SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 806

PROGRAMSKA IMPLEMENTACIJA PRONALAŽENJA OBJEKATA KASKADOM BOOSTANIH HAAROVIH KLASIFIKATORA

Tomislav Babić

Zagreb, srpanj 2009.

Sadržaj

1. Uvod

Prepoznavanje prometnih znakova je od sve veće važnosti, kako u svijetu, tako i u Hrvatskoj. Vozač u automobilu može previdjeti prometni znak zbog umora ili bilo kojeg od drugih razloga. Stoga je cilj ugraditi u vozila sustave temeljene na računalnom vidu koji bi mogli detektirati i raspoznati prometne znakove u prihvatljivom vremenskom intervalu. Takvi sustavi bi se mogli koristiti kao potpora vozaču pružajući mu informacije kojih možda nije bio svjestan. Rezultat korištenja takvih sustava bio bi povećanje sigurnosti u prometu.

Jedna od metoda za detekciju znakova koja se pokazala poprilično uspješnom razvili su Paul Viola i Michael J. Jones. Metoda se temelji na kaskadi boostanih Haarovih klasifikatora [1,2,3]. Prvi korak u korištenju te metode jest učenje nad skupom ručno označenih znakova i pozadinskih slika. Tijekom procesa učenja se stvara već spomenuta kaskada klasifikatora. Drugi korak je pronalaženje znakova korištenjem jedne takve kaskade.

U sklopu ovog završnog rada proučit će se metoda za pronalaženje objekata. Također će se proučiti postojeće rješenje Rainera Lienharta koje se nalazi u sklopu programskog okruženja OpenCV. Naposljetku će se razviti vlastita programska komponenta integrirana u programsko okruženje cvsh.

2. Pronalaženje objekata kaskadom boostanih Haarovih klasifikatora

Paul Viola i Michael J. Jones su osmislili metodu za brzo pronalaženje objekata na slici korištenjem kaskade boostanih (pojačanih) Haarovih klasifikatora. Primarna svrha metode bila je detekcija lica, no budući da koristi algoritme strojnog učenja, metoda se može primijeniti za detekciju bilo koje vrste objekata, uključujući i prometne znakove. S tipičnim parametrima algoritam Viole i Jonesa pronalazi objekte u pronalazi petnaest slika po sekundi, s tada konvencionalnim procesorom Pentium III takta na 700 Mhz, gdje su slike bile dimenzija 384x288 [1]. S promjenom parametara može doći i do značajnih promjena u brzini obrade.

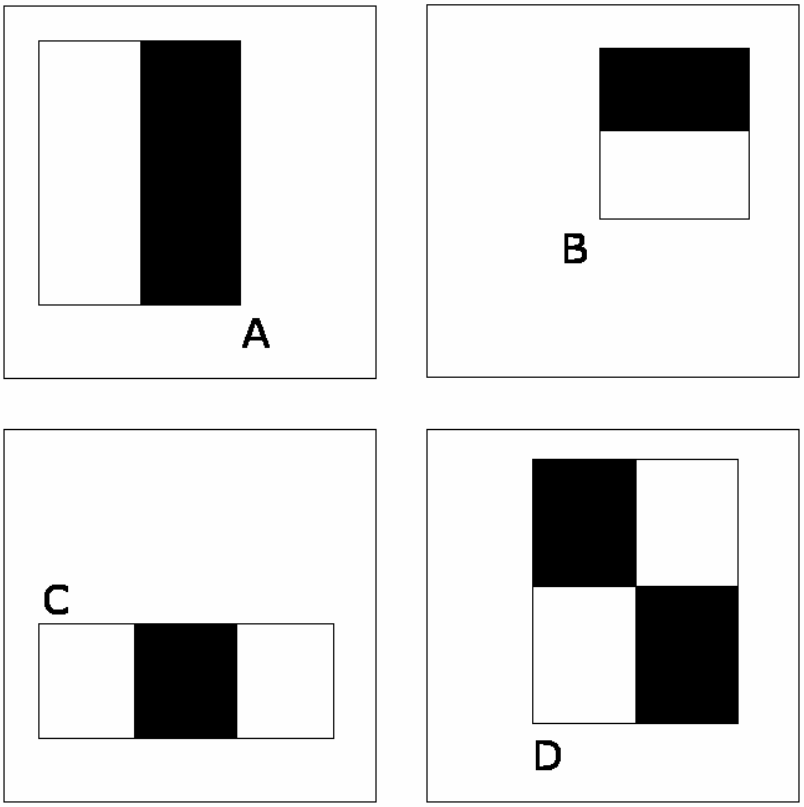
2.1. Koncept detekcije objekata binarnom klasifikacijom

Jedan od novijih pristupa detekciji objekata temelji se na binarnoj klasifikaciji. U postupku detekcije koristi se okno koje više puta prolazi kroz cijelu sliku. Na svakoj poziciji okna primjenjuje se klasifikator, te se detekcija prijavljuje na svim položajima gdje je klasifikator dao pozitivan odgovor.

Ovisno o ulaznim parametrima, odnosno dijelu slike koji se ispituje, klasifikator može dati pozitivan ili negativan odgovor. Ukoliko je izlaz negativan, tada klasifikator javlja da traženi objekt nije u oknu, odnosno obrnuto ako je izlaz pozitivan. Takav klasifikator s dva izlaza se naziva još i binarni klasifikator.

2.2. Haarove značajke i integralna slika

Prilikom detekcije objekata većini ljudi bi najvjerojatnije palo na pamet uspoređivati vrijednosti piksela s pikselima znakova iz skupa za učenje. Međutim, takav pristup pokazao se procesno izuzetno zahtjevnim. Umjesto toga, Viola i Jones su odlučili koristiti tri tipa Haarovih značajki. Haarove značajke se računaju kao razlika suma piksela unutar različitih pravokutnih područja slike [4], kao što je ilustrirano na slici 1.



Slika 1. Četiri osnovna tipa značajki Viole i Jonesa. Sve ostale značajke u detekcijskom oknu dobivamo translacijom i skaliranjem osnovnih značajki.

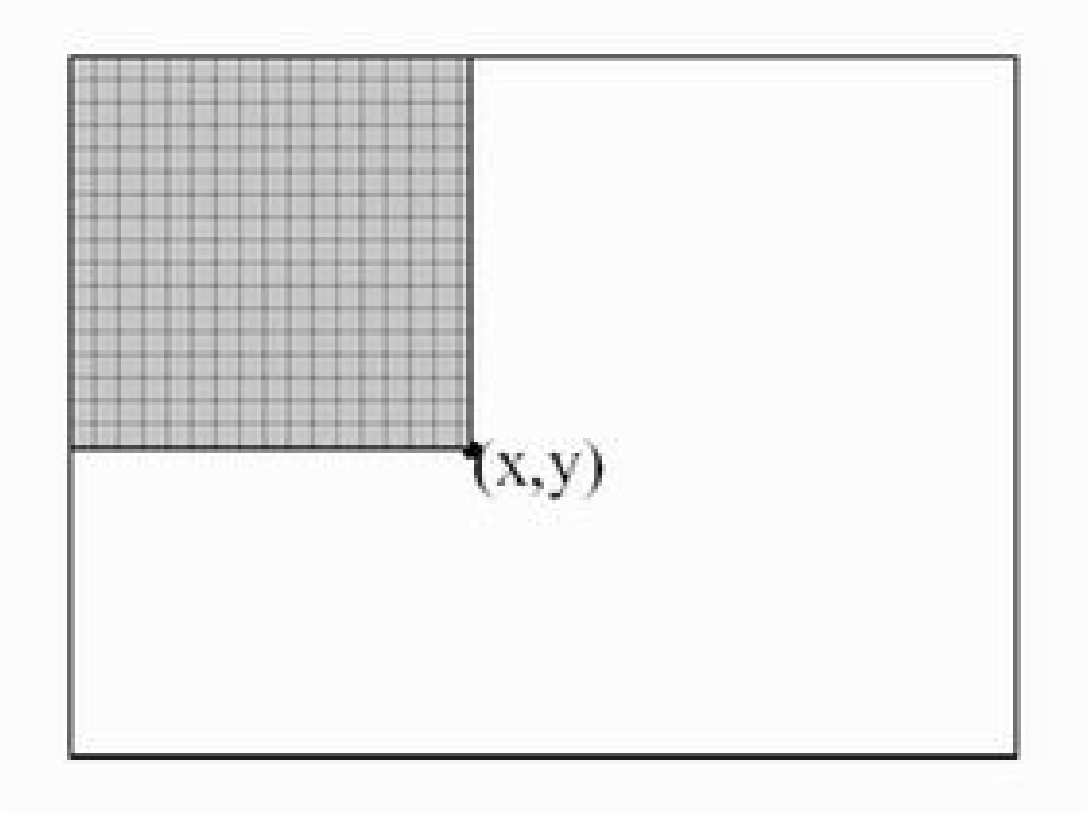
Značajke sa dva pravokutnika (značajke A i B na slici Slika *1*) računaju se kao razlika između sume piksela u bijelom i sume piksela u tamnom pravokutniku. Značajka s tri pravokutnika (značajka C na slici Slika *1*) se računa kao zbroj suma piksela dvaju vanjskih pravokutnika umanjena za sumu piksela unutrašnjeg pravokutnika. Naposljetku, značajka s četiri pravokutnika (značajka D na slici 1) se računa kao razlika između suma tamnih i bijelih dijagonalno posloženih pravokutnika.

Naravno, računanje sume piksela u svakom pravokutniku je vremenski izuzetno računski zahtjevan proces, pogotovo ako se to mora učiniti mnogo puta za pregled jedne slike. Za ubrzanje tog postupka koristi se tzv. integralna slika [1] kao što je ilustrirano na slici 2. Svaki piksel u integralnoj slici S_i se dobije tako da se izračuna zbroj svih piksela originalne slike S lijevo i iznad traženog piksela, uključujući i traženi:

$$S_i(x,y) = \sum_{x' \le x, y' \le y} S(x',y') \tag{1}$$

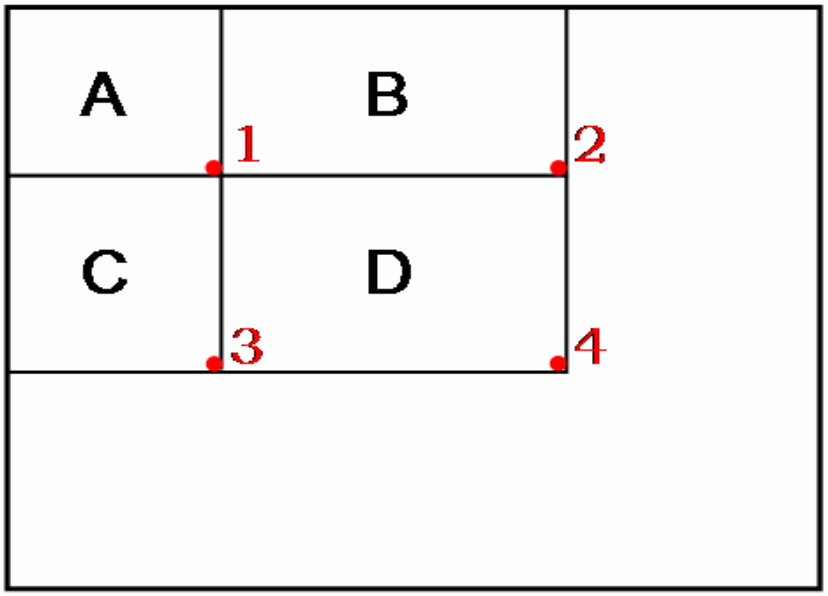
Ipak i postupak dobivanja integralne slike se može ubrzati i izračunati u samo jednom prolazu koristeći sumu S_s svih piksela u stupcu iznad traženog piksela:

$$S_s(x,y) = S_s(x,y-1) + S(x,y)$$
$$S_i(x,y) = S_s(x,y) + S_i(x-1,y) \quad (2)$$



Slika 2. Piksel na koordinatama x i y računa se kao suma piksela u označenom pravokutniku

Korištenjem integralne slike može se vrlo brzo i jednostavno izračunati suma piksela unutar bilo kojeg pravokutnika u originalnoj slici kao što je ilustrirano na slici 3. Za izračun vrijednosti Haarove značajke s dva pravokutnika, dovoljno je pristupiti integralnoj slici šest puta, za značajku sa tri pravokutnika vrijednost se može izračunati s osam pristupa integralnoj slici te za značajku s četiri pravokutnika dovoljno je pristupiti devet puta.



Slika 3. Suma piksela u pravokutniku D na originalnoj slici može se izračunati kao: D = S_{i1} – S_{i2} – S_{i3} + S_{i4}, gdje su S_{i1}, S_{i2}, S_{i3} i S_{i4} vrijednosti piksela u integralnoj slici S_i na označenim lokacijama. To se može lako pokazati uvrštavanjem: S_{i1} = A, S_{i1} = A + B, S_{i1} = A + C, S_{i1} = A + B + C + D.

2.3. Boostani Haarovi klasifikatori

Binarni Haarov klasifikator H_i dobivamo usporedom vrijednosti značajke h_i s pragom θ_i. Ako je značajka veća od praga, tada klasifikator daje pozitivan odgovor *a* odnosno negativan odgovor *b* u obrnutom slučaju. Dakle, Haarov klasifikator može se predstaviti sljedećim izrazom:

$$H_i = \begin{cases} a & \text{ako } h_i \geq \theta_i \\ b & \text{inač e} \end{cases} \tag{3}$$

Međutim iz oblika Haarovih značajki (slika 1), može se naslutiti da jedan takav Haarov klasifikator ne bi bio izrazito uspješan u detekciji objekata.Takav klasifikator naziva se slabim klasifikatorom.

Iz toga slijedi zaključak da je očito nužno kombinirati više značajki kako bi postupak detekcije bio zadovoljavajuć. Međutim, broj značajki za okno od 24x24 piksela iznosi oko 45 000 [5], stoga bi bilo neprimjereno i skupo ispitati ih sve u postupku detekcije. U tu svrhu koristi se boostanje odnosno meta-algoritam strojnog učenja koji iz skupa slabih klasifikatra gradi jedan snažni klasifikator [6].

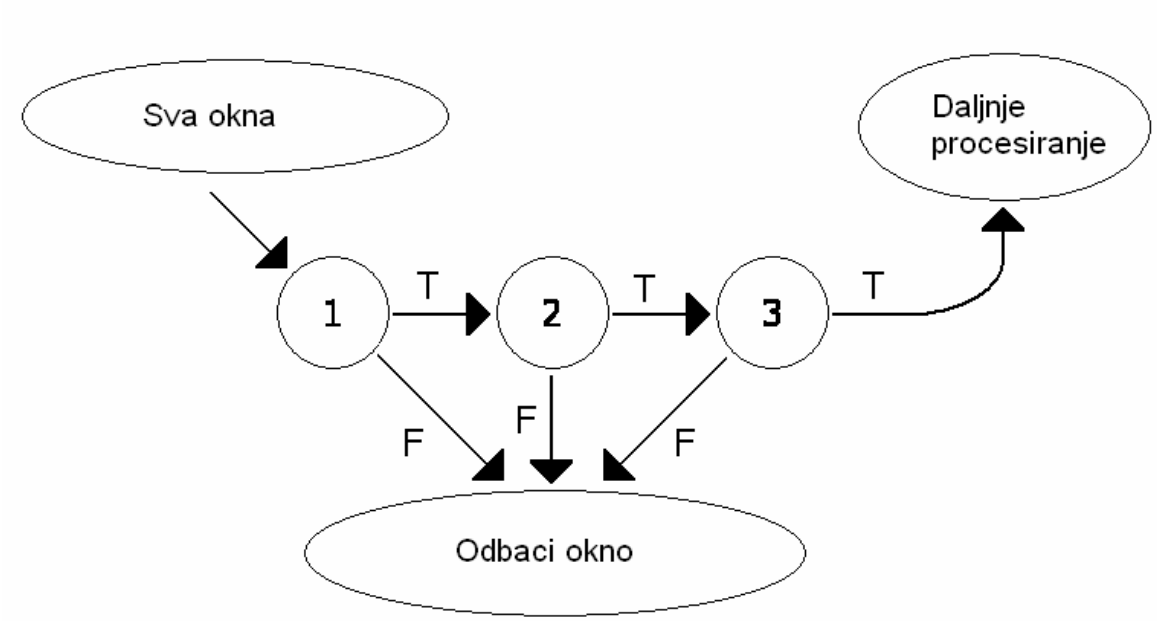
Tijekom procesa detekcije svaki slabi klasifikator će pridonositi pozitivno u slučaju kada detektira traženi objekt, odnosno negativno u obrnutom slučaju. Na temelju doprinosa svih slabih klasifikatora, boostani klasifikator može detektirati objekt u oknu ili javiti da ga nema. Ako *Hi* predstavlja Haarov slabi klasifikator, t_j prag boostanog klasifikatora dobiven tijekom postupka učenja, *Bj* boostani snažni klasifikator, a *n* broj slabih klasifikatora koji čine snažni klasifikator, boostani klasifikator se može opisati sljedećom formulom:

$$B_j = \begin{cases} 1 & \text{ako} \sum_{i=1}^{n} H_{ji} > t_j \\ 0 & \text{inače} \end{cases} \quad (4)$$

2.4. Ulančavanje boostanih klasifikatora

Pokazuje se da bi se boostani klasifikator koji bi bio prikladan za kvalitetne postupke detekcije morao sastojati od iznimno velikog broja značajki. Međutim, obrada tako velikog broja značajki za svaku poziciju okna bi uzela puno vremena. Valja primijetiti da na većini lokacija unutar slike nema traženih objekata. Stoga je bolje rješenje koristiti više boostanih klasifikatora koji će imati visoki postotak eliminacije lokacija u kojima se ne nalaze traženi objekti. Ulančavanjem takvih boostanih klasifikatora u kasakadu postupak detekcije se može znatno ubrzati.

Ako boostani klasifikator u i-tom segmentu lanca zadovoljava prag to znači da je traženi objekt možda u oknu, te se nastavlja s evaluacijom u segmentu i+1. Ukoliko prag nije zadovoljen, ne pozivaju se daljnji klasifikatori, već se ustanovljuje da traženi objekt nije u oknu te se ono pomiče na novu poziciju. Takav "serijski" spoj klasifikatora se naziva kaskada (slika 4).



Slika 4. Arhitektura kaskade

Radi ubrzavanja performansi cilj je da se kroz cijelu kaskadu prođe na što manje lokacija slike, stoga će snažni klasifikatori biti posloženi po postotku uspješnosti eliminacije okna u kojima nema traženog objekta. Tako će prvi klasifikator biti onaj koji ima najveći uspjeh u tome.

Što je kaskada dublja, odnosno ima više klasifikatora u sebi, to će i imati manje pogrešnih detekcija. Neka je *F* procjena pogrešnih detekcija za kaskadu, *K* broj klasifikatora u kaskadi, a f_i procjena procjena pogrešnih detekcija za pojedini klasifikator, tada vrijedi:

$$F = \prod_{i=1}^{K} f_i \tag{5}$$

Isto tako će s porastom dubine kaskade i postotak promašenih detekcija biti nešto veći. Neka je *D* procjena točno detektiranih objekata za cijelu kaskadu, *K* broj klasifikatora, a d_i procjena detektiranih objekata za pojedini klasifikator, tada vrijedi:

$$D = \prod_{i=1}^{K} d_i \tag{6}$$

2.5. Normalizacija detekcijskog okna

Važno je napomenuti da se prilikom postupka učenja provodi normalizacija varijancom, stoga je to isto nužno primijeniti u postupku detekcije [1]. Varijanca odnosno standardna devijacija se može brzo izračunati pomoću obične integralne slike i integralne slike kvadriranih slikovnih elemenata Ako je *m* srednja vrijednost (engl. mean), p_{ij} vrijednost piksela unutar okna, a *N* ukupan broj piksela u oknu, standardna devijacija σ se može odrediti kao:

$$\sigma^2 = m^2 - \frac{1}{N}\sum_{i,j} p_{ij}^2 \tag{7}$$

Ako *w* i *h* predstavljaju širinu odnosno visinu okna tada se srednja vrijednost piksela može lako izračunati pomoću integralne slike S_i:

$$m = \frac{S_i(x,y) - S_i(x+w,y) - S_i(x,y+h) + S_i(x,y)}{(x+w)\cdot(y+h)} \tag{8}$$

Suma kvadratnih vrijednosti piksela se može izračunati pomoću integralne slike kvadriranih slikovnih elemenata *Sq*:

$$\sum_{i,j} p_{ij}^2 = S_q(x,y) - S_q(x+w,y) - S_q(x,y+h) + S_q(x,y) \tag{9}$$

$$S_q(x,y) = \sum_{x'\le x, y'\le y} S(x',y')^2 \tag{10}$$

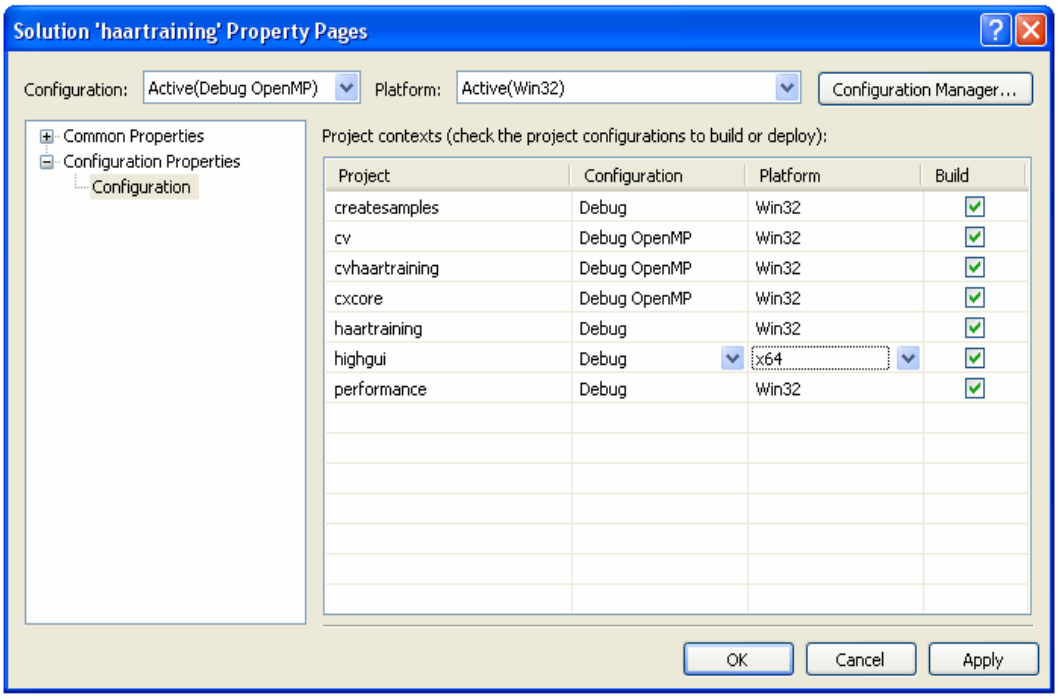
Prilikom detekcije efekt normalizacije se može postići dijeljenjem vrijednosti značajke s devijacijom.

3. Rad s programskim paketom OpenCV

3.1. Pokretanje i priprema

Poznata implementacija algoritma za učenje i pronalaženje objekata Viole i Jonesa nalazi se u sklopu programskog paketa OpenCV [7]. OpenCV se izdaje pod licencom FreeBSD, a verzija 1.0 koja je korištena u ovom završnom radu je dostupna na Internetu i može se pronaći na adresi: http://sourceforge.net/project/showfiles.php?group_id=22870&package_id=16937/.

Nakon preuzimanja paketa, potrebno ga je i prevesti. U tu svrhu u ovom završnom radu je korišten MS Visual Studio 2005. Nakon toga potrebno je u stablu izvornog koda paketa OpenCV izgraditi aplikaciju u kojoj je implementiran algoritam Viole i Jonesa. Ta implementacija nalazi se u kazalu: instalacijski_direktorij\apps\HaarTraining\make. Prije same izgradnje rješenja potrebno je u postavkama rješenja za komponentu highgui postaviti platformu win32 umjesto x64 (slika 5).

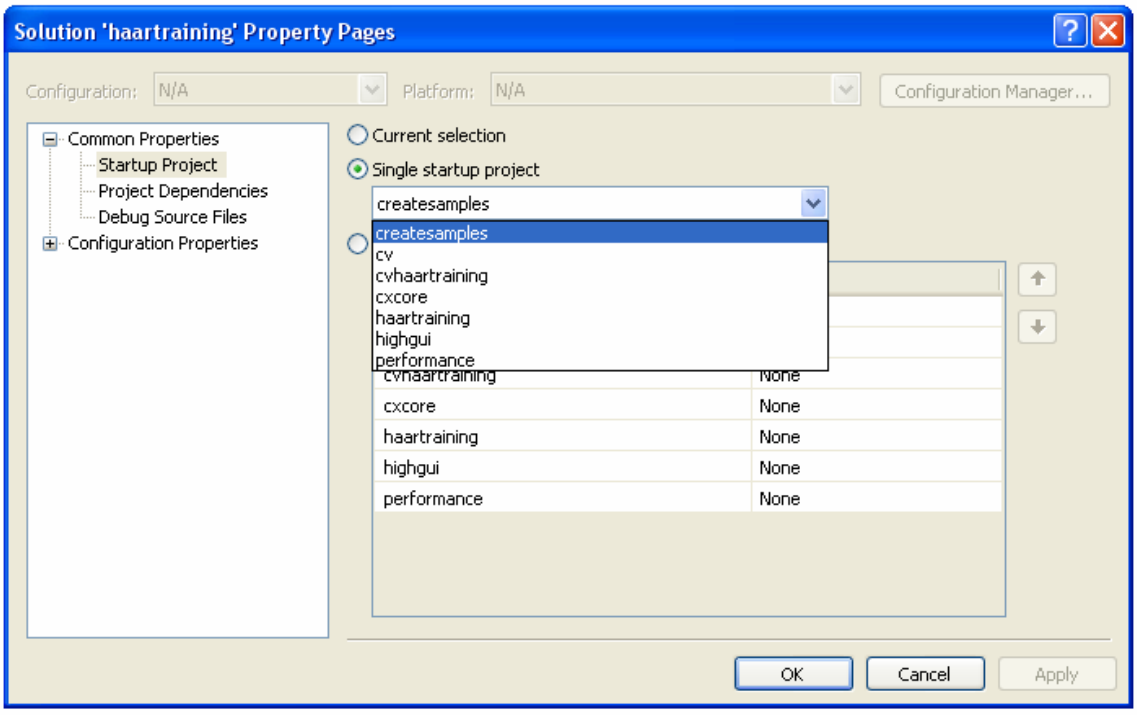


Slika 5. Prozor s postavkama za haartraining rješenje. Prozoru se može pristupiti desnim klikom na rješenje → Properties → Configuration Properties

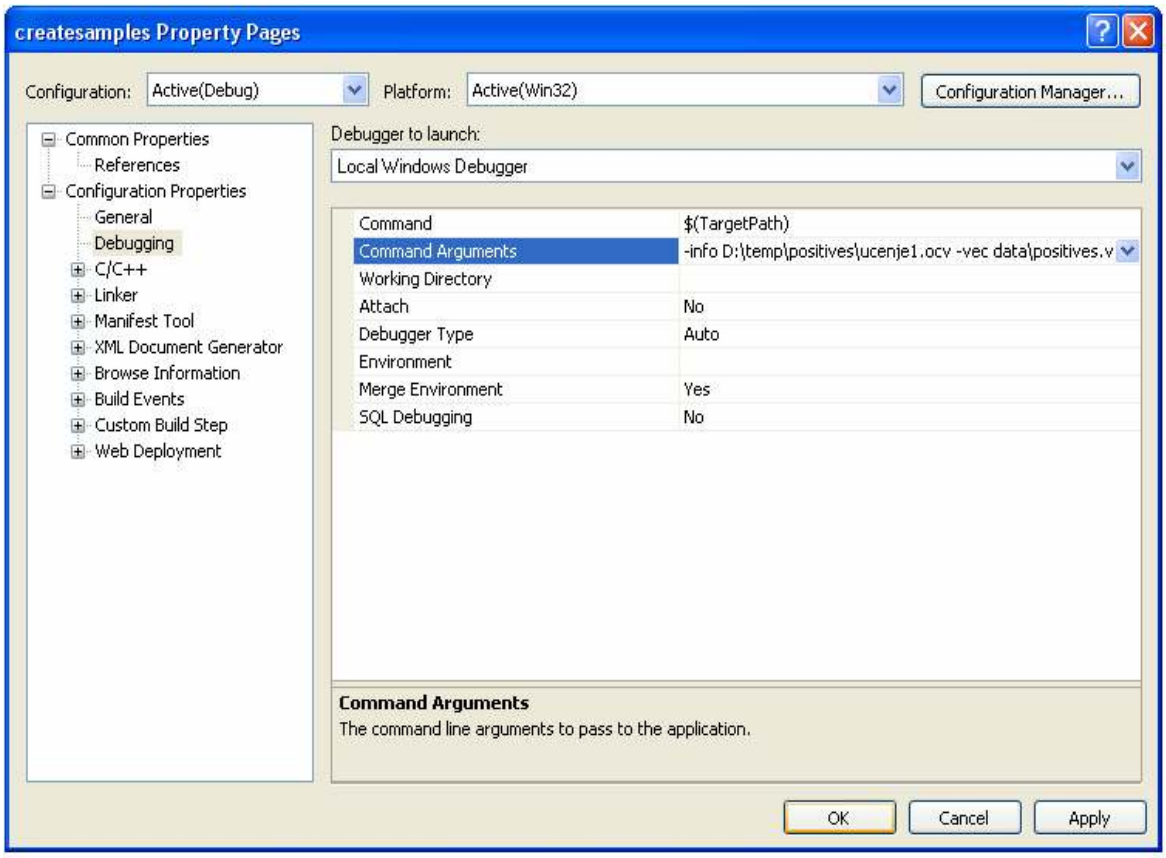
Nakon prevođenja rješenja potrebno je pripremiti tri skupa slika. U prvom, pozitivnom, skupu slika potrebno je označiti tražene objekte. Drugi, negativni, skup slika treba sadržavati slike na kojima uopće neće biti traženih objekata. Za postupak učenja se koriste ova dva skupa. Za testiranje postupka detekcije također je potreban još jedan, testni, skup slika. Idealno, negativni skup će imati puno više slika od pozitivnog [6]. Imena slike te podatke o pravokutnicima u kojima se nalaze znakovi potrebno je zadati kroz tekstualnu datoteku koja treba biti u istoj mapi na disku kao i slike.

Prije pokretanja procesa učenja potrebno je iz pozitivnog skupa slika izdvojiti objekte. To se čini pokretanjem komponente `createsamples` (slika 6). Također potrebno je komponenti predati sljedeće argumente (slika 7):

```
-info lokacija_pozitivnog_skupa_slika\ime_datoteke_pozitivnog_skupa

-vec zeljena_lokacija\positives.vec

-num broj_pozitivnog skupa_skupa_slika

Primjer: -info D:\pos\ucenje1.ocv -vec data\positives.vec -num 824
```



Slika 6. Prozor na kojem se može odrediti koja komponenta u aplikaciji će se pokrenuti. Prozor s može pristupiti desnim klikom na Solution → Properties → Common Properties → Startup Project



Slika 7. Prozor na kojem se mogu zadati argumenti s kojima će se komponenta izvršni program. Može mu se pristupiti desnim klikom na željeni projekt → Properties → Configuration Properties → Debugging

Nakon pokretanja komponente `createsamples` dobivene rezultate se može vidjeti ako se komponenti preda sljedeći argument:

```
-vec lokacija_vec_datoteke\positives.vec -w 24 -h 24
```

Primjer: `-vec data\positives.vec -w 24 -h 24`

Sljedeći korak je učenje. Potrebno je komponentu `haartraining` postaviti kao početnu te joj predati argumente:

```
-data lokacija_za_kaskadu\cascade

-vec lokacija_vec_datoteke\positives.vec

-bg lokacija_negativnog_skupa\ime_datoteke_negativnog_skupa

-npos broj slika u pozitivnom skupu

-nneg broj slika u negativnom skupu

-nstages željena dubina kaskade
```

```
-mem količina memorije za korištenje prilikom učenja

-mode BASIC
```

Primjer:

```
-data data\cascade -vec data\positives.vec -bg D:\pozadin\pozadine.dat
-npos 824 -nneg 97 -nstages 30 -mem 1300 -mode BASIC -w 24 -h 24
```

Valja napomenuti da proces učenja može potrajati nekoliko dana, ovisno o količini pozitivnih i negativnih slika.

Nakon što je proces učenja završio može se pristupiti pronalaženju traženih objekata na testnom skupu slika. Potrebno je komponentu `performance` označiti kao početnu ili alternativno pokrenuti iz komandne linije te joj predati sljedeće argumente:

```
-data lokacija_kaskade\cascade

-info lokacija_testnog_skupa_slika\ime_datoteke_testnog_skupa

-w 24 -h 24 -rs 30
```

Primjer:

```
-data data\cascade -info D:\pos\test1.ocv -w 24 -h 24 -rs 30
```

Po završetku rada komponente `performance` na ekranu će se ispisati podaci o pronađenim, promašenim i pogrešno detektiranim objektima u pojedinim slikama, ukupno vrijeme trajanja procesa, te ukupan broj pronađenih i pogrešno detektiranih objekata. Također će se na lokaciji testnog skupa slika stvoriti nove slike na kojima će crvenim pravokutnicima biti označeni nađeni objekti. Postupak spremanja slika može se izbjeći tako da se komponenti `performance` kao argument preda `-ni`.

3.2. Analiza programske komponente zadužene za detekciju objekata

U glavnoj funkciji glavne komponente programa za detekciju (`performance`) se učitavaju zadani argumenti, nakon čega se sa zadane lokacije učitava kaskada pozivom funkcije `cvLoadHaarClassifierCascade`. Funkcija kao argumente prima lokaciju kaskade te veličinu početnog okna. Sama struktura podataka kaskade zadana je na sljedeći način:

```
typedef struct CvHaarClassifierCascade
{
    int  flags;
    int  count;
    CvSize orig_window_size;
    CvSize real_window_size;
    double scale;
    CvHaarStageClassifier* stage_classifier;
    CvHidHaarClassifierCascade* hid_cascade;
}
  CvHaarClassifierCascade;
```

Unutar strukture `CvHaarClassifierCascade` valja primijetiti i strukturu podataka `CvHidHaarClassifierCascade` koja predstavlja tzv. skrivenu kaskadu.

```
struct CvHidHaarClassifierCascade
{
    int  count;
    int  is_stump_based;
    int  has_tilted_features;
    int  is_tree;
    double inv_window_area;
    CvMat sum, sqsum, tilted;
    CvHidHaarStageClassifier* stage_classifier;
    sqsumtype *pq0, *pq1, *pq2, *pq3;
    sumtype *p0, *p1, *p2, *p3;

    void** ipp_stages;
  };
```

Neskrivena kaskada u sebi sadrži podatke o klasifikatorima, kao što su pragovi, veličina i tip značajki, dok se u skrivenoj kaskadi nalaze pokazivači, koji pokazuju na integralnu sliku kvadriranih slikovnih elemenata, odnosno običnu integralnu sliku. Struktura kaskade i skrivene kaskade se može vidjeti u sljedećem odsječku izvornog koda:

Nakon učitavanja kaskade učitava se datoteka u kojoj se nalaze podaci o testnom skupu slika te pokreće petlja `while` u kojoj se u svakom prolazu učitava nova slika te položaji traženih objekata koji se nalaze u njoj. U sklopu petlje `while` poziva se funkcija `cvHaarDetectObjects`, te se rezultati te funkcije uspoređuju s podacima koji se nalaze u datoteci, nakon čega se rezultati usporedbe ispisuju na

ekran.

Funkcija `cvHaarDetectObjects` na početku inicijalizira i alocira potrebu memoriju za integralnu sliku, integralnu sliku kvadriranih slikovnih elemenata, rezultate te poziva funkciju pretvorbe slike u boji u sivu (engl. greyscale) sliku:

```
CV_CALL( temp = cvCreateMat( img->rows, img->cols, CV_8UC1 ));
CV_CALL( sum = cvCreateMat( img->rows + 1, img->cols + 1, CV_32SC1 ));
CV_CALL( sqsum = cvCreateMat( img->rows + 1, img->cols + 1, CV_64FC1 ));
CV_CALL( temp_storage = cvCreateChildMemStorage( storage ));

...

seq = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvRect), temp_storage );
seq2 = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvAvgComp), temp_storage );
result_seq = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvAvgComp), storage );

...

if( CV_MAT_CN(img->type) > 1 )
{
      cvCvtColor( img, temp, CV_BGR2GRAY );
      img = temp;
}
```

U sljedećem koraku se ispituje grananje `if( flags & CV_HAAR_SCALE_IMAGE )`. Varijabla flags je deklarirana u samoj definiciji funkcije i njezina podrazumijevana vrijednost je jednaka nuli. S obzirom na to, programski odsječak se preskače sve do `else` grananja.

Funkcijom `cvIntegral` se siva slika pretvara u integralnu sliku izračunom piksela kao sume svih piksela lijevo i iznad njega, te spremanjem rezultata u objekt `sum`, odnosno integralnu sliku kvadriranih slikovnih elemenata izračunom piksela kao sume kvadrata svih piksela lijevo i iznad njega, te spremanje rezultata u objekt `sqsum`.

S obzirom da je varijabla `do_canny_pruning` deklarirana na sljedeći način:

```
int do_canny_pruning = flags & CV_HAAR_DO_CANNY_PRUNING;
```

njezina podrazumijevana vrijednost je nula, te se programski odsječci koji zahtijevaju njezinu vrijednost neće izvoditi.

Također treba napomenuti da se preskaču i svi programski odsječci uvjetovani

varijablom `cascade->is_tree`, koji zahtijevaju da kaskada bude u obliku stabla.

U sljedećem koraku započinje izvođenje petlje `for` koja u svakom novom prolazu povećava veličinu pomičnog detekcijskog okna za određeni faktor. Podrazumijevana vrijednost tog faktora je 1.2 i može se promijeniti mijenjanjem sadržaja varijable `scale_factor`. Petlja će se izvoditi sve dok okno ne prekrije sliku u cijelosti.

Kako bi se postigle brže performanse te izbjeglo ponavljanje nepotrebnih dijelova koda u samom procesu detekcije nužno je na početku rada petlje inicijalizirati okno pozivom funkcije `cvSetImagesForHaarClassifierCascade`. Dotična funkcija vrši inicijalizaciju kaskade za okno zadane veličine. Prvo se iz kaskade dobavlja skrivena kaskada, a zatim se izračunava stvarna veličina okna odnosno inverz njegove površine. Ti podaci su ključni kako bi se kasnije mogla izračunati standardna devijacija. Također se postavljaju pokazivači integralnih slika na početnu poziciju okna:

```
cascade = _cascade->hid_cascade;

   ...

_cascade->scale = scale;
_cascade->real_window_size.width = cvRound( _cascade->orig_window_size.width *
scale );
_cascade->real_window_size.height = cvRound( _cascade->orig_window_size.height *
scale );
cascade->sum = *sum;
cascade->sqsum = *sqsum;

equ_rect.x = equ_rect.y = cvRound(scale);
equ_rect.width = cvRound((_cascade->orig_window_size.width-2)*scale);
equ_rect.height = cvRound((_cascade->orig_window_size.height-2)*scale);
weight_scale = 1./(equ_rect.width*equ_rect.height);
cascade->inv_window_area = weight_scale;

cascade->p0 = sum_elem_ptr(*sum, equ_rect.y, equ_rect.x);
cascade->p1 = sum_elem_ptr(*sum, equ_rect.y, equ_rect.x + equ_rect.width );
cascade->p2 = sum_elem_ptr(*sum, equ_rect.y + equ_rect.height, equ_rect.x );
cascade->p3 = sum_elem_ptr(*sum, equ_rect.y + equ_rect.height,
                                 equ_rect.x + equ_rect.width );
cascade->pq0 = sqsum_elem_ptr(*sqsum, equ_rect.y, equ_rect.x);
cascade->pq1 = sqsum_elem_ptr(*sqsum, equ_rect.y, equ_rect.x + equ_rect.width );
cascade->pq2 = sqsum_elem_ptr(*sqsum, equ_rect.y + equ_rect.height, equ_rect.x );
```

```
cascade->pq3 = sqsum_elem_ptr(*sqsum, equ_rect.y + equ_rect.height,
                                      equ_rect.x + equ_rect.width );
```

Nakon toga započinje izvođenje odsječaka uvjetovanih vrijednošću makroa `CV_ADJUST_FEATURES`. Podrazumijevana vrijednost je `CV_ADJUST_FEATURES` jedan. Međutim, tijekom višestrukog izvođenja cijele funkcije pokazalo se da dotični odsječci ne utječu na rezultate za potrebe ovog završnog rada.

U sljedećem koraku funkcije započinje se s inicijalizacijom svih značajki u kaskadi za trenutnu veličinu okna.

```
for( k = 0; k < nr; k++ )
{
       CvRect tr;
       double correction_ratio;

...

       {
              tr.x = cvRound( r[k].x * scale );
              tr.width = cvRound( r[k].width * scale );
       }

...

       {
              tr.y = cvRound( r[k].y * scale );
              tr.height = cvRound( r[k].height * scale );
       }

...

       correction_ratio = weight_scale * (!feature->tilted ? 1 : 0.5);

...

       {
              hidfeature->rect[k].p2 = sum_elem_ptr(*tilted, tr.y +
                                          tr.width, tr.x + tr.width);
              hidfeature->rect[k].p3 = sum_elem_ptr(*tilted, tr.y +
              tr.width + tr.height,tr.x + tr.width - tr.height);
              hidfeature->rect[k].p0 = sum_elem_ptr(*tilted, tr.y, tr.x);
              hidfeature->rect[k].p1 = sum_elem_ptr(*tilted, tr.y +
                                          tr.height, tr.x - tr.height);
       }

       hidfeature->rect[k].weight = (float)(feature->rect[k].weight *
                                                    correction_ratio);
       if( k == 0 )
              area0 = tr.width * tr.height;
```

```
        else
                sum0 += hidfeature->rect[k].weight * tr.width * tr.height;
}
hidfeature->rect[0].weight = (float)(-sum0/area0);
```

Kao što se može primijetiti iz izvornog koda, veličina svih pravokutnika u značajkama se skalira istom vrijednošću kao i okno. Na kraju petlji se osigurava da omjer veličina pravokutnika u značajci ostane isti. Taj se postupak izvodi zato što prilikom skaliranja, a zatim zaokruživanja koordinata značajki, omjer pravokutnika u značajci se može promijeniti. Odnosno omjer svijetlih i tamnih površina u značajci nakon skaliranja ne mora nužno biti isti. Također valja napomenuti da se u ovom završnom radu nisu koristile značajke rotirane za 45^0, te su svi programski odsječci uvjetovani varijablom `feature->tilted` preskočeni.

Time završava rad funkcije `cvSetImagesForHaarClassifierCascade`, te funkcija `cvHaarDetectObjects` započinje pomicati okno kroz sliku. Za svaku poziciju okna poziva se funkcija `cvRunHaarClassifierCascade` koja ispituje postojanje traženog objekta u oknu. Zbog optimizacije okno će dva puta proći kroz cijelu sliku. U prvom prolazu će se tek za prva dva klasifikatora u kaskadi ispitati, te zapamtiti rezultat. Ako je rezultat pozitivan, okno će se pomaknuti za dvostruko više piksela negu u slučaju negativnog rezultata. U drugom prolazu će se obraditi ostatak kaskade.

Kao što je već spomenuto, funkcija `cvRunHaarClassifierCascade` je zadužena za pronalaženje objekta na slici. Nakon provjere da li se okno nalazi u granicama slike, funkcija započinje s izračunom pomaka okna od početne pozicije (gornji lijevi kut slike). Kada se pokazivači na integralnu sliku uvećaju za taj pomak, pokazivat će upravo na kutove trenutne pozicije okna. Zatim se izračunava varijanca koja je nužne u postupku normalizacije okna:

```
if( pt.x < 0 || pt.y < 0 ||
    pt.x + _cascade->real_window_size.width >= cascade->sum.width-2 ||
    pt.y + _cascade->real_window_size.height >= cascade->sum.height-2 )
    EXIT;

p_offset = pt.y * (cascade->sum.step/sizeof(sumtype)) + pt.x;
pq_offset = pt.y * (cascade->sqsum.step/sizeof(sqsumtype)) + pt.x;
mean = calc_sum(*cascade,p_offset)*cascade->inv_window_area;
variance_norm_factor = cascade->pq0[pq_offset]- cascade->pq1[pq_offset] -
```

```
                        cascade->pq2[pq_offset] + cascade->pq3[pq_offset];
variance_norm_factor = variance_norm_factor*cascade->inv_window_area -
                                                              mean*mean;
if( variance_norm_factor >= 0. )
    variance_norm_factor = sqrt(variance_norm_factor);
else
    variance_norm_factor = 1.;
```

Nakon toga može se pristupiti ispitivanju kaskade. Petljom se prolazi kroz boostane klasifikatore da bi se pristupilo slabim klasifikatorima odnosno značajkama. Već je spomenuto da se značajka sastoji od više pravokutnika koji mogu biti tamni (pravokutnik se oduzima od značajke) ili svijetli (pravokutnik se dodaje značajki). Izračunava se razlika integralne slike tamnih i svijetlih pravokutnika te se rezultat uspoređuje s pragom slabog klasifikatora. Na temelju rezultata usporedbe, klasifikator će pozitivno ili negativno doprinositi ispitivanju. Kada se izračuna doprinos svih slabih klasifikatora, njihov zbroj se uspoređuje s pragom snažnog klasifikatora. Ukoliko je rezultat pozitivan, kreće se s obradom sljedećeg klasifikatora u kaskadi. Ako je rezultat negativan, prekida se rad i izlazi se iz funkcije:

```
for( j = 0; j < cascade->stage_classifier[i].count; j++ )
{
    CvHidHaarClassifier* classifier =cascade->stage_classifier[i].classifier + j;
    CvHidHaarTreeNode* node = classifier->node;
    double sum, t = node->threshold*variance_norm_factor, a, b;

    sum =calc_sum(node->feature.rect[0],p_offset) * node->feature.rect[0].weight;
    sum +=calc_sum(node->feature.rect[1],p_offset) *node->feature.rect[1].weight;

    if( node->feature.rect[2].p0 )
      sum+=calc_sum(node->feature.rect[2],p_offset)*node->feature.rect[2].weight;

    a = classifier->alpha[0];
    b = classifier->alpha[1];
    stage_sum += sum < t ? a : b;
}
if( stage_sum < cascade->stage_classifier[i].threshold )
{
    result = -i;
    EXIT;
}
```

Nakon povratka u funkciju `cvHaarDetectObjects` ispituje se stanje varijable `min_neighbors` iz čega se odlučuje hoće li funkcija pri završetku rada eliminirati rezultate u blizini jedno drugoga. Podrazumijevana vrijednost varijable je jedan. Slijedi povratak u glavnu funkciju komponente `performance`.

4. Programska izvedba detekcije kaskadom boostanih klasifikatora

4.1. Oblikovanje programskog rješenja

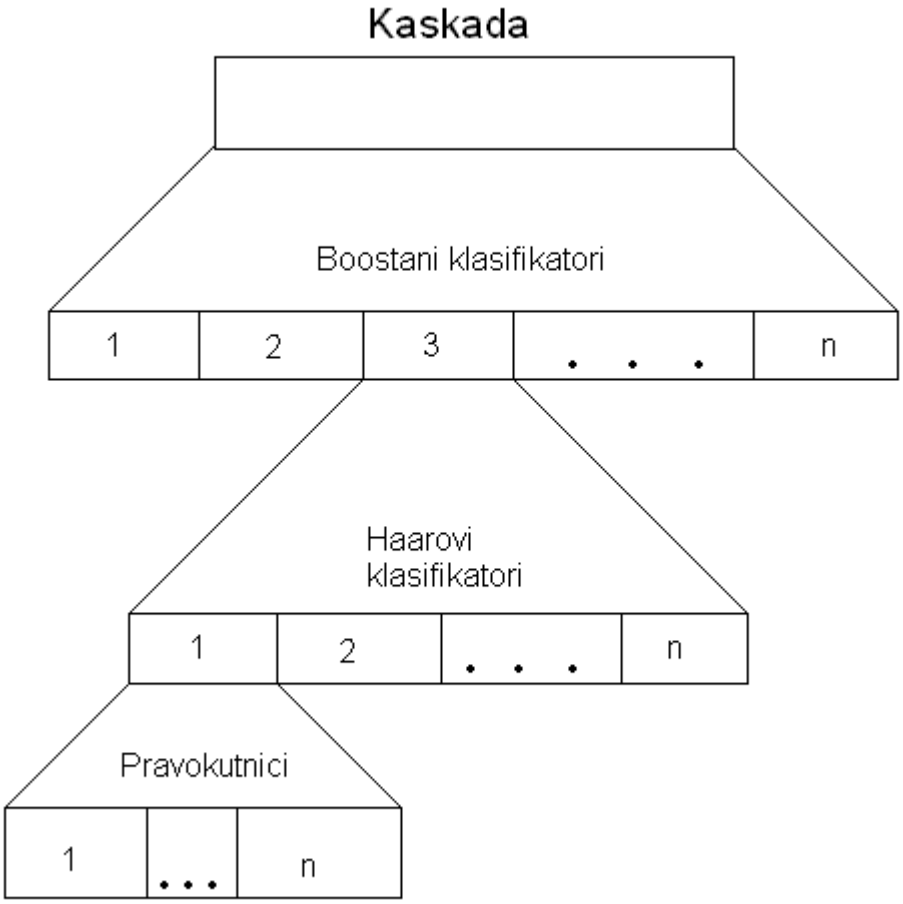
Prilikom rješavanja problema detekcije znakova korištenjem kaskade boostanih klasifikatora, prvi zadatak je oblikovati ga u skladu s načelom jedinstvene odgovornosti, kako bi se postupak pisanja programskog koda olakšao. U tu svrhu osmišljene su tri komponente.

4.1.1. Komponenta za obradu kaskade

Komponenta `ext_vj_casade` je zadužena za učitavanje kaskade i za obradu podataka koji se nalaze u kaskadi. Podatkovna struktura same kaskade je zadana na sljedeći način:

```
class Cascade
{
public:
	std::vector<BoostClassifier> boost_classifier;
	int count;
	int width;
	int height;
	int area;
	int orig_width;
	int orig_height;
public:
	Cascade();
	~Cascade();
public:
	void load(char const* imeKazala);
	bool isValid(){return orig_width>=0;}
public:
	void initCascade ( int* integral, double factor, int max_width, int
max_height);
	int runCascade ( int* integral, int* sqInt, int x, int y, int
max_width, int max_height, double factor);
  };
```

Kaskada se sastoji od više snažnih klasifikatora koji su definirani klasom `BoostClassifier`. Kaskada također sadrži podatke o veličini i površini okna. Isto kao što kaskada sadrži snažne klasifikatore, tako i oni sadrže slabe klasifikatore definirane klasom `HaarClassifier`, koji pak sadrže pravokutnike od kojih se sastoji značajka (slika 8). Pravokutnici su definirani klasom `Rect`. Za učitavanje kaskade je zadužena funkcija `load`.



Slika 8. Prikaz oblikovanja podatkovne strukture kaskade u izvornom kodu

Funkcija `initCascade` inicijalizira kaskadu kada se okno nalazi u gornjem lijevom kutu slike. Prvo se računa veličina i površina okna za trenutno skaliranje. Zatim će funkcija za svaki pravokutnik značajke dobaviti njegovu težinu, a zatim ju podijeliti s skaliranom površinom koju pravokutnik zauzima u oknu. Pri tome se također osigurava da omjer tih rezultata za tamne i svijete značajki bude jednak.

Funkcija `runCascade` provjerava nalazi li se traženi objekt na trenutnoj poziciji okna. Isto kao i u rješenju iz programskog okruženja OpenCV, funkcija će izračunati varijancu odnosno disperziju za okno na trenutnoj poziciji. Zatim će ispitivati značajke odnosno slabe klasifikatore, tako što će izračunati razliku vrijednosti integralne slike za svijetle i tamne pravokutnike u značajci. Na temelju doprinosa slabih klasifikatora ispituje se snažni klasifikator s pragom učitanim iz kaskade. Ako je prag zadovoljen ispituje se sljedeći snažni klasifikator, ako nije, funkcija završava s radom i javlja da se traženi objekt ne nalazi na trenutnoj poziciji okna.

4.1.2. Komponenta s dodatnim pomoćnim funkcijama

Da bi se programsko rješenje ispravno napravilo, nije dovoljna samo komponeta koja radi s kaskadom. U tu svrhu je osmišljena komponenta `ext_vj` čiju glavninu čine tri funkcije. Funkcija `rgbToGrey` pretvara sliku u boji u sivu sliku po formuli:

$$P_s(x,y) = 0.299 \cdot R(x,y) + 0.587 \cdot G(x,y) + 0.114 \cdot B(x,y)$$

gdje *R(x,y)*, *G(x,y)*, *B(x,y)* predstavljaju crvenu, zelenu odnosno plavu komponentu piksela, a P_s vrijednost piksela u sivoj slici.

Funkcija `greyToIntegral` izračunava integralnu sliku, kao i integralnu sliku kvadriranih slikovnih elemenata iz sive slike.

4.1.3. Implementacija i integracija u programsko okruženje cvsh

Za integraciju rješenja u okruženje cvsh, ključna je komponenta `alg_vj`. Struktura kaskade je privatno definirana unutar klase `alg_vj`. Kako bi se izbjeglo učitavanje kaskade za svaku sliku kaskade se učitava u konstruktoru klase `alg_vj`.

```
alg_vj::alg_vj(const CtorArg& ):
  imgs_(2),
  anns_(2),
  pathCascade_("cascade")
{
      cascade_.load(pathCascade_.c_str());
      f = fopen("test1.txt","r");
      gl_hit=gl_miss=gl_fp = 0;
  }
```

Obrada pojedine slike započinje pozivom funkcije `alg_vj::process`. Nakon inicijalizacije potrebnih varijabli i alokacije memorije za sliku, potrebno je sliku pretvoriti u sivi format pozivom funkcije:

```
rgbToGrey (imgs_[0].fmt().width(),
        imgs_[0].fmt().height(),
        (unsigned char*) imgs_[0].bits(),
        imgGrey );
```

U objektu `imgs_[0]` se nalaze podatci o izvornoj slici, kao što su širina, visina odnosno sam sadržaj piksela. Rezultat se vraća pokazivačem u dvodimenzionalno polje `imgGrey`.

Zatim je potrebno dobiti integralnu sliku i integralnu sliku kvadriranih slikovnih elemenata, što se čini pozivom funkcije:

```
greyToIntegral (imgs_[0].fmt().width(),
        imgs_[0].fmt().height(),
        imgGrey, imgInt,imgSqInt  );
```

Izuzev visine i širine slike koji se predaju funkciji, također se predaje i siva slika koja se obrađuje. Rezultati se spremaju u dvodimenzionalna polja `imgInt` i `imgSqInt`.

Nakon što je dobivena integralna slika, potrebno je krenuti s detektiranjem objekata na slici. Prvo se određuje veličina okna skaliranjem, te se odmah poziva funkcija kaskade za inicijalizaciju okna sljedećom naredbom:

```
cascade->initCascade(imgInt, factor, width, height);
```

Nakon što je inicijalizacija obavljena, okno se počinje pomicati kroz sliku. Za svaku poziciju okna se ispituje nalazi li se traženi objekt u njemu sljedećim pozivom funkcije:

```
result = cascade->runCascade(imgInt, imgSqInt, j, i, width, height,
factor);
```

Ukoliko je dobiven pozitivan rezultat, gornja lijeva i donja desna koordinata trenutne pozicije okna se spremaju u vektorsku strukturu podataka `ann_` koja je deklarirana u klasi `alg_vj`:

Korištenjem funkcije `addRectangle` možemo na izvornoj slici prikazati pravokutnike dobivene koordinatama rješenja. Funkcija `addRectangle` je

deklarirana na sljedeći način:

```
static void addRectangle(win_ann_abstract& ann,
    const grid::Pixel& p0,
    const grid::Pixel& p1,
     int col, int width);
```

Po završetku rada komponente `alg_vj` za primljenu sliku, algoritam predaje programskom okruženju izvornu sliku s označenim nađenim objektima.

4.2. Upute za rad

Programsko okruženje cvsh se sastoji od mnogobrojnih algoritama i komponenti. Koji od tih algoritama će se koristiti, zadaje se kroz argumente, slično kao i za programsko okruženje OpenCV (slika Slika ***9***). Da bi ova komponenta za traženje znakova ispravno radila, nužno joj je predati podatke o imenu algoritma i putanji izvorne datoteke. Dodatno može se koristiti i naredba –i za interakcije tijekom izvođenja algoritma.

Algoritam koji će se koristiti se zadaje na sljedeći način:

```
-a = ime_algoritma
```

Potrebno je zadati i adresu datoteke u kojoj se nalaze podaci o slikama koje se obrađuju:

```
-sf = lokacija_datoteke\ime_datoteke.txt
```

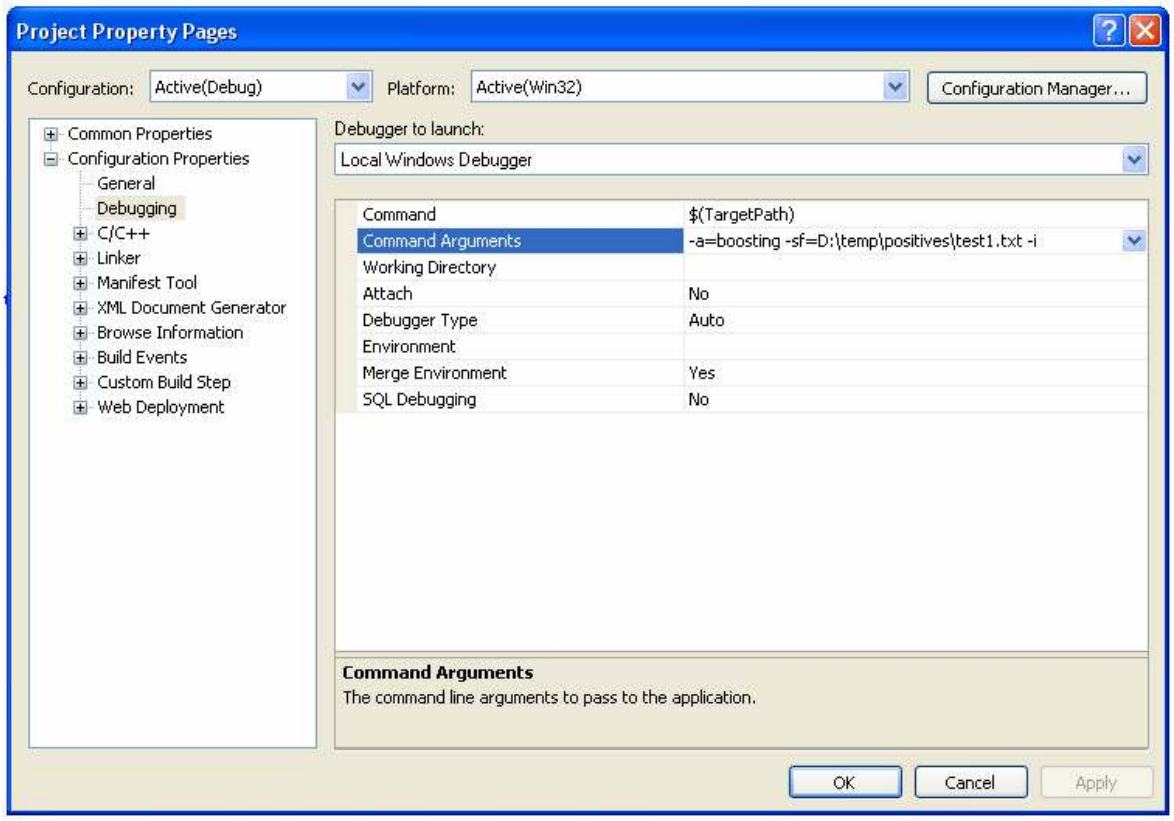
Također, opcionalno, može se zadati i lokacija kaskade.

```
-c = lokacija_kaskade\cascade
```

Primjer cijele naredbe:

```
-a=vj -sf=D:\temp\positives\test1.txt –c=cascade –i
```

U slučaju da lokacija kaskade nije zadana, kako bi algoritam ispravno radio, kaskada mora biti spremljena na istoj lokaciji na kojoj se nalazi i izgrađeno rješenje cijelog programskog okruženja.



Slika 9. Prozor na kojem se mogu zadati argumenti s kojima će se određena komponenta pokrenuti. Može mu se pristupiti desnim klikom na projekt → Properties → Configuration Properties → Debugging

5. Analiza, evaluacija i usporedba rezultata

Za evaluaciju postupka detekcije objekata, korišteno je prijenosno računalo HP Compaq 6720s, s procesorom od dvije jezgre takta 1.6 GHz i 1024 MB radne memorije. Dimenzije slika koje su korištene iznose 720x576 piksela. Postupak učenja je izveden za znakove opasnosti, odnosno prometne znakove u obliku trokuta s crvenim obrubom, te žutom ili bijelom unutrašnjosti. Testiranje je provedeno na dva testna uzorka. Prvi testni uzorak se sastojao od 91 slike na kojoj se nalazio 101 znak, a drugi testni uzorak se sastojao od 172 slike na kojoj su nalazilo 188 znakova. U tablici 1, kao i u tablici 2 mogu se vidjeti rezultati, odnosno brzina izvođenja programske komponente zadužene za detekciju objekata koja se nalazi u sklopku OpenCV programskog paketa. Također se mogu vidjeti i rezultati dobiveni izvođenjem programa razvijenog u sklopu ovog završnog rada i integriranog u cvsh programsko okruženje.

Programsko rješenje razvijeno u sklopu završnog rada ima uspješnost od 82% naspram 67% od onog u paketu OpenCV. Iz tog podatka, kao i iz ostalih podataka iz tablice 1, vidimo da je rješenje implementirano u okruženju cvsh nešto osjetljivije. Značajno veći broj pogrešnih detekcija je djelomično rezultat te osjetljivosti, ali i činjenice da se u implementaciji unutar paketa OpenCV nalazi funkcija za eliminiranje pravokutnika koji se nalaze u neposrednoj blizini.

Valja napomenuti da se upotrebom većeg skupa za učenje mogu dobiti bolji rezultati detekcije [3].

Tablica 1 Usporedba implementacije u programskom paketu OpenCV i programskom okruženju cvsh

	Ispitni skup slika	Broj znakova	Broj uspješnih detekcija	Broj promašenih detekcija	Broj pogrešnih detekcija
OpenCV	1	101	79	22	33
cvsh	1	101	89	12	128
OpenCV	2	188	115	73	93
cvsh	2	188	148	40	179

Iz tablice 2 možemo vidjeti da je vrijeme izvođenja programskog rješenja razvijenog u sklopu završnog rada otprilike deset puta sporije od implementacije unutar paketa OpenCV.

Tablica 2 Usporedba brzine izvođenja dviju implementacija

	Ispitni skup slika	Broj slika	Ukupno vrijeme izvođenja (s)	Prosječno vrijeme izvođenja po slici (slika/s)
OpenCV	1	91	33,4	2,725
cvsh	1	91	357,3	0,255
OpenCV	2	172	59,9	2,871
cvsh	2	172	631,4	0,272
OpenCV	1+2	263	93,3	2,819
cvsh	1+2	263	988,7	0,266

Prilikom profiliranja koda razvijenog programskog rješenja pokazalo se da se najviše vremena troši na obradu funkcije `runCascade`, otprilike 76% vremena. Takvo što je i očekivano s obzirom na 615001 poziv te funkcije tijekom obrade jedne slike. Međutim, teško je izdvojiti glavnog čimbenika, jer sve aritmetičke operacije daju određeni doprinos u vremenskom trajanju funkcije.

Drugi čimbenik pri utrošku vremena jesu procesi `ftol2_pentium4` i `ftol2_sse`. Riječ je o procesima okruženja MS Visual Studio 2005, zaduženima za pretvorbu jednog tipa podataka u drugi. U ovom konkretnom slučaju riječ je o pretvorbi decimalnog zapisa brojeva, odnosno tipa podataka `double` i `float` u cjelobrojni zapis tj. tip podatka `int`. Otprilike 15% vremena otpada na obradu te dvije funkcije.

Treći čimbenik je pomoćna funkcija `round`, na koju otpada otprilike 7% vremena. Sve ostale funkcije zauzimaju manje od 1% vremena ukupnog izvođenja.

Za profiliranje koda je korištena probna verzija aplikacije Intel® VTune™ Performance Analyzer.

5.1. Primjeri uspješne detekcije

Na slici 10 možemo vidjeti kako je znak uspješno detektiran u povoljnim uvjetima. Kvaliteta slike je jako dobra, nema zamućenja, znak je u relativnoj blizini. Samu lakoću detekcije ovog znaka potvrđuje i iznimno velik broj pozitivnih detekcija na njemu.

Slika 10 Primjer uspješne detekcije znaka

Na slikama 11 i 12 možemo vidjeti kako su znakovi detektirani iako je slika zbog kretanja automobila zamućena.

Slika 11 Primjer uspješne detekcije znaka

Slika 12 Primjer uspješne detekcije znaka

Na slikama 13 i14 možemo vidjeti kako su znakovi uspješno detektirani unatoč daljini, odnosno maloj veličini u odnosu na dimenzije slike.

Slika 13 Primjer uspješne detekcije znaka

Slika 14 Primjer uspješne detekcije znaka

Na prikazanim primjerima vidimo da za pronalazak znakova uvjeti ne moraju biti idealni. Doduše, važno je napomenuti da na istim primjerima nema vremenskih nepogoda ili drugih uvjeta koji bi mogli negativno utjecati na detekciju.

5.2. Pogrešno detektirani objekti

Na slici 15. možemo vidjeti kako sjene i guma poljoprivrednog vozila čine oblik nalik trokutu, a kontrast između sjena i boje trave je dovoljan za pogrešnu detekciju.

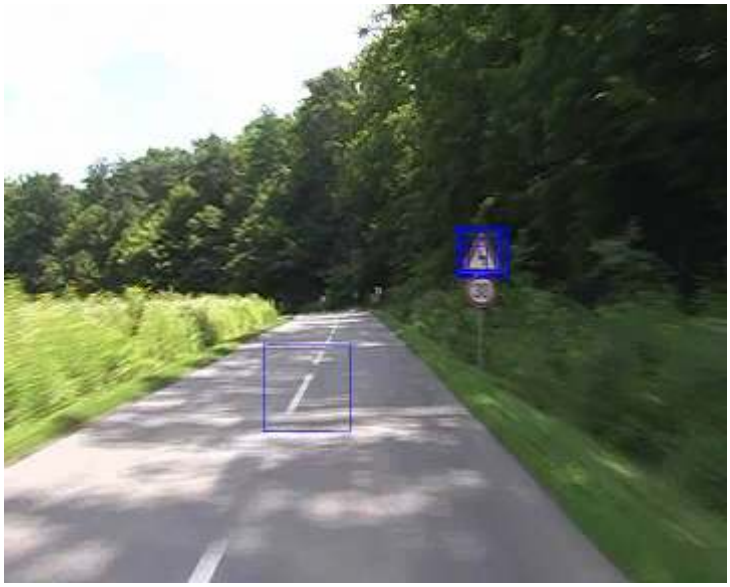
Slika 15 Pogrešna detekcija

Na slici 16 vidimo kako oblici oblika kuća odnosno krovova i prozora doprinose pogrešnim detekcija na stambenim objektima.

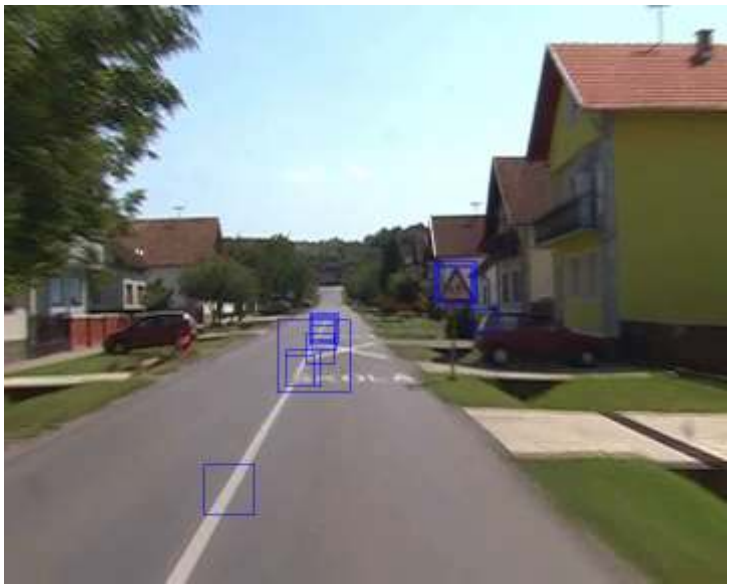
Slika 16 Pogrešna detekcija

Na slikama 17 i 18 možemo vidjeti tipične primjere gdje zbog doprinosa bijele isprekidane crte na tamnoj cesti dolazi do pogrešne detekcije. Slika 18 je rezultat

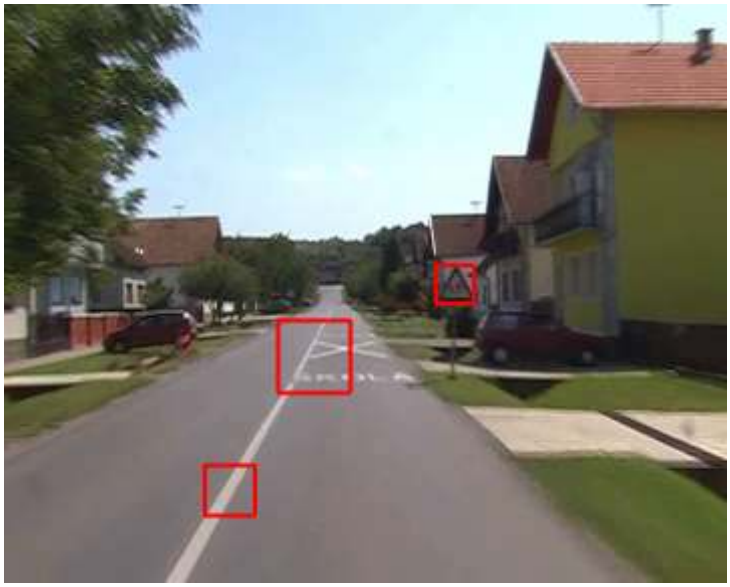
dobiven iz paketa OpenCV i uspoređujući ju sa slikom 17, možemo vidjeti kako zbog eliminacije susjednih detekcija Lienhartova implementacija prijavljuje manje pogrešnih detekcija.



Slika 17 Pogrešna detekcija



Slika 18 Pogrešna detekcija



Slika 19 Pogrešna detekcija

Slika 20 Pogrešna detekcija

Na slici 20 izuzev tipične pogrešne detekcije na cesti, mogu se vidjeti tipične pogrešne detekcije u stablima odnosno granama drveća. Pomnijim promatranje može se vidjeti kako oblici koji se pojavljuju uslijed igre sjena odnosno položaja grana drveća često zatvaraju oblike trokuta.

5.3. Promašene detekcije

Određen broj promašenih detekcija se dogodio kada su prometni znakovi bili položeni na cesti, kao na slikama 21 i 22. Na prvi pogled, moglo bi se reći da to

nema veze, ali ono što valja primijetiti jest da sva tri znaka koja nisu detektirana su također nakošena, odnosno zbog nagiba zemlje je došlo do značajne rotacije znakova. Također na slici 22 možemo vidjeti kako je jedan znak uspješno detektiran, te da taj znak nije bio nakošen.

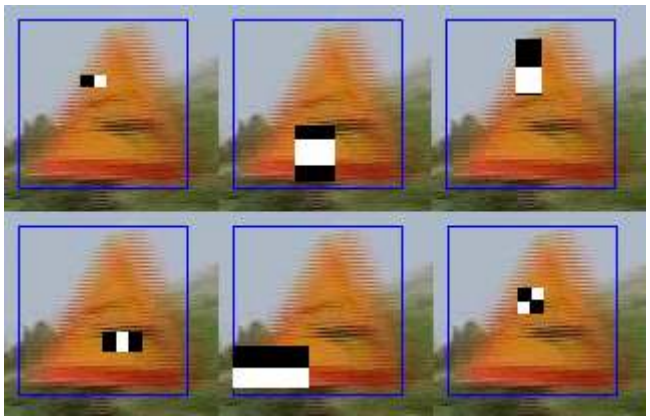
Slika 21 Promašena detekcija dva znaka

Slika 22 Promašena detekcija jednog i uspješna detekcija drugog znaka.

Na slici 23 se može vidjeti promašena detekcija, dapače u postupku detekcije znak nije uspio niti jednom proći drugu razinu kaskade.

Slika 23. Promašena detekcija

Za isti znak na slici 24 može se vidjeti niz značajki koje se nalaze u sklopu prvog boostanog klasifikatora u kaskadi. Jedino druga značajka u prvom redu će dati pozitivan odgovor, dok preostale daju negativan. Ako pogledamo znak u sivom spektru na slici 25 možemo vidjeti da rubovi prometnog znaka nisu dovoljno tamni u odnosu na unutrašnjost. Samim time ni razlika između tamnih i bijelih područja značajke nije dovoljno velika za pozitivnu detekciju.



Slika 24 Značajke unutar okna

Slika 25. Siva slika s promašenim znakom sa slike 21.

Iz navedenih primjera možemo vidjeti da na promašene detekcije najviše utječu položaj odnosno rotacija znaka, kao i izbljeđivanje boja odnosno smanjenje kontrasta između tamnih i svijetli područja na znakovima.

6. Zaključak

Cilj ovog završnog rada bio je proučiti način rada algoritma Viole i Jonesa za traženje objekata kaskadom boostanih klasifikatora i analizirati način rada rješenja u sklopu okruženja OpenCV, te na temelju stečenih znanja izgraditi vlastito rješenje. Također zadatak je bio ocijeniti dobiveni algoritam u svrhu pronalaženja znakova na slikama.

Performanse razvijene implementacije su slabije od performansi referentne implementacije iz paketa OpenCV. Otprilike 3.75 sekundi koliko je potrebno za obradu slike je premalo da bi se algoritam u trenutnom stanju mogao koristiti za detekciju znakova u realnom vremenu. U ovom kontekstu je važno spomenuti da Lienhartov program radi deset puta brže, odnosno u stanju je obraditi tri slike u sekundi. Ti rezultati pokazuju da prostor za optimizaciju performansi algoritma postoji te da bi se trebao iskoristiti.

Kvalitete dobivenih rezultata su obećavajuće, naime oko 80% znakova koji se pojavljuju na slikama su pronađeni. Do promašenih detekcija može doći uslijed efekta zamućenja, preplitanja ili rotacije znaka. Treba primijetiti da se značajan broj pogrešnih detekcija znakova zbiva u dijelovima drveća, odnosno ceste kao i u krovovima i prozorima kuća i zgrada. Korištenjem kaskade naučenom na većem broju označenih znakova mogu se dobiti još bolji rezultati.

U budućnosti svakako valja razmotriti problem optimizacije performansi. Također treba vidjeti kako se kvaliteta rezultata može poboljšati korištenjem boja u postupku detekcije. Za otklanjanje lažnih detekcije mogu se koristiti geometrijska ograničenja što bi također imalo povoljan utjecaj na performanse. U svakom slučaju, sasvim je izvjesno da niti jedna metoda zasebno neće moći riješiti problem prepoznavanja znakova na zadovoljavajući način, stoga se kao logičan zaključak nameće korištenje različitih metoda koje će nadomjestiti nedostatke jedna drugoj.

7. Literatura

[1] Viola P., Jones M., Robust Real-Time Face Detection, International Journal of Computer Vision 57(2), 137–154, http://lear.inrialpes.fr/people/triggs/student/vj/viola-ijcv04.pdf, 2004.

[2] Čuljak, M.: Primjena strojno naučenih klasifikatora u računalnom vidu, Seminar, Fakultet elektrotehnike i računarstva, Zagreb, 2009.

[3] Brkić, K., Pinz, A., Šegvić, S.: Traffic sign detection as a component of an automated traffic infrastructure inventory system, OAGM'09, Stainz, 2009.

[4] Haar-like features – Wikipedia, http://en.wikipedia.org/wiki/Haar-like_features, 20. lipnja 2007.

[5] Robust real time object detection – Wikipedia, http://en.wikipedia.org/wiki/Robust_real-time_object_detection, 12. prosinca 2007.

[6] Boosting – Wikipedia, http://en.wikipedia.org/wiki/Boosting, 9. veljače 2007.

[7] Welcome – OpenCV Wiki, http://opencv.sf.net, 14. veljače 2006.

[8] Florian A., How-to build a cascade of boosted classifiers based on Haar-like features, Object Detection – OpenCV Wiki, http://robotik.inflomatik.info/other/opencv/OpenCV_ObjectDetection_HowTo.pdf, 2. rujna 2003.

[9] Lienhart R., Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection, DAGM'03, 25th Pattern Recognition Symposium, http://www.lienhart.de/Publications/DAGM2003.pdf, rujan 2003.

Pronalaženje objekata kaskadom boostanih Haarovih klasifikatora

Sažetak

U sklopu ovog rada razmatra se pronalaženje znakova u slikama snimljenih iz vozila u pokretu. U tu svrhu koristi se algoritam Viole i Jonesa za detekciju objekata u realnom vremenu. U implementaciji ove metode koristi se kaskada binarnih klasifikatora koji se dobivaju boostanjem slabijih klasifikatora dobivenih direktno iz Haarovih značajki.

U radu je analizirana implementacija algoritma unutar programskog paketa OpenCV te je, također, napravljena vlastita implementacija u jeziku C++, koja je integrirana u programsko okruženje cvsh.

Također je napravljena evalucija postupka detekcije obje implementacije u sklopu koje su dane su ocjene o brzini te točnosti opisanog postupka.

Ključne riječi:

Računalni vid, prometni znakovi, detekcija, realno vrijeme, Haarove značajke, binarni klasifikatori, boostanje, strojno učenje, algoritam Viole i Jonesa.

Object detection using a cascade of boosted Haar classifiers

Abstract

This paper considers detection of traffic signs in the pictures taken from a moving vehicle. For that purpose an algorithm by Viola and Jones for robust real time object detection is being used. In the process of implementation of this method a cascade of binary classifiers is used. Classifiers are created by boosting weaker classiefers based on Haar features.

This paper analyses the implementation of said algorithm within OpenCV program package. Also one's own implementation has been created within C++ language that was integrated in cvsh environment.

An evaluation of both implementations was done as well. The evaluation consisted of testing speed and accuracy of described procedure.

Key words:

Computer vision, traffic signs, detection, real time, Haar features, binary classifiers, boosting, machine learning, Viola and Jones algorithm.

Privitak

Uz rad je priložen DVD-ROM s izvornim kodom i izvršnom verzijom programskog okruženja cvsh, te izvorni kod programskog paketa OpenCV. Uz to, priloženi su skupovi slika za učenje i testiranje.