaje da samo manji dio tih značajki može kvalitetno klasificirati lica od ne-lica.

2.4. AdaBoost

Kako bi našli značajke koje su „najbolje“ za raspoznavanje lica od ne-lica, nad skupom svih značajki primijenili su modificirani AdaBoost algoritam. AdaBoost gradi linearnu kombinaciju slabih klasifikatora kako bi stvorio jaki klasifikator.

2.4.1. Slabi klasifikator

Slabi klasifikator je takva funkcija koja točno klasificira više od pola elemenata ulaznog skupa, tj. točno klasificira 51% primjera.

Za proces treniranja potrebna su nam dva skupa. Prvi sadrži slike traženih lica, a drugi ne-lica koje želimo eliminirati. Te slike je prvo potrebno skalirati na veličinu 24 × 24 i normalizirati ih. Za svaku Haarovu značajku se stvara slabi klasifikator tako da se ta značajka izračuna na skupu za treniranje. Konstrukcijom težinskog histograma nalazi se optimalni prag koji odjeljuje lica od ne-lica za tu značajku, a

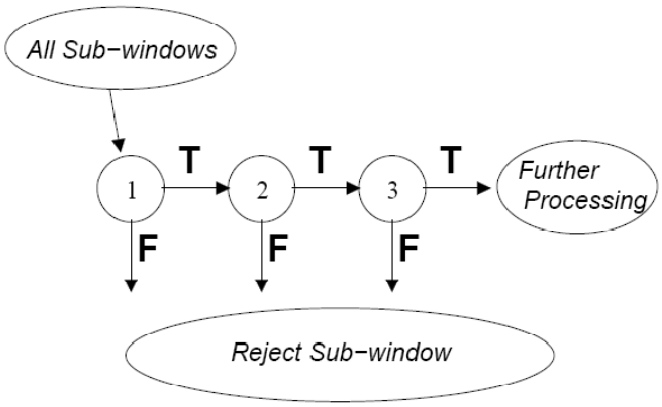
potom se uz svaki klasifikator h_j veže Haarova značajka f_j, prag Θ_j i paritet p_j. Paritet p_j definira za danu značajku da li vrijednost koja se dobije primjenom na sliku lica pozitivna ili negativna.

$$h_j(x) = \begin{cases} 1, & p_j f_j(x) \leq \Theta_j p_j \\ 0, & inace \end{cases} \quad (1.4)$$

2.4.2. Treniranje AdaBoosta

Nad tako dobivenim setom slabih klasifikatora primjenjuje se AdaBoost algoritam koji od njih gradi jaki klasifikator.

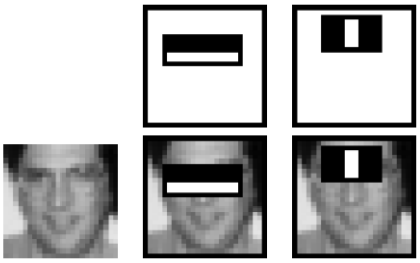
Viola-Jones pokusali su na dva načina graditi takav klasifikator. Jedan od načina bio je generiranje oko dvjestotinjak značajki pomoću kojih se radila klasifikacija. Takav klasifikator bio je vrlo precizan u svom radu, međutim računanje dvjesto značajki pokazalo se kao vrlo spor postupak. Drugi način je daleko efikasniji u proračunu, a radi se o gradnji kaskade jednostavnih detektora od kojih se svaka kaskada sastoji od malog broja jednostavnih značajki.



Slika 4 Prikaz kaskade detektora

AdaBoost algoritam služi za treniranje pojedine razine kaskade. Sam algoritam odabire najbolja obilježja koja odvajaju slike lica od slika ne-lica. Za rad algoritma potreban je veći broj slika lica, te slučajno odabranih slika koje nisu lica. Svakoj značajki algoritam pridjeljuje težinu i prag odlučivanja, koji razdjeljuju slike lica od slika ne-lica. Svaka razina kaskade je detektor koji dobro detektira lica, ali generira veliki broj lažnih lica(engl. *false positive*). Pažljivim treniranjem svakog stupnja

kaskade ta se greška smanjuje u svakom koraku. Krajnji rezultat rada AdaBoosta su značajke koje najbolje klasificiraju lice od ne-lica. (Slika 4.)



Slika 5 Najbolje značajke koje je odabrao AdaBoost. Prva opisuje razliku intenziteta čela i područja oko očiju lica, a druga određuje razliku intenziteta nosa i očiju.

Tablica 1 Pseudokod AdaBoost algoritma

- Uz dani skup primjeraka za učenje $(x_1, y_1), \ldots, (x_n, y_n)$ gdje je $y_i = 0,1$ za negativne i pozitivne primjerke.
- Inicijaliziraj težinske faktore $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ za $y_i = 0,1$, gdje su m i l brojevi pozitivnih i negativnih primjera za učenje
- Za $t = 1, \ldots, T$
  - Normaliziraj težine: $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  - Za svaku značajku j, treniraj klasifikator h_j koji je ograničen na upotrebu jedne značajke. Greška se evaluira s obzirom na w_t, $\varepsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  - Odaberi klasifikator h_t, s najmanjom pogreškom ε_t.
  - Popravi težine: $w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$, gdje je $e_i = 0$ ako je x_i dobro klasificiran, a $e_i = 1$ u suprotnom. Vrijedi $\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}$
- Konačni jaki klasifikator je sljedeći: $h(x) = \begin{cases} 1, & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, & inace \end{cases}$ gdje je $\alpha_t = \log\left(\frac{1}{\beta_t}\right)$

Detaljnije o implementaciji samog algoritma može se pronaći u [1],[4] i [6].

Tablica 2 Pseudokod Viola Jones algoritma

Korisnik izabire vrijednosti *f*- maksimalnu vjerojatnost lažne detekcije (engl. *false positive rate*) razine, te *d* – minimalnu vjerojatnost detekcije znaka po razini

- Korisnik izabire ukupni false positive rate, F_{target}
- P = skup pozitivnih primjera
- N = skup negativnih primjera
- $F_0=1.0$; $D_0=1.0$
- i=0
- Dok je $F_i > F_{target}$
  - i=i+1;
  - $n_i=0$; $F_i = F_{i-1}$;
  - dok je $F_i > f * F_{i-1}$
    - $n_i = n_i + 1$
    - koristi P i N za učenje klasifikatora sa n_i obilježja koristeći AdaBoost
    - evaluiraj trenutni klasifikator kako bi se odredili F_i i D_i
    - smanjuj prag odlučivanja za *i*-ti klasifikator sve dok trenutni klasifikator ne postigne vjerojatnost detekcije znaka barem $d*D_{i-1}$
- N={Ø}
- Ako je $F_i > F_{target}$ evaluiraj trenutni detektor na skupu ne-znakova i stavi lažne detekcije u skup N

2.5. OpenCV

Kako je detekcija lica samo jedan od koraka u razvoju programske podrške u ovom radu, primjenjuje se već gotova implementacija ViolaJones detektora unutar OpenCV biblioteke funkcija.

OpenCV je otvorena biblioteka funkcija iz računalnog vida. Originalno razvijana od strane Intela, dok je danas otvorenog koda i za njen daljnji razvoj brine zajednica entuzijasta i profesionalaca.

OpenCV instalacijska datoteka može se dohvatiti sa http://opencvlibrary.sourceforge.net/.

3. Raspoznavanje lica

Nakon što smo detektirali sva lica na slici sljedeći korak koji bi željeli napraviti je izgraditi računalni sustav kojem će ulaz biti jedna ili više slika istog lica, a izlaz će biti odgovor na pitanje „Nalazi li se zadano lice na nekoj slici?“.

Problem raspoznavanja lica [7] dijeli se na dva logička slučaja: samo **raspoznavanje lica** i **verifikacija**. Oba problema pretpostavljaju da postoji galerija slika sa poznatim identitetima. Kod problema raspoznavanja testna slika lica daje se sustavu kao ulaz, a zadatak je sustava da odluči nalazi li se zadano lice unutar galerije slika. Kod verifikacije, na ulaz se daju ista testna slika, kao kod raspoznavanja, i identitet koji smo saznali raspoznavanjem. Zadatak je verifikacije da iz galerije uzme sva lica za dani identitet i usporedi ih sa testnom slikom, te da odluči radi li se o istoj osobi.

Kako se navodi u [7], postoji više načina da se napravi raspoznavanje lica. Jedan od načina je usporedba varijacija nad slikama istog lica kako bi računalo naučilo prepoznavati dano lice. Druga metoda ne koristi učenje, a temelji se na kompleksnim značajkama koje se izračunavaju nad licima.

Viola i Jones vođeni idejama iz prijašnjih radova osmislili su vlastiti sustav. Njihov pristup ponovo se temelji na odabiru dobrih značajki postupkom učenja korištenjem AdaBoost algoritma. Iz velikog seta značajki koji se generira nad slikom odabiru se one značajke koje su osjetljive na obilježja različitih osoba, ali neosjetljive na različita obilježja jedne te iste osobe odnosno lica. Rezultat njihovog algoritma je klasifikator koji kombinira nekoliko stotina značajki koje se mogu izračunati nad parom ulaznih slika u manje od milisekunde.

Postupak je sljedeći: na početku je potrebno izgraditi konačnu bazu slika poznatih lica. Moguće je da u bazi postoji više slika lica za isti identitet. Sustav izvlači parove istih lica iz baze, te pomoću njih uči koje značajke najbolje odražavaju to lice.

Ulaz algoritma su parovi „isto lice“ i „različita lica“ iz kojih algoritam uči, odnosno bira najbolje značajke. Naposljetku algoritam konstruira funkciju sličnosti (engl. *similarity function)* koja daje numeričku mjeru sličnosti dvaju lica u odnosu na neku

granicu (engl. *threshold*). Alternativno, sustav se može napraviti tako da lica ne odbacuje u odnosu na granicu, već da uvijek izračuna koje je lice iz baze najsličnije zadanom licu.

Međutim u ovom radu, pristup problemu je ponešto banalniji od dosad opisanih ideja Viole i Jonesa. Naime ne vrši se nikakav postupak učenja najboljih značajki. Značajke se samo izračunavaju uzduž i poprijeko, te korištenjem svih skaliranja, nad zadanom slikom lica. Potom se vrši računanje korelacije nad dvama vektorima vrijednosti koji se dobivaju izračunavanjem značajki nad danim slikama, tj. onima dobivenim iz tražene slike i onima dobivenim iz detektiranog lica. Vrijednosti koje se pritom dobivaju mogu biti indikativne, ali nikako apsolutno točne, osim u nekim posebnim slučajevima, ali o tome kasnije.

Na početku se pokreće detekcija lica za zadanim parametrima. Poziva se aplikacija (opisana malo kasnije u tekstu) koja vraća listu detektiranih lica.

Nakon što detektor lica „isporuči“ listu detektiranih lica moguće je započeti postupak izračunavanja značajki. Kako bi osigurali jednaku duljinu vektora vrijednosti značajki, potrebno je slike lica nad kojima se izračunavaju značajke svesti na jednaku veličinu. Odlučeno je da se detektirana lica preskaliraju bikubičnom interpolacijom na veličinu traženog lica. Nakon toga izračunavaju se integralna slika detektiranog i traženog lica, te se potom računaju vrijednosti značajki na način kako je opisano u prethodnom poglavlju.

Dobivena dva vektora vrijednosti potom se uspoređuju na različite načine. Zapravo matematički se pitamo koliko je jedan vektor sličan drugome. Odgovor na to pitanje daje nam statistika. Matematički pojam koji „mjeri“ sličnost između dva n-dimenzionalna vektora naziva se korelacija. Sličnost vektora mjerit će se na tri načina: Euklidskom udaljenošću, Kosinus sličnošću i Pearsonovom korelacijom.

Euklidska udaljenost. Euklidska udaljenost temelji se na Pitagorinom teoremu. To je klasična udaljenost između dva vektora. Npr. udaljenost između vektora u dvodimenzionalnom prostoru računamo kao :

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (2.1)$$

gdje su x i y koordinate točaka u dvodimenzionalnom prostoru.

U trodimenzionalnom prostoru poznata nam je sljedeća formula:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.2}$$

Međutim pri računanju značajki dobivat ćemo vektore vrijednosti puno većih dimenzija. Definiramo li vektor vrijednost značajki nad detektiranim licem kao $\vec{P} = (p_1, p_2, ..., p_n)$ i vektor vrijednosti značajki nad traženim licem $\vec{Q} = (q_1, q_2, ..., q_n)$ tada ćemo udaljenost između tih vektora računati kao:

$$D = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{2.3}$$

Ova mjera nam govori koliko su dva vektora udaljena jedan od drugoga. Što je ovaj broj manji reći ćemo da su lica sličnija. U idealnom slučaju dobit ćemo vrijednost nula, što znači da se radi o identičnim vektorima.

Kosinus sličnost. Kosinus sličnost je mjera sličnosti n-dimenzijskih vektora računanjem kosinusa kuta koji zatvaraju dva vektora. Definiramo li vektore P i Q na analogan način kao kod Euklidske udaljenosti, tada kosinus sličnost računamo prema sljedećoj formuli:

$$S = \cos(\Theta) = \frac{P \cdot Q}{|P||Q|} \tag{2.4}$$

gdje je Θ kut između vektora P i Q, a |P| i |Q| norme vektora.

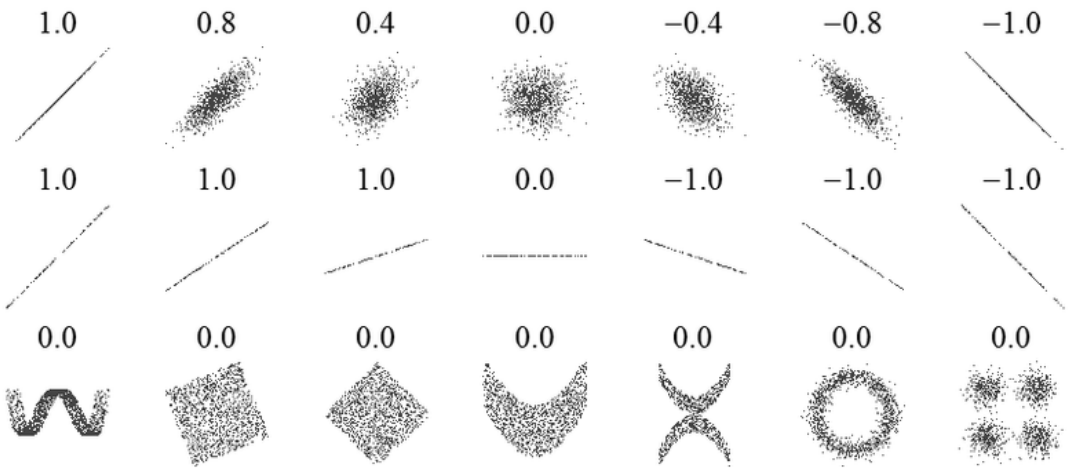
Vrijednost ove funkcije može biti između [-1,1]. Vrijednost je nula kad su vektori okomiti. Kod usporedbe lica reći ćemo da su lica to sličnija što je ovaj broj bliže jedinici.

Pearson korelacija. Pearson korelacija je linearna mjera sličnosti između dva vektora. Vrijednost korelacije je broj iz intervala [-1,1]. Korelacija +1, znači da postoji savršena pozitivna linearna veze između dva vektora. Formula za izračunavanje Pearsonovog koeficijenta nad danim uzorkom je sljedeća:

$$r = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \overline{X})^2} \sqrt{\sum_{i=1}^{n} (Y_i - \overline{Y})^2}} \tag{2.5}$$

gdje su x i y vrijednosti vektora, a $\overline{X}$ i $\overline{Y}$ srednje vrijednosti vektora. Za dokaz ove formule vidi [10]. Pri usporedbi lica reći ćemo da su lica to sličnija što je Pearsonov koeficijent bliži vrijednosti jedan.

U dodatku B mogu se pregledati algoritmi koji računaju ove koeficijente.



Slika 6 Primjeri vrijednosti korelacija nad različitim distibucijama parova (x,y)

4. Implementacija

U okviru rada bilo je potrebno razviti i programsku potporu kojom se implementiraju dosad opisani algoritmi. Za razvijanje programske potpore odabrani su jezici C++ i C#. C++ je odabran ponajviše zbog svoje brzine izvođenja i lakšeg povezivanja sa OpenCV bibliotekom, iako je moguće povezivanje OpenCV biblioteke i sa C#.

Razvijeni program sastoji se od dva dijela. Prvi dio je Windows Console Aplikacija koja detektira lica uz pomoć OpenCV-a, a drugi dio je Windows Forms Aplikacija koja poziva detekciju kao novi proces i obavlja daljnje izračunavanje.

Budući da je za daljnji rad potrebno razvojno okruženje Visual Studio, opisat ćemo ukratko postupak konfiguracije Visual Studija kako bi mogao raditi sa OpenCV bibliotekom.

4.1. Konfiguracija razvojnog okruženja

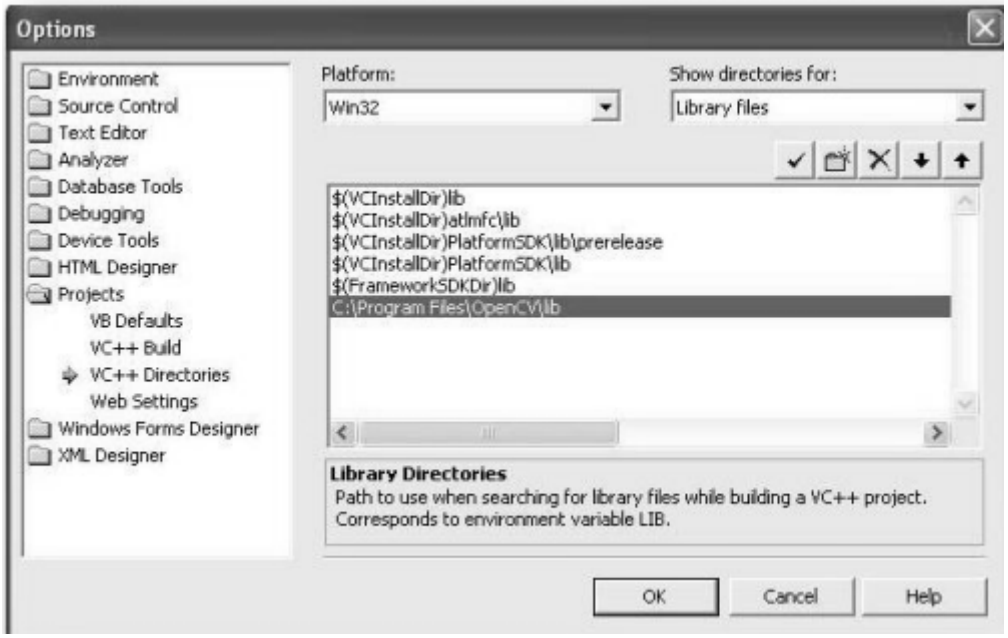
Postupak konfiguracije može se pogledati detaljno pod [8], ovdje će ipak biti ukratko objašnjeno što se treba konfigurirati. OpenCV instalacija može se dohvatiti sa već prije spomenute adrese: http://sourceforge.net/projects/opencvlibrary ili ukoliko SourceForge nije dostupan pretragom na Googleu.

Poželjno je instalirati OpenCV nakon instalacije Visual Studija jer će u tom slučaju čarobnjak automatski konfigurirati sve što je potrebno. U slučaju da čarobnjak ipak ne uspije, slijedi postupak konfiguracije. Također je poželjno instalaciju obaviti sa originalnim postavkama, kako bi lakše bilo pratiti daljnje upute u slučaju pogreške.

Za početak, potrebno je raspakirati OpenCV arhivu i kopirati sve .dll datoteke iz OpenCV arhive u Windows/System32 direktorij. Nakon toga pokrećemo Visual Studio. Kad se učita Visual Studio u alatnoj traci odabere se Tools → Options. Pojavit će se novi prozor.

U novom prozoru u listi s lijeve strane (potrebno je odabrati Projects → VC++ Directories, te u gornjem desnom uglu pod stavkom Show directories odabrati Library files. Potom treba dodati sljedeći direktorij u listu:

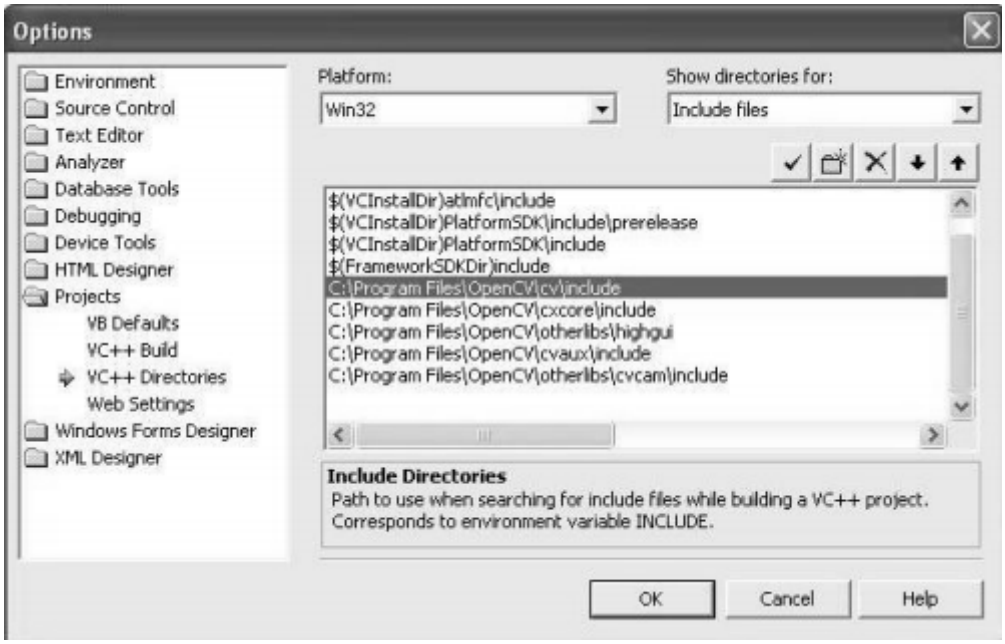
- *„C:\Program Files\OpenCV\lib.“*



Slika 7 Opcije koje je potrebno konfigurirati u Library files

Zatim je potrebno u padajućem izborniku odabrati Include files opciju i dodati sljedeće direktorije (Slika 7.):

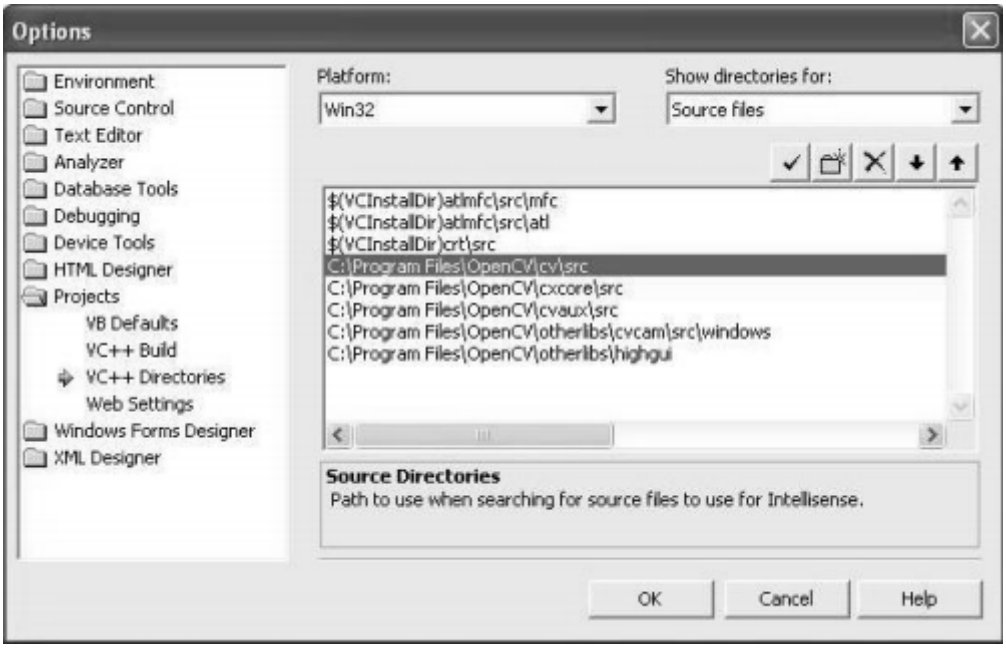
- *"C:\Program Files\OpenCV\cv\include"*
- *"C:\Program Files\OpenCV\cxcore\include"*
- *"C:\Program Files\OpenCV\otherlibs\highgui"*
- *"C:\Program Files\OpenCV\cvaux\include"*
- *"C:\Program Files\OpenCV\otherlibs\cvcam\include"*



Slika 8 Opcije koje je potrebno konfigurirati u izborniku Include files

Naposljetku je još potrebno iz padajućeg izbornika odabrati Source files i dodati sljedeće direktorije(Slika 8.):

- *"C:\Program Files\OpenCV\cv\src"*
- *"C:\Program Files\OpenCV\cxcore\src"*
- *"C:\Program Files\OpenCV\cvaux\src"*
- *"C:\Program Files\OpenCV\otherlibs\highgui"*
- *"C:\Program Files\OpenCV\otherlibs\cvcam\src\windows".*



Slika 9 Opcije koje je potrebno podesiti u izborniku Source Files

Odaberite OK i Visual Studio je konfiguriran za rad sa OpenCV-em.

Nažalost, pokazalo se da spajanjem i razdvajanjem projekata, Visual Studio ponekad „počisti“ ovu konfiguraciju. U trenutku kad se pojave greške linkera, prvi korak bi bio ponovo provjeriti ove postavke.

4.2. Detektor lica

Izvorni kod koji se opisuje u ovom poglavlju nalazi se u dodatku A. Detekcija lica realizirana je kao Windows Console aplikacija koja se kasnije programski poziva kao zasebni proces. Napisana je u programskom jeziku C++.

Radi se o vrlo jednostavnoj aplikaciji koja na svoj ulaz prima pet parametara. Prvi parametar je staza do .xml datoteke koja sadrži već gotovi klasifikator. Takve datoteke generira algoritam treniranja iz OpenCV biblioteke, pa će se ta ista datoteka morati poslati i detektoru. Ukoliko je OpenCV instaliran po pretpostavljenim postavkama, ovi klasifikatori mogu se naći u mapi „C:\Program Files\OpenCV\data\haarcascades“. Iz praktičnih razloga dotične datoteke mogu se također pronaći u Radnom direktoriju završnog rada.

Drugi parametar koji prima aplikacija je osnovna velicina detektora. Viola-Jones algoritam radi sa osnovnom veličinom 24x24 piksela, ali moguće je eksperimetiranje i sa drugim veličinama.

Treći parametar je faktor skaliranja osnovnog detektora. Viola i Jones u svom radu koriste povećanje od 25% u svakom novom prolazu kroz sliku tijekom postupka detekcije.

Četvrti je parametar staza do slike nad kojom će se vršiti detekcija, a peti je parametar staza i ime tekstualne datoteke u koju će se zapisati pozicije detektiranih lica. Kasnije se dotična datoteka parsira kako bi se prikazale pozicije lica.

U pretpostavljenom načinu rada aplikacija ne ispisuje nikakve podatke. Ukoliko se želi vidjeti izlaz same aplikacije moguće je uključiti testni način rada postavljanjem varijable testMode na vrijednost veću od 0. U testnom načinu aplikacija iscrtava zadanu sliku i crvene pravokutnike oko svih detektiranih lica.

Cijela aplikacija je u potpunosti odvojiva od ostatka kôda, te se kao takve može koristiti i izvan okvira ovog projekta.

Neovisno o načinu rada izlaz programa je tekstualna datoteka u kojoj se su zapisane pozicije svih detektiranih lica. Lica su zapisana kao pravokutnici, sa X i Y vrijednostima gornjeg lijevog vrha pravokutnika, te širinom i visinom pravokutnika. Indeksi X i Y određuju se iz koordinatnog sustava slike kojem je središte u gornjem lijevom vrhu. (Tablica 3.)

Tablica 3 Primjer tekstualne datoteke sa zapisom jednog lica

```
----------
L: 2 // redni broj lica
53   //X vrijednost gornjeg lijevog vrha pravokutnika
21   //Y vrijednost gornjeg lijevog vrha pravokutnika
39   //Širina pravokutnika
39   //Visina pravokutnika
----------
```

Ovisno o parametrima koji se zadaju detekcija lica može potrajati i duže od nekoliko sekundi. Eksperimentalno se pokazalo da najdulje detekcije traju do tridesetak sekundi. Strpljenje je vrlina.

4.3. Raspoznavanje lica

Postupak raspoznavanja lica temelji se na ranije opisanim idejama. Unutar C# forme izračunavaju se značajke, potom se dobiveni vektori uspoređuju korelacijskim metodama i dobiveni rezultati prikazuju se na formi.

Postupak prepoznavanja započinje klikom na gumb „Usporedi“(detaljne upute o korištenju programa nalaze se u sljedećem poglavlju). Algoritam na početku generira integralnu sliku od originalne slike. Kod koji generira integralnu sliku može se vidjeti u dodatku B. Sljedeći odsječak koda računa jednu značajku, na svim pozicijama i skaliranjima, nad danom integralnom slikom.

Tablica 4 C# izvorni kod koji generira značajke nad danom integralnom slikom.

```
foreach ( Feature Feature in baseFeatures )
{
// nova veličina značajke ( skaliranje )
  double tmpOsnovnaVelicina = this.osnovnaVelicinaFeatura;
  while ( tmpOsnovnaVelicina < integralImageTrazena.Width )
  {
    for (int y = 0 ; y < integralImageTrazena.Height
                        - ( tmpOsnovnaVelicina ) ; y += korakFeature )
    {
         for ( int x = 0 ; x < integralImageTrazena.Width
                        - ( tmpOsnovnaVelicina ) ; x += korakFeature )
         {
// Izračunaj novu značajku i dodaj je u listu generiranih značajki za
//svaki kanal
//Plavi kanal
genFeaturiTrazenoLice.Add( BaseFeatures.IzracunajBaseFeatureBlue(
                           Feature,
                           integralImageTrazena,
                           x, y, tmpOsnovnaVelicina ) );
//Zeleni kanal
genFeaturiTrazenoLice.Add( BaseFeatures.IzracunajBaseFeatureGreen(
                           Feature,
                           integralImageTrazena,
                           x, y, tmpOsnovnaVelicina ) );
//Crveni kanal
genFeaturiTrazenoLice.Add( BaseFeatures.IzracunajBaseFeatureRed(
                           Feature,
                           integralImageTrazena,
                           x, y, tmpOsnovnaVelicina ) );
         }
     }
  tmpOsnovnaVelicina *= (double)scaleFeature;
  }
}
```

Samo izračunavanje značajki provodi se na način kako je opisano ranije (Slika 3). Svaka značajka sastoji se od nekoliko pravokutnih dijelova. Za izračunavanje pravokutnih dijelova potrebne su nam najmanje četiri reference, u slučaju

susjednih vrhova ponekad i manje. U sljedećoj tablici prikazan je kod koji računa jednu značajku na zelenom kanalu:

Tablica 5 C# izvorni kod koji izračunava jednu značajku na danoj poziciji

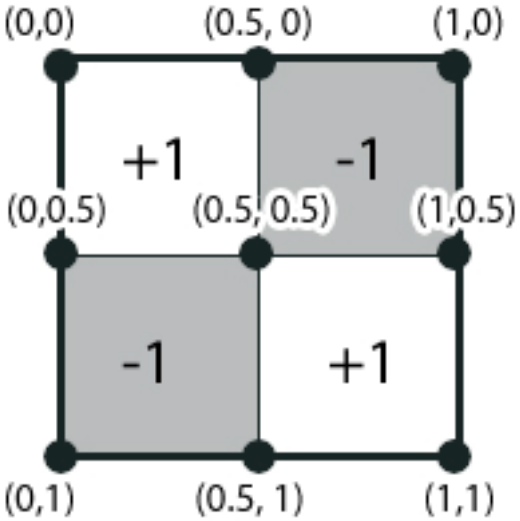
```
public static double IzracunajBaseFeatureGreen( Feature feature,
                                                 IntegralImage ii,
                                                 int x, int y,
                                                 double osnovnaVelicina )
        {
            double suma = 0;
            // značajka se sastoji od pravokutnih regija, moramo računati
            // sumu svake pravokutne regije
            for ( int i = 0 ; i < feature.Pravokutnici.Count ; i++ )
            {
            // računamo granice pravokutnika
                double lijevo = x + feature.Pravokutnici[ i ].Left *
                                osnovnaVelicina;
                double gore = y + feature.Pravokutnici[ i ].Top *
                                osnovnaVelicina;
                double desno = x + feature.Pravokutnici[ i ].Right *
                                osnovnaVelicina;
                double dolje = y + feature.Pravokutnici[ i ].Bottom *
                                osnovnaVelicina;
           // računamo potrebne zbrojeve pravokutnika prema slici 3
                double A =ii.IntegralImageGreen[(int)gore,(int)lijevo];
                double AB=ii.IntegralImageGreen[(int)gore,(int)desno];
                double AC=ii.IntegralImageGreen[(int)dolje,(int)lijevo];
                double ABCD=ii.IntegralImageGreen[(int)dolje,(int)desno];
           // suma unutar pravokutnika D je…
                double D = ABCD + A - ( AB + AC );
           // na kraju samo provjera treba li pravokutnu regiju oduzeti
           // ili zbrojiti u krajnju sumu značajke
                if ( feature.Pravokutnici[ i ].Sign > 0 )
                {
                    suma += D;
                }
                else
                {
                    suma -= D;
                }
            }
           // vraćamo izračunatu vrijednost značajke
            return suma;
        }
```

Ukupno je definirano 115 značajki. Cijeli skup značajki prikazan je na Slici 10 i Slici 11. Značajke su zapisane u tekstualnoj datoteci *features.txt*. Program na početku mora učitati listu ovih značajki kako bi ih kasnije mogao izračunavati. Svaka značajka sastoji se od n pravokutnih dijelova. Veličina značajke definira se relativnim koordinatama na intervalu [0,1] po x i y osi. Kasnije se ta veličina množi sa osnovnom veličinom značajke kako bi se dobile prave vrijednosti.

Npr. značajka u prvom redu, trećem stupcu na slici 10 definirala bi se u tekstualnoj datoteci na sljedeći način:

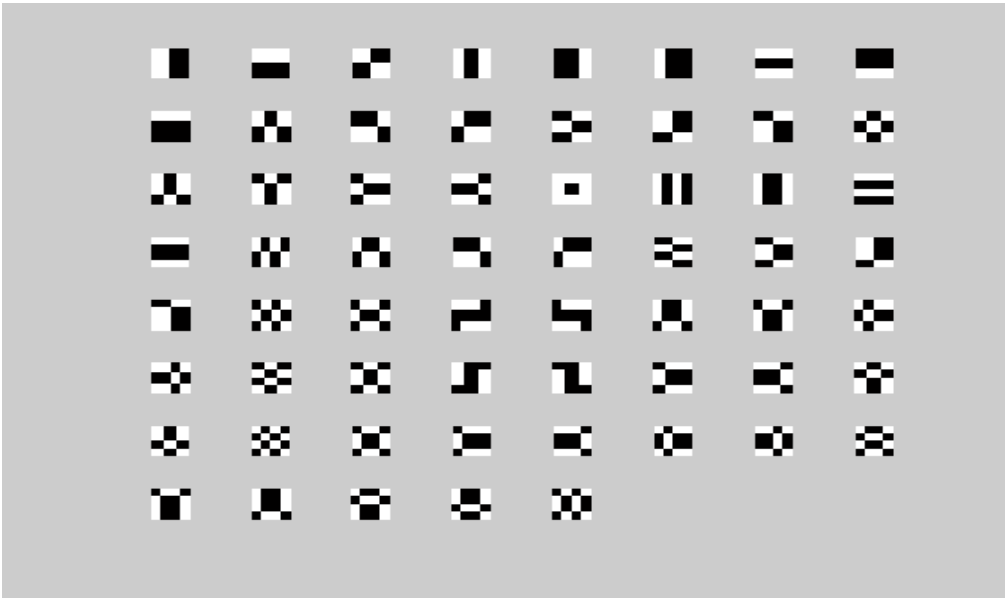
Tablica 6 Primjer zapisa značajke

```
…
feature2x2_1  4
0.00 0.00 0.50 0.50   1
0.50 0.00 1.00 0.50  -1
0.00 0.50 0.50 1.00  -1
0.50 0.50 1.00 1.00   1
…
```

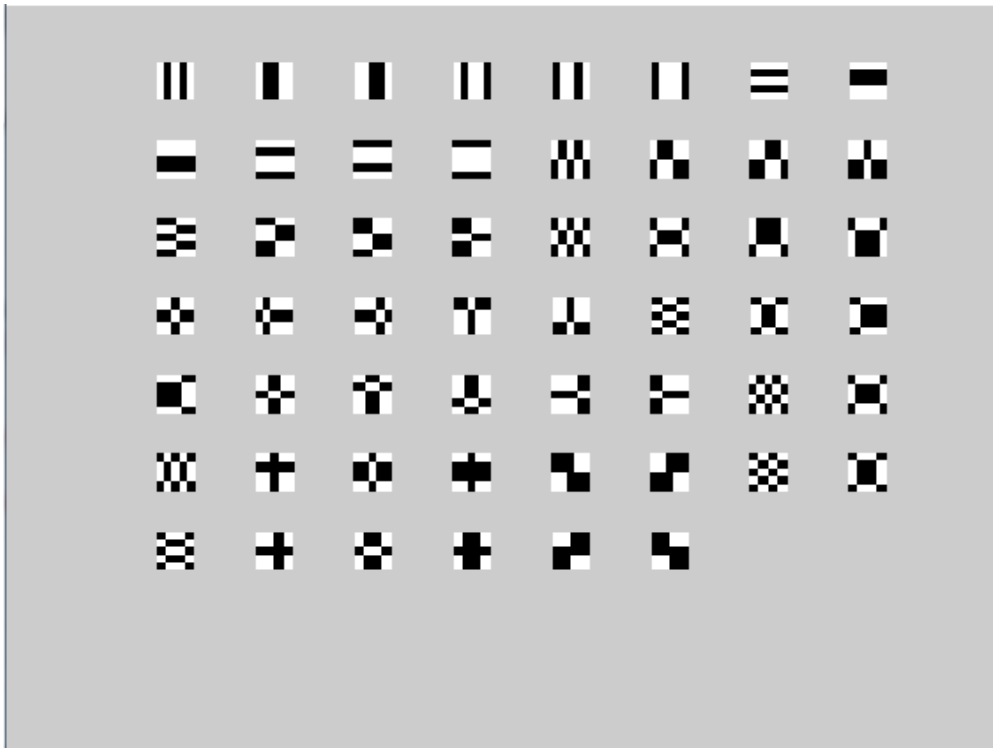


Slika 10 Prikaz koordinata značajke

U prvom redu nalazi se ime značajke, te broj pravokutnih dijelova od kojih se značajka sastoji. Svaki sljedeći redak definira jedan pravokutni dio. U ovom slučaju: prvi pravokutni dio kreće od koordinate (0,0) u gornjem lijevom kutu, pa do koordinate (0.5,0.5) u donjem desnom kutu. Jedinica na kraju retka signalizira da sumu unutar te regije treba zbrojiti, dok minus jedan signalizira da sumu te pravokutne regije treba oduzeti. Uz poštovanje gore navedenih pravila, korisnik slobodno može definirati dodatne značajke.



Slika 11 Prvi set značajki



Slika 12 Drugi set značajki

Kada su sve značajke izračunate šalju se na usporedbu metodama koje izračunavaju Euklidsku udaljenost, kosinus sličnost i Pearsonov koeficijent korelacije. Izvorni kôd metoda koje izračuvaju korelaciju nalazi se u dodatku C.

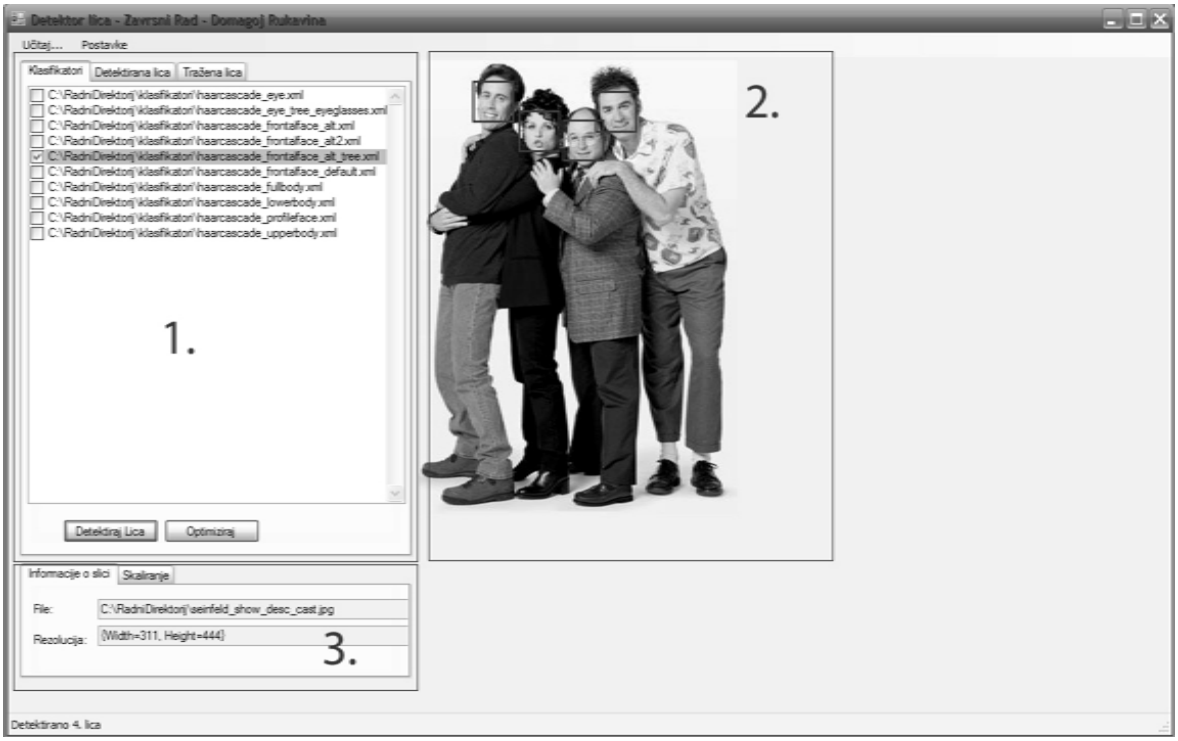
Slika 13 Primjer prolaska različitih značajki preko lica

4.4. Upute za korištenje programa

Za početak, potrebno je kopirati „RadniDirektorij“ u startni direktorij tvrdog diska, npr. C:\, kako bi sve datoteke potrebne za rad bile na stazi sa kratkim imenom.

Ponekad se zbog predugačke staze program počne neuobičajeno ponašati. Nakon ovog koraka staza do radnog direktorija trebala bi biti *C:\RadniDirektorij*.

Pri pokretanju programa pojavljuje se grafičko sučelje.



Slika 14 Grafičko sučelje programa

Kada je program uspješno pokrenut, potrebno je izvršiti učitavanja koja su potrebna za rad progama. U izborniku se nalaze dvije stavke „Učitaj“ i „Postavke“. Klikom na izbornik „Učitaj“ spušta pojavljuju se mogućnosti učitavanja. Prvo je potrebno učitati klasifikatore klikom na odgovarajuću stavku. Klasifikatori se nalaze u radnom direktoriju(*C:\RadniDirektorij\klasifikatori*). Nakon uspješnog učitavanja lista klasifikatora pojavljuje se u kartici klasifikatori, u regiji označenoj brojkom 1 na slici 13.

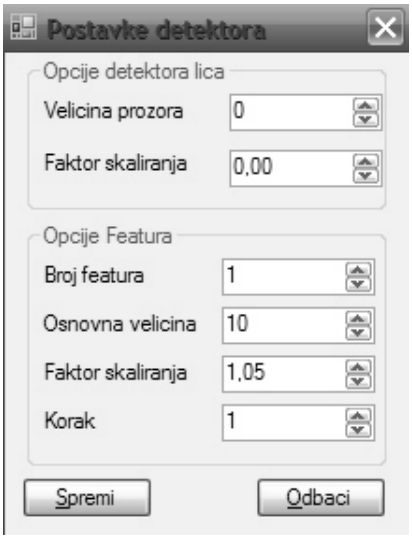
Nakon toga potrebno je učitati radnu sliku. To je slika nad kojom će se detektirati i prepoznavati lica. Poželjno je sve slike za testiranje staviti u RadniDirektorij. Nakon uspješnog učitavanja slika se pojavljuje u regiji broj 2 na slici 13. U regiji 3. na slici 13 prikazane su neke informacije o slici, a sliku je moguće i po potrebi skalirati.

Potom je potrebno izvršiti učitavanje značajki klikom na odgovarajuću stavku u izborniku „Učitaj“. Stavke se nalaze u *C:\RadniDirektorij\znacajke*. Prije učitavanja

datoteke sa značajkama moguće je odrediti koliko da se značajki učita od maksimalnih 115 klikom na izbornik „Postavke“ i podešavanjem opcije „Broj značajki“. Nakon uspješnog učitavanja u statusnoj traci ispisuje se broj uspješno učitanih značajki.

Naposljetku je potrebno učitati direktorij u kojem se nalaze lica koja tražimo. Neka preddefinirana lica se već nalaze u C:\RadniDirektorij\trazenaLica.

Postavke. Tijekom korištenja programa moguće je mijenjati neke parametre. Klikom na „Postavke“ pojavljuje se izbornik (Slika 14).



Slika 15 Grafičko sučelje postavki

Moguće je definirati veličinu detektora i faktor skaliranja detektora lica. Druga grupa postavki definira, redom, broj značajki koje se učitavaju iz datoteke *features.txt,* osnovnu veličinu značajke u pikselima, faktor skaliranja značajke i korak s kojim značajka „putuje“ po slici.

Rad s programom. Jednom kada smo učitali sve što je potrebno u kartici „Klasifikatori“ moguće je označiti proizvoljnu kombinaciju klasifikatora. Program će detektirati lica za svaki klasifikator i vratiti ukupnu listu detektiranih lica. Detekcija lica pokreće se gumbom „Detektiraj“. Gumb „Optimiziraj“ iz liste izbacuje one pravokutnike koji se nalaze unutar drugih pravokutnika nakon detekcije lica.

Nakon detekcije slike detektiranih lica pohranjuju se u direktorij C:\RadniDirektorij\detektiranaLica, te se pojavljuju u listi detektiranih lica na kartici „Detektirana lica“. Dvostrukim klikom na neku stavku moguće je vidjeti sliku detektiranog lica.

Potom se prebacujemo na karticu tražena lica. Potrebno je označiti jedno lice iz liste i potom kliknuti usporedi. Program ispisuje rezultate usporedbe u toj istoj kartici.

Tablica 7 Primjer ispisa nakon usporedbe

Ispis je oblika: „Usporedio lice: C:\RadniDirektorij\trazenaLica\jerry.jpg sa licem: C:\RadniDirektorij\detektiranaLica\detektirano_Lice1.jpg Euklidska udaljenost: 651235,886029018 Cos slicnost: 0,985192698879252 Pearson korelacija: 0,984429902539338

Dobivene podatke treba interpretirati kako je opisano u teorijskom dijelu.

5. Testiranje i rezultati

Testiranje programa napravljeno je na različitim grupnim slikama, te sa različitim traženim slikama.

Primjer Seinfeld

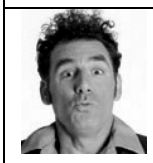
Korištenjem originalnih postavki i označavanjem klasifikatora *haarcascade_frontalface_alt_tree.xml* dobivaju se rezultati na slici 15. Detekcija radi dobro i gotovo se uvijek „igranjem“ sa parametrima može dobiti slika sa minimalnim brojem lažnih detekcija.

Prepoznavanje je napravljeno na nizu različitih ulaznih podataka. Rezultati prepoznavanja nalaze se u tablici 8. U stupcima su lica koja je našao detektor, a u horizontalnom retku su zadane tražene slike.

Slika 16 Detektirana lica

Tablica 8. Rezultati testirana na primjeru 1. U stupcima se nalaze detektirana lica, a svaki redak je rezultat usporedbe traženog lica sa detektiranim licima. Najbolje podudaranje je boldano svaki put.

				
	Euklidska udaljenost: 1215344,79314637 Cos slicnost: 0,956115488578606 Pearson korelacija: 0,954056750394117	Euklidska udaljenost: 651235,886029018 Cos slicnost: 0,985192698879252 Pearson korelacija: 0,984429902539338	Euklidska udaljenost: 1533702,1775247 Cos slicnost: 0,919769249149264 Pearson korelacija: 0,916754809195486	**Euklidska udaljenost: 0 Cos slicnost: 1 Pearson korelacija: 1**

	Euklidska udaljenost: 1167408,67208532 Cos slicnost: 0,973904986502014 Pearson korelacija: 0,972699130812638	Euklidska udaljenost: 852451,153056878 Cos slicnost: 0,979794129590291 Pearson korelacija: 0,978803765910288	Euklidska udaljenost: 1416361,14902944 Cos slicnost: 0,933402925237788 Pearson korelacija: 0,930934311445785	**Euklidska udaljenost: 717438,074707497 Cos slicnost: 0,988594036110422P earson korelacija: 0,988007524308385**
	Euklidska udaljenost: 2494281,74491576 Cos slicnost: 0,837124903973675 Pearson korelacija: 0,829388376219617	Euklidska udaljenost: 2410126,89178558 Cos slicnost: 0,845934344308959 Pearson korelacija: 0,837827164886803	Euklidska udaljenost: 2766068,61130811 Cos slicnost: 0,79312478428126 Pearson korelacija: 0,784500850990653	**Euklidska udaljenost: 2325079,08108262 Cos slicnost: 0,857680037650857 Pearson korelacija: 0,850436367184047**
	Euklidska udaljenost: 1899724,45248568 Cos slicnost: 0,956052403617987 Pearson korelacija: 0,954151303506112	Euklidska udaljenost: 965386,269908579 Cos slicnost: 0,986273534622312 Pearson korelacija: 0,985631450737346	Euklidska udaljenost: 2332471,18370881 Cos slicnost: 0,922692900263603 Pearson korelacija: 0,91999383566685	**Euklidska udaljenost: 166719,624831632 Cos slicnost: 0,99964430787345 Pearson korelacija: 0,999627262289537**
	Euklidska udaljenost: 286930925,930561 Cos slicnost: 0,910463993404962 Pearson korelacija: 0,909111408092558	Euklidska udaljenost: 406497728,735668 Cos slicnost: 0,824730396131889 Pearson korelacija: 0,822274171632353	Euklidska udaljenost: 375960523,030106 Cos slicnost: 0,850996121041494 Pearson korelacija: 0,849093385068564	**Euklidska udaljenost: 208197952,810401 Cos slicnost: 0,957357083092943 Pearson korelacija: 0,956656200900599**
	Euklidska udaljenost: 37711888,9467274 Cos slicnost: 0,896807870778518 Pearson korelacija: 0,894825810481395	Euklidska udaljenost: 37952729,3022454 Cos slicnost: 0,88529884999743 Pearson korelacija: 0,883066824349291	Euklidska udaljenost: 29822918,6814296 Cos slicnost: 0,938111328470063 Pearson korelacija: 0,9370467834449	**Euklidska udaljenost: 26411507,7159195 Cos slicnost: 0,939964657216245 Pearson korelacija: 0,938948206134928**
	**Euklidska udaljenost: 8942201,56246055 Cos slicnost: 0,954408080131891 Pearson korelacija: 0,952957268601033**	Euklidska udaljenost: 10684917,1075826 Cos slicnost: 0,932719676527854 Pearson korelacija: 0,930540851451274	Euklidska udaljenost: 9513475,93274824 Cos slicnost: 0,947816951541693 Pearson korelacija: 0,946195215544153	Euklidska udaljenost: 9557289,97142014 Cos slicnost: 0,94683589380381 Pearson korelacija: 0,945163189639494
	Euklidska udaljenost: 18647113,2243685 Cos slicnost: 0,975828273140745 Pearson korelacija: 0,975221430371892	**Euklidska udaljenost: 13751310,52775 Cos slicnost: 0,986963543773642 Pearson korelacija: 0,986640823284239**	Euklidska udaljenost: 29293024,540495 Cos slicnost: 0,939935565656101 Pearson korelacija: 0,938397088037895	Euklidska udaljenost: 22260726,1052047 Cos slicnost: 0,966108610036148 Pearson korelacija: 0,965237835046522
	Euklidska udaljenost: 58026725,1297108 Cos slicnost: 0,781938656278296 Pearson korelacija: 0,776307648720725	Euklidska udaljenost: 63416227,7042975 Cos slicnost: 0,731498628002202 Pearson korelacija: 0,724587858709196	**Euklidska udaljenost: 51193180,9703324 Cos slicnost: 0,829932902381982 Pearson korelacija: 0,82547741739634**	Euklidska udaljenost: 53215736,2639278 Cos slicnost: 0,808624659430153 Pearson korelacija: 0,803591279552332

Iz ovih podataka vidljivo je da algoritam gotovo uvijek, uspijeva ispravno prepoznati najsličnije lice. Pri prepoznavanju u tablici 8. korišteno je samo devet značajki od mogućih sto petnaest. Međutim, pokazalo se da broj značajki ne utječe značajno na krajnji rezultat. (Tablica 9). Ideja algoritma nije bila da se postavi neka granica sličnosti i da se na taj način „filtriraju“ slična lica od različitih. Jednostavno uvijek ćemo dobiti neku mjeru sličnosti između traženog i detektiranog lica. Stoga ovaj algoritam zapravo ne odgovara na pitanje „Postoji li

dano lice na slici?“, nego nam daje odgovor na pitanje „Koliko su lica na slici slična zadanom licu?“.

Druga bitna stvar koja se može primijetiti je da su vrijednosti korelacije vrlo bliske. Zašto je tome tako odgovorili su već i Viola i Jones. Činjenica je naime, da samo računanje značajki nad licima daje vrlo slične vektore vrijednosti, tj. mogli bismo reći: „Sva lica su slična“. Kako bi bolje razdvojili isto lice od različitih lica, potrebno je iz skupa značajki odabrati one značajke koje će najbolje razdvajati isto lice od drugih lica. Viola i Jones su to učinili modificiranim AdaBoostom, ali mogu se primijeniti i druge tehnike učenja.

U nastavku je dana tablica dobivenih vrijednosti za četiri primjera iz prošle tablice, uz računanje pedeset značajki. Računanje svih 115 značajki redovno neslavno završi ranije jer se popuni radna memorija računala na testnom računalu.

Tablica 9 Rezultati usporedbe korištenjem 50 značajki

				
	Euklidska udaljenost: 48723829,7304037 Cos slicnost: 0,869259273481268 Pearson korelacija: 0,868802515645859	Euklidska udaljenost: 45702639,5535096 Cos slicnost: 0,871629071364852 Pearson korelacija: 0,871499528519092	Euklidska udaljenost: 38353337,5195336 Cos slicnost: 0,923295213820703 Pearson korelacija: 0,922888597104299	**Euklidska udaljenost: 32761811,8216337 Cos slicnost: 0,932085787084262 Pearson korelacija: 0,931746599377225**
	**Euklidska udaljenost: 10950887,1635222 Cos slicnost: 0,950054373874602 Pearson korelacija: 0,949863807681032**	Euklidska udaljenost: 12664909,9802917 Cos slicnost: 0,930314438752967 Pearson korelacija: 0,930305882158264	Euklidska udaljenost: 11034938,0467609 Cos slicnost: 0,948892691860885 Pearson korelacija: 0,948632430646195	Euklidska udaljenost: 12181396,6824927 Cos slicnost: 0,936148313539979 Pearson korelacija: 0,935829185998457
	Euklidska udaljenost: 24233146,1436869 Cos slicnost: 0,969440083813399 Pearson korelacija: 0,96934187600583	**Euklidska udaljenost: 17610345,7750736 Cos slicnost: 0,983985100331786 Pearson korelacija: 0,983986384354917**	Euklidska udaljenost: 34586942,4219533 Cos slicnost: 0,938016605192766 Pearson korelacija: 0,937882808339087	Euklidska udaljenost: 27411582,5284148 Cos slicnost: 0,959930194936343 Pearson korelacija: 0,95985689054135
	Euklidska udaljenost: 64586108,1389443 Cos slicnost: 0,800973366873093 Pearson korelacija: 0,800216121797208	Euklidska udaljenost: 71257226,3850292 Cos slicnost: 0,747382673945848 Pearson korelacija: 0,746784204870083	**Euklidska udaljenost: 56293090,0453395 Cos slicnost: 0,849736784114735 Pearson korelacija: 0,848933784827968**	Euklidska udaljenost: 62655368,3248588 Cos slicnost: 0,805032581854449 Pearson korelacija: 0,804036280159354

Za kraj, evo što se dobiva kad algoritmu prezentiramo lice koje nije na slici. Treba imati na umu što algoritam radi, dobivamo mjeru sličnosti nepoznatog lica sa detektiranim licima.

Tablica 10 Rezultati usporedbe lica kojeg nema na slici sa danim licima. Rezultati nisu krivi, već prikazuju sličnost traženog lica sa detektiranim licima.

				
	Euklidska udaljenost: 17072489,7155488 Cos slicnost: 0,871253681478114 Pearson korelacija: 0,870809697040444	Euklidska udaljenost: 18854719,2217804 Cos slicnost: 0,824207415201383 Pearson korelacija: 0,82407198752772	Euklidska udaljenost: 10827066,4931543 Cos slicnost: 0,951468898575385 Pearson korelacija: 0,951162016988123	Euklidska udaljenost: 14820516,2803255 Cos slicnost: 0,885765939905676 Pearson korelacija: 0,885201874208243

Podaci pokazuju da su Newman i Kramer najsličniji što bi moglo objasniti zašto su najbolji prijatelji.

6. Zaključak

Rezultati dobiveni ovim pristupom su očekivani, ali svejedno tu se ne bi trebalo zaustaviti. Kako bi se ispravila pogreška nedovoljnog razlučivanja različitih lica svakako bi u budućem razvoju trebalo uključiti komponentu učenja dobrih značajki kako bi naposljetku ista lica bila bolje razdvojena od drugih lica. Kako nastaviti dalje? Mogućnosti su različite, implementacija modificiranog AdaBoost algoritma prema uputama Viole i Jonesa ili kreiranje umjetne neuronske mreže ili možda nekog trećeg pristupa. U svakom slučaju, program je spreman za dodatna poboljšanja i optimizacije kojih će sigurno i biti.

Želja je autora da ovaj program postane koristan alat koji bi svoju primjenu mogao potražiti u identifikaciji korisnika računala, naprednom pretraživanju slika koje bi znalo odgovoriti postoji li osoba na danoj slici, automatsko označavanje osoba na slici itd.

7. Literatura

[1] Viola P., Jones M. ; *Robust Real-Time Object Detection;* International Journal of Computer Vision, 57/2, 2004, str. 137–154

[2] *Computer Vision*, http://en.wikipedia.org/wiki/Computer_vision, 25.05.2009.

[3] *Haar functions*, http://mathworld.wolfram.com/HaarFunction.html, 30.05.2009.

[4] Bahlmann C. , Zhu Y. , Ramesha V., Pellkofer M. , Koehler T. ; *System for Traffic Sign Detection, Tracking, and Recognition Using Color, Shape, and Motion Information;* Intelligent Vehicles Symposium; 2005; str. 255- 260

[5] *AdaBoost*; 19.12.2008.; *http://en.wikipedia.org/wiki/AdaBoost;* 20.4.2009.

[6] Jensen O.H.; *Implementing the Viola-Jones Face Detection Algorithm;* Kongens Lyngby 2008

[7] Bradski G., Kaehler A. ; *Learning OpenCV ;* prvo izdanje*;* Sebastopol, SAD: O'Reilly Media, 2008

[8] Agam G., *Introduction to programming with OpenCV*, Department of Computer Science 27.1.2006. ; http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html; 21.5.2009.

[9] Viola P., Jones M. ; *Face Recognition Using Boosted Local Features*, Submitted to The IEEE International Conference on Computer Vision 2003, 2003.

[10] *OpenCV with Visual C++ 6.0, 2005 Express, and 2008 Express*, http://opencv.willowgarage.com/wiki/VisualC%2B%2B; 31.5.2009.

[11] *Euclidean distance*, http://en.wikipedia.org/wiki/Euclidean_distance, 1.06.2009.

[12] *Wolfram World - Correlation Coefficient*, http://mathworld.wolfram.com/CorrelationCoefficient.html, 1.06.2009.

8. Sažetak

U radu se razmatra, teorijski i praktično, detekcija zadanog lica u grupnim slikama pomoću Haarovih značajki. Postupak je podijeljen u dvije logičke cjeline: detekcija svih lica na slici i raspoznavanje zadanog lica među detektiranim licima.

U teorijskom dijelu opisuju se ideje i koncepti kojima se detektiraju i raspoznavaju lica, a u praktičnom dijelu opisuje se implementacija tih ideja korištenjem programskih jezika C# i C++.

Za detekciju lica koristi se poznati Viola-Jones algoritam koji se temelji na izračunavanju Haarovih značajki, a potom se vrši raspoznavanje izračunavanjem Haarovih značajki nad detektiranim licima i nad traženim licem te računanjem korelacije između dobivenih vektora vrijednosti.

Ključne riječi: viola jones, detekcija lica, prepoznavanje lica, adaboost, haarove značajke, umjetna inteligencija, računalni vid.

Dodatak A: izvorni kod nadjiLice.cpp

```
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"

#include <iostream>
#include <fstream>
using namespace std;

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

/*Postavljeni su default parametri*/

// Kaskade
const char* cascade_name = "C:/Program
Files/OpenCV/data/haarcascades/haarcascade_frontalface_alt_tree.xml";
// Separator tekstualne datoteke
const char* separator = "----------";
// Ime rezultantne datoteke
const char* FILENAME;
// Faktor skaliranja
double scaling_factor = 1.25;
// Velicina osnovnog detektora
int detector_size = 24;
// TestMode
int testMode=0;

// Function prototype for detecting and drawing an object from an image
void detect_and_draw( IplImage* image );

// Main function, defines the entry point for the program.
int main( int argc, char* argv[] )
{
	//
	// …
	// RAZNE PROVJERE PARAMETARA SU IZREZANE
	// …

	// Provijeri da li je prvi zadani parametar kaskada
	cascade_name=argv[1];
	// Provijeri unos velicine detektora
	detector_size=atoi(argv[2]);
	scaling_factor=strtod(argv[3],NULL);
	// provjeri postojanje ulazne datoteke
	FILENAME = argv[5];

	// Create an image
	IplImage *img = cvLoadImage( argv[4] ,1 );

	if (img == NULL)
```

```
      {
            printf("Nije zadana ispravna slika!");
            return -4;
      }
      // Call the function to detect and draw the face positions
      detect_and_draw( img );

      if (testMode)
      cvWaitKey();

    // return 0 to indicate successfull execution of the program
    return 0;
}

// Function to detect and draw any faces that is present in an image
void detect_and_draw( IplImage* img )
{

    // Create memory for calculations
    static CvMemStorage* storage = 0;
    // Create a new Haar classifier
    static CvHaarClassifierCascade* cascade = 0;

    int scale = 1;

    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width/scale,img-
>height/scale), 8, 3 );
      temp = cvCloneImage( img );
    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;

    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

    // Check whether the cascade has loaded successfully. Else report and
error and quit
    if( !cascade )
    {
        fprintf( stderr, "ERROR: Nisam uspio stvoriti kaskadu!\n" );
        return;
    }

    // Allocate the memory storage
    storage = cvCreateMemStorage(0);

    // Create a new named window with title: result
      if (testMode)
    cvNamedWindow( "Rezultat", 1 );

    // Clear the memory storage which was used before
    cvClearMemStorage( storage );

  // Find whether the cascade is loaded, to find the faces. If yes, then:
    if( cascade )
    {
        // There can be more than one face in an image. So create a
growable sequence of faces.
        // Detect the objects and store them in the sequence
            CvSeq* faces = cvHaarDetectObjects( temp,
```

```
      cascade, storage,
      scaling_factor, 2 , CV_HAAR_DO_CANNY_PRUNING,
            cvSize(detector_size,detector_size) );
            // Otvori datoteku za dopisivanje
            ofstream myfile;
            myfile.open (FILENAME,ios::app);

            // Prodji kroz sva lica.
        for( i = 0; i < (faces ? faces->total : 0); i++ )
        {
           //Napravi pravokutnik za svako detektirano lice
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
            // Find the dimensions of the face,and scale it if necessary
            pt1.x = r->x*scale;
            pt2.x = (r->x+r->width)*scale;
            pt1.y = r->y*scale;
            pt2.y = (r->y+r->height)*scale;

                  cout<<separator<<endl;
                  cout<<"L:"<<i<<endl;
                  cout<<pt1.x<<endl;
                  cout<<pt1.y<<endl;
                  cout<<pt2.x<<endl;
                  cout<<pt2.y<<endl;

                  myfile<<separator<<endl;
                  myfile<<"L:"<<i<<endl;
                  myfile<<pt1.x<<endl;
                  myfile<<pt1.y<<endl;
                  myfile<<(r->width)*scale<<endl;
                  myfile<<(r->height)*scale<<endl;

            // Nacrtaj pravokutnik
            cvRectangle( temp, pt1, pt2, CV_RGB(255,0,0), 1, 8, 0 );
        }
            myfile.close();
    }
    // Show the image in the window named "result"
      if (testMode)
      {
            cvShowImage( "Rezultat", temp );
      }
    // Release the temp image created.
    cvReleaseImage( &temp );
      // Wait for user input before quitting the program
      if (testMode)
      {
            cvWaitKey();
      }
      // Release the image
      cvReleaseImage(&img);
      // Destroy the window previously created with filename: "Rezultat"
      if (testMode)
      {
            cvDestroyWindow("Rezultat");
      }
}
```

Dodatak B:računanje integralne slike u C#

```
private void IzracunajIntegralneBGR()
        {
            // zakljucaj sliku da bi pristupao bitovima (vidi link ispod)
            BitmapData bmd = bm.LockBits(
                       new Rectangle( 0, 0, bm.Width,   bm.Height ),
                       System.Drawing.Imaging.ImageLockMode.ReadWrite,
                       bm.PixelFormat );
            // 3 kanala
            int PixelSize = 3;
            double[ , ] sumRedaBlue = new double[ bm.Height, bm.Width ];
            double[ , ] sumRedaRed = new double[ bm.Height, bm.Width ];
            double[ , ] sumRedaGreen = new double[ bm.Height, bm.Width ];
            // objasnjenje koda na
http://www.bobpowell.net/lockingbits.htm ili na stranicama MSDN-a
         // potrebno podesiti opciju "Allow unsafe code" u project
properties
            unsafe
            {
                for ( int x = 0 ; x < bmd.Height ; x++ )
                {
                    byte* row = (byte*)bmd.Scan0 + ( x * bmd.Stride );
                    for ( int y = 0 ; y < bmd.Width ; y++ )
                    {
                        // Kanali slike spremljeni su u slijedu BGR

                        // PLAVI KANAL row[ ( x * PixelSize ) ] = 0;
                        // ZELENI KANAL row[ ( x * PixelSize ) + 1 ] = 0;
                        // CRVENI KANAL row[ ( x * PixelSize ) + 2 ] = 0;
                        // izracun integralne slike prema uputi iz
originalnog viola-jones clanka

                        // Racunam integralnu sliku na Blue kanalu
                        if ( ( y - 1 ) < 0 )
                            sumRedaBlue[x,y] = row[ x * PixelSize ];
                        else
                            sumRedaBlue[x,y] = sumRedaBlue[ x, y - 1 ] +
                            row[ x * PixelSize ];
                        if ( ( x - 1 ) < 0 )
                            integralImageBlue[x,y]=sumRedaBlue[ x, y ];
                        else
                            integralImageBlue[ x, y ] =
                            integralImageBlue[ x - 1, y ] +
                            sumRedaBlue[ x, y ];
                        // Racunam integralnu sliku na Green kanalu
                        if ( ( y - 1 ) < 0 )
                         sumRedaGreen[ x, y ]=row[(x * PixelSize) + 1];
                        else
                         sumRedaGreen[ x, y ] = sumRedaGreen[ x, y - 1 ]
                         + row[ ( x * PixelSize ) + 1 ];
                        if ( ( x - 1 ) < 0 )
                            integralImageGreen[ x, y ]=sumRedaGreen[x,y];
                        else
                            integralImageGreen[ x, y ] =
                            integralImageGreen[ x - 1, y ] +
                            sumRedaGreen[ x, y ];
                        // Racunam integralnu sliku na Red kanalu
                        if ( ( y - 1 ) < 0 )
                            sumRedaRed[ x, y ] = row[(x*PixelSize) + 2 ];
                        else
```

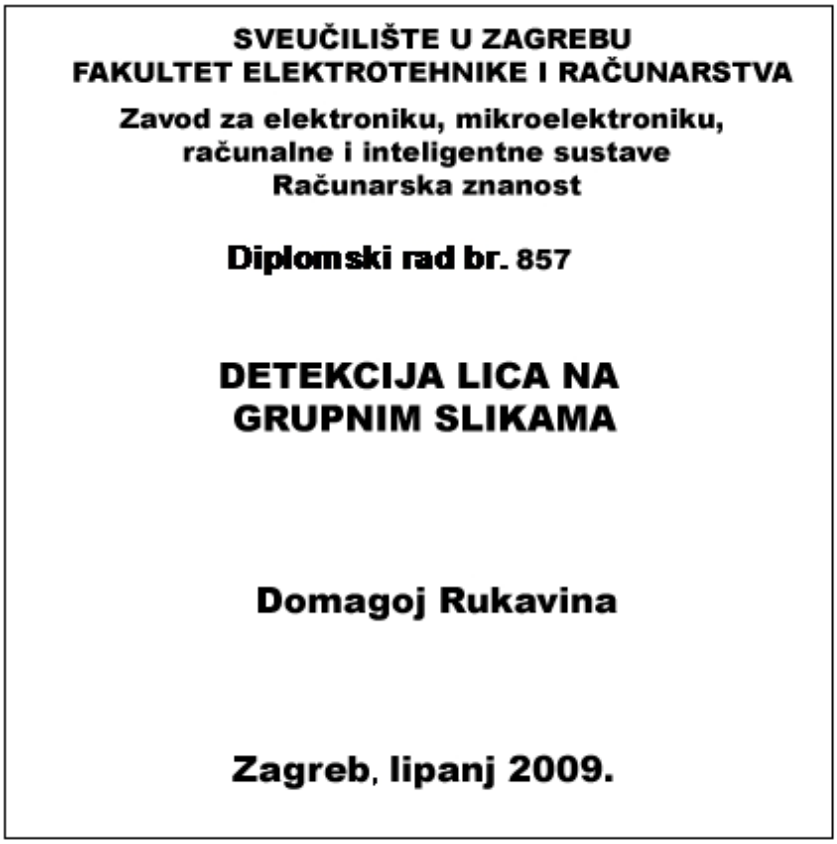
```
                        sumRedaRed[ x, y ] = sumRedaRed[ x, y - 1 ] +
                        row[ ( x * PixelSize ) + 2 ];
                    if ( ( x - 1 ) < 0 )
                        integralImageRed[ x, y ] = sumRedaRed[ x,y ];
                    else
                        integralImageRed[ x, y ] =
                        integralImageRed[ x - 1, y ] +
                        sumRedaRed[ x, y ];
                }
            }
        }
        // pospremi byteove nazad u sliku
        bm.UnlockBits( bmd );
    }
```

Dodatak C: C# izvorni kod za izračuvanje korelacije

```
double IzracunajCosinusSlicnost(List<double> genFeaturiTrazenoLice,
                                List<double> genFeatureDetLice )
        {
            // nema usporedbe ako duljina vektora nije jednaka
            if ( genFeaturiTrazenoLice.Count != genFeatureDetLice.Count )
                return double.NaN;

            double sumaUmnozak = 0;
            double umnozak = 0;
            for ( int i = 0 ; i < genFeatureDetLice.Count ; i++ )
            {
              umnozak = genFeatureDetLice[i] * genFeaturiTrazenoLice[i];
              sumaUmnozak += umnozak ;
            }
            double sumaX = 0;
            for ( int i = 0 ; i < genFeatureDetLice.Count ; i++ )
            {
                sumaX += Math.Pow( genFeatureDetLice[ i ], 2 );
            }
            double sumaY = 0;
            for ( int i = 0 ; i < genFeaturiTrazenoLice.Count ; i++ )
            {
                sumaY += Math.Pow( genFeaturiTrazenoLice[ i ], 2 );
            }
            double nazivnik = Math.Sqrt( sumaX * sumaY );
            return sumaUmnozak / nazivnik;
        }

double IzracunajEuklidskuUdaljenost( List<double> genFeaturiTrazenoLice,
                                     List<double> genFeatureDetLice )
        {
            if ( genFeaturiTrazenoLice.Count != genFeatureDetLice.Count )
                return double.NaN;

            double razlika = 0;
            double suma = 0;
            for ( int i = 0 ; i < genFeatureDetLice.Count ; i++ )
            {
               razlika = Math.Pow(genFeatureDetLice[i] -
                         genFeaturiTrazenoLice[i],2);
               suma += razlika;
            }
            double udaljenost = Math.Sqrt( suma );
            return udaljenost;
        }

double pearsonCorrelation( double[] x, double[] y, int n )
        {
            double result = 0;
            int i = 0;
            double xmean = 0;
            double ymean = 0;
            double s = 0;
            double xv = 0;
            double yv = 0;
            double t1 = 0;
            double t2 = 0;
```

```
        xv = 0;
        yv = 0;
        if ( n <= 1 )
        {
            result = 0;
            return result;
        }

        //
        // Srednja vrijednost
        //
        xmean = 0;
        ymean = 0;
        for ( i = 0 ; i <= n - 1 ; i++ )
        {
            xmean = xmean + x[ i ];
            ymean = ymean + y[ i ];
        }
        xmean = xmean / n;
        ymean = ymean / n;

        //
        // brojnik i nazivnik
        //
        s = 0;
        xv = 0;
        yv = 0;
        for ( i = 0 ; i <= n - 1 ; i++ )
        {
            t1 = x[ i ] - xmean;
            t2 = y[ i ] - ymean;
            xv = xv + Math.Pow( t1,2 );
            yv = yv + Math.Pow( t2,2 );
            s = s + t1 * t2;
        }
        if ( xv == 0 | yv == 0 )
        {
            result = 0;
        }
        else
        {
            result = s / ( Math.Sqrt( xv ) * Math.Sqrt( yv ) );
        }
        return result;
    }
```

Dodatak D:Omot za CD



RZ - Diplomski rad br. 857 – Domaqoi Rukavina

RZ - Diplomski rad br. 857 – Domaqoi Rukavina