SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Završni rad br. 809

Precizno lociranje prometnih znakova gradijentnom optimizacijom

Petar Šime Čepo

Zagreb, lipanj 2009.

Zahvala

Zahvaljujem se svom mentoru, Siniši Šegviću na korektnom odnosu prema studentima, predanosti svojoj ulozi mentora i pomoći koju je u svakom trenutku bio spreman pružiti. U tekst ovog rada utkana je podrška mojih roditelja, brata i sestre koji su me podržavali tokom cijelog školovanja. Posebno se zahvaljujem svojoj djevojci Tanji koja me je motivirala u teškim trenucima. Isto tako, zahvaljujem se svojim prijateljima Andreju Bernardu, Davoru, Juri, Marini i Vladi. Bez njih bi studiranje bilo mnogo dosadnije...

Sadržaj

1 Uvod

Od samih početaka razvoja računala javila se potreba kako izgraditi sustave koji će se ponašati ljudski i pomagati ljudima u njihovim aktivnostima. Jedna od vrlo čestih aktivnosti suvremenog čovjeka je upravljanje vozilom u prometu i prepoznavanje prometnih znakova. Na temelju prepoznatih znakova čovjek odlučuje u prometu. Ljudske odluke često su smrtonosne pa se u novije vrijeme radi na računalnim sustavima koji će upravljati vozilom umjesto čovjeka. Još uvijek računalo ne može donositi bolje odluke u prometu nego čovjek pa se intenzivno radi na usavršavanju sustava za automatsko upravljanje. Kako bi takav sustav mogao donositi odluke u prometu, on mora biti u stanju detektirati prometne znakove. Za detekciju prometnih znakova sustav za detekciju mora imati neko inicijalno znanje o znakovima. On mora poznavati sve vrste prometnih znakova koji mu služe kao predlošci za usporedbu sa znakovima na koje naiđe u prometu. Prvi zadatak sustava za detekciju prometnih znakova je zaključiti da su dva znaka (predložak i znak na cesti) istog tipa bez obzira na različite fizikalne karakteristike (osvjetljenje, položaj znaka, nijanse boja). Njegov drugi zadatak je odrediti točnu lokaciju znaka u prometu. Poglavlje "Računalni vid" pobliže opisuje motivacijske faktore za razvoj inteligentnih sustava računalnog vida.

1.1 Računalni vid

Kako bi inteligentni stroj bio u stanju imitirati čovjeka, on mora moći gledat svijet oko sebe. Pojam "gledati" uključuje mogućnost viđenja svijeta svojim "očima", ali i mogućnost analize viđenog. Znanstveni radovi u području računalnog vida zasnivaju se na obradi slike koja može biti u nekoliko različitih oblika (video sekvenca, pogledi iz više kamera ili multidimenzionalni podaci iz čitača) [3].

Razvoju računalnog vida uvelike je pridonio biološki vid, disciplina koja proučava kako živa bića gledaju i obrađuju viđene podatke. Mnogi algoritmi u umjetnoj inteligenciji čija je vremenska složenost eksponencijalna mogu se zamijeniti s algoritmima koji se temelje na oponašanju prirode, polinomske su složenosti i daju vrlo dobre (ali ne i savršene) rezultate [3].

Kao znanstvena disciplina počeo se razvijati krajem 70-ih godina 20-og stoljeća kada su stvorena računala koja mogu obraditi sliku. Većina aplikacija računalnog vida nema sposobnost učenja, ali se u zadnje vrijeme sve više radi na tome. Računalni vid dio je umjetne inteligencije i njeni podaci su ulaz inteligentnog sustava. Klasifikacijom ulaznih podataka u skupine bavi se raspoznavanje uzoraka. Kako bi sustavi računalnog vida odražavali realni svijet, za njihovu implementaciju potrebno je poznavati fiziku, poglavito optiku. Takvi sustavi detektiraju elektromagnetsko zračenje vidljivog spektra koje se odbija od objekte i tvori sliku. Neurobiologija je povezana sa računalnim vidom na način da se građa ljudskog i životinjskog živčanog sustava pokušava implementirati u sustave računalnog vida. Ostale discipline koje su vezane za računalni vid su (Slika 1): obrada signala, matematika, obrada i analiza slike, robotski vid i strojni vid [3].

1.2 Opis problema

Slike pribavljene iz vozila u pokretu sadrže u sebi trokutne prometne znakove. Položaj prometnog znaka na slici otprilike je poznat. Zadatak rada je precizno odrediti položaj znaka unutar slike, odrediti položaj crvenog ruba znaka i prikazati histogram nijansi crvene boje unutar ruba. Točna lokacija znaka određuje se algoritmima gradijentne optimizacije koji su detaljno opisani u sljedećem poglavlju. Crveni rub znaka lako je odrediti pošto je rub predloška nakon provedenog algoritma jednak po obliku rubu znaka. Histogram će prikazati učestalost pojave određene nijanse crvene boje u rubu znaka.

2 Poravnavanje slike s predloškom gradijentnom optimizacijom

Kako bi se slika uspješno poravnala sa predloškom koriste se brojni algoritmi gradijentne optimizacije. Oni se baziraju na iterativnom uspoređivanju predloška i slike i modificiranju predloška da što više nalikuje slici ili obrnuto. Svaki algoritam ima definiran maksimalni broj iteracija i prag koliko predložak i slika mogu biti različiti da bi se algoritam nastavio. Kada su predložak i znak najsličniji, algoritam se prekida. Konvergencija algoritma uvelike ovisi o početnom položaju predloška naspram slike. Ako je središte predloška smješteno gdje i središte znaka te ako je predložak malo manji od slike, konvergencija algoritma je skoro pa zagarantirana. Četiri najčešće korištena algoritma gradijentne optimizacije su: algoritam Lucasa i Kanadea (slijedni aditivni), slijedni kompozicijski, inverzni kompozicijski i inverzni aditivni algoritam. U zadatku je upotrijebljen slijedni algoritam iz razloga što se kod slijednih algoritama modificira predložak u odnosu na znak, a upravo je modificirani predložak potreban za točno lociranje znaka.

2.1 Algoritam Lucasa i Kanadea

Cilj algoritma je poravnati predložak $T(x)$ sa ulaznom slikom gdje je $x = (x, y)^\tau$ vektor koji sadrži koordinate piksela. Neka je $W(x; p)$ transormacija vektora $x = (x, y)^\tau$ iz koordinatnog sustava predloška u koordinatni sustav slike. $W(x; p)$ je definirana kao umnožak matrice transformacije $W(p)$, gdje je vektor parametara $p = (p_1, p_2, ..., p_n)^\tau$ i vektora $x = (x, y)^\tau$ [4].

$$\mathrm{W}(\mathrm{x};\mathrm{p}) = \begin{pmatrix} 1+\mathrm{p}_1 & \mathrm{p}_3 & \mathrm{p}_5 \\ \mathrm{p}_2 & 1+p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

U ovom slučaju parametri p_1 i p_4 označavaju skaliranje u smjeru osi x, odnosno u smjeru osi y. Parametri p_3 i p_4 označavaju rotaciju, a parametri p_5 i p_6 translaciju u smjeru x osi, odnosno u smjeru y osi. Broj parametara može biti proizvoljan. Za naš slučaj potrebno je imati svih šest elemenata jer se predložak u većini slučajeva mora nejednoliko skalirati, rotirati i translatirati.

2.1.1 Cilj algoritma Lucasa i Kanadea

Ovaj algoritam nastoji minimizirati sumu kvadrata razlike intenziteta vrijednosti piksela predloška i slike[4]:

$$\sum_x \left[I(W(x; p)) - T(x) \right]^2 \rightarrow \min \tag{2}$$

Minimizacija sume mora se odvijati na podpikselnoj razini u slučaju da je predložak manji od izvorne slike zato što predložak sadrži manje piksela od izvorne slike pa se jedan piksel predloška preslika u više piksela izvorne slike.

Budući da se parametri p stalno mijenjaju, pod pretpostavkom da nam je poznata vrijednost vektora p moguća je optimizacija algoritma izrazom[4]:

$$\sum_x \left[I\left(W\left(x; p+\Delta p\right)\right) - T\left(x\right)\right]^2 \rightarrow \min \tag{3}$$

Nakon minimiziranja izraza (3) vrijednost parametara vektora $p = (p_1, p_2, ..., p_n)^\tau$ se mijenja za neki Δp. U svakom koraku vrijednosti parametara se mijenjaju na ovaj način[4]:

$$p \leftarrow p + \Delta p\,. \tag{4}$$

Algoritam se prekida u trenutku kada promjena parametara matrice transformacije Δp padne ispod zadanog praga, tj. kada vrijedi $\|\Delta p\| < \varepsilon$.

2.1.2 Izvod algoritma Lucasa i Kanadea

Kako je formula (3) nelinearna i teška za programsku implementaciju, ona se može linearizirati sljedećim izrazom:

$$\sum_x \left[I(W(x; p)) + \nabla I \frac{\delta W}{\partial p} \Delta p - T(x) \right]^2. \tag{5}$$

Varijabla $\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$ je gradijent izvorne slike na mjestu $W(x; p)$ odnosno na koordinatama x koje su transformirane matricom transformacije $W(x; p)$. Derivacija matrice transformacije po parametrima p naziva se Jakobijeva matrica. [4]:

$$\frac{\partial W}{\partial p} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}. \tag{6}$$

Derivacija izraza (1) glasi:

$$\frac{\partial W}{\partial p} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}. \tag{7}$$

Neka je varijablom a definiran umnožak gradijenta piksela sa njegovom Jakobijevom matricom:

$$\left[\nabla I \frac{\partial W}{\partial p}\right] . \tag{8}$$

Hesseova matrica tada je definirana kao [4]:

$$H = \sum_x a^\tau a. \tag{9}$$

Promjena parametara Δp predstavlja malu promjenu predloška prema njegovom konačnom obliku[4]:

$$\Delta p = H^{-1} \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^\tau \left[T(x) - I(W(x; p))\right], \tag{10}$$

odnosno skraćeno možemo napisati:

$$\Delta p = H^{-1} b \left[T(x) - I(W(x; p))\right]. \tag{11}$$

Promjena parametara Δp predstavlja umnožak inverza Hesseove matrice , razlike predloška i slike i varijable b, gdje varijabla b sume umnožaka svakog piksela gradijenta sa Jakobijevom matricom tog piksela.

2.1.3 Koraci algoritma

Sada kad su poznati svi potrebni izrazi za izvršavanje algoritma mogu se definirati koraci algoritma [4]:

Ponavljaj:

1) Transformiraj izvornu sliku I sa matricom transformacije $W(x; p)$ da bi se izračunalo $I(W(x; p))$.
2) Izračunaj razliku predloška i slike $T(x) - I(W(x; p))$.
3) Transformiraj gradijent ∇I sa matricom transformacije $W(x; p)$.
4) Izračunaj Jakobijevu matricu $\frac{\partial W}{\partial p}$ na mjestu transformiranog piksela gradijenta iz prethodnog koraka.
5) Izračunaj umnožak gradijenta ∇I i Jakobijeve matrice $\frac{\partial W}{\partial p}$.
6) Izračunaj Hesseovu matricu prema izrazu (9).
7) Izračunaj $\sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^\tau \left[T(x) - I(W(x; p))\right]$.

8) Izračunaj Δp koristeći izraz (11).
9) Osvježi vrijednosti parametara koristeći izraz (4).

dok $\|\Delta p\| \leq \varepsilon$.

Na početku iteracije potrebno je transformirati piksel predloška u koordinatni sustav izvorne slike da bi kasnije vidjeli da li su transformirani piksel i piksel predloška maksimalno slični. Isto tako moramo dobiti transformirani piksel gradijenta. Gradijent usmjerava algoritam prema području gdje će on konvergirati. Hesseova matrica je računalno najzahtjevniji dio algoritma zato jer je potrebno svaki piksel gradijenta po x osi i po y osi množiti sa dva vektora Jekobijeve matrice od kojih je svaki duljine n. Na kraju algoritma izračunavaju se vrijednosti promjene parametara i dodaju vrijednostima parametara prethodnog koraka.

2.1.4 Vremenska složenost algoritma Lucasa i Kanadea

U razmatranju vremenske složenosti koriste se parametri n i N, gdje je n broj parametara matrice transformacije, a N broj piksela predloška. Prvi korak ima složenost $O(nN)$ zato jer se svaki od N piksela izvorne slike množi za matricom transformacije čiji je rang n. U drugom koraku složenost je $O(N)$ zato jerse izračunava razlika svih N piksela predloška i korespondentnih piksela izvorne slike. Treći korak je složenosti $O(nN)$ iz istog razloga kao i prvi korak, samo što nije riječ o izvornoj slici nego o gradijentu. Četvrti korak ima složenost $O(nN)$ zato jer se izračunava Jakobijeva matrica ranga n za svaki piksel gradijenta. U petom koraku se N piksela gradijenta množe sa Jakobijevom matricom pa je složenost $O(nN)$. Šesti korak je vremenski najzahtjevniji i njegova je složenost $O(n^2N)$ jer se za svaki piksel gradijenta izračunava umnožak dvaju Jakobijevih matrica. Sedmi korak je vremenske složenosti $O(nN)$ iz istog razloga kao i peti. Osmi korak je složenosti $O(n^3)$ zato jer je u izrazu (11) Hesseova matrica ranga n^2, a varijabla b ranga n. Deveti korak je složenosti $O(N)$ jer za N piksela predloška mijenjamo vrijednosti parametara.

Tablica 1 pokazuje vremensku složenost algoritma po koracima[4]:

Tablica 1

1. korak	2. korak	3. korak	4. Korak	5. korak	6. korak	7. korak	8. korak	9. korak	Ukupno
$O(nN)$	$O(N)$	$O(nN)$	$O(nN)$	$O(nN)$	$O(n^2N)$	$O(nN)$	$O(n^3)$	$O(N)$	$O(n^2N+n^3)$

2.2 Ostali algoritmi poravnavanja gradijentnom optimizacijom

Algoritam Lucas-Kanade je prvi algoritam gradijentne optimizacije. Kasnije razvijeni inverzni algoritmi izračunavaju Hesseovu matricu prije početka iterativnog dijela algoritma pošto njeno izračunavanje troši najviše vremena. To je ostvareno na način da se zamijeni uloga predloška i slike, odnosno da se oblikizvorne slike transformira u predložak. Algoritmi koji se naredno opisuju su: kompozicijski, inverzni kompozicijski i inverzni aditivni algoritam. Oni neće biti detaljno opisani nego će samo biti naglašene njihove razlike u odnosu na algoritam Lucas-Kanade.

2.2.1 Kompozicijski algoritam

Ovaj algoritam je vrlo sličan aditivnom algoritmu Lucas-Kanade jer također izračunava Hesseovu matricu u svakoj iteraciji. Formula minimizacije kvadrata razlike ipak se malo razlikuje od one u algoritmu Lucas-Kanade[4]:

$$\sum_x \left[I(W(W(x;\Delta p);p)) - T(x) \right]^2 . \tag{12}$$

Ova formula označava da će se matricom $W(W(x;\Delta p);p)$ transformirati piksel predloška koji je prethodno transformiran matricom $W(x;\Delta p)$, dok se u algoritmu Lucas-Kanade trensformira piksel koji nije prethodno transformiran za Δp.

Kod kompozicijskog algoritma se na kraju iteracije parametri mijenjaju prema ovom izrazu[4]:

$$W(x;p) \longleftarrow W(x;p) \circ W(x;\Delta p). \tag{13}$$

Točnije rečeno, mijenja se cijela matrica transformacije koja kao parametar x prima samu sebe, sa parametrom Δp.

Vremenska složenost ovog algoritma iznosi $O(n^2 N + n^3)$. Kompozicijski algoritam u pred-iteraciji računa Jakobijeva matrica $\frac{\partial W}{\partial p}$, a ostale faze su mu iste kao i kod algoritma Lucas-Kanade. Za više informacija o algoritmu pogledati u [4], str. 9.

2.2.2Inverzni kompozicijski algoritam

Najvažnija razlika ovog algoritma u odnosu na prošla dva je u tome što se u ovom algoritmu Hesseova matrica računa prije faze iteracija, a zatim se rabi u fazama iteracija. Zbog toga je ostvarena značajna vremenska ušteda. Predložak i izvorna slika sada su

zamijenili svoje uloge i izvorna slika se modificira kako bi bila što sličnija predlošku. Cilj algoritma je minimizirati kvadrat razlike intenziteta predloška i izvorne slike[4]:

$$\sum_x \left[T(W(x;\Delta p)) - I(W(x;p))\right]^2 . \tag{14}$$

Isto kao i u običnom kompozicijskom algoritmu uzima se piksel na mjestu pomaknutom za Δp i on se uspoređuje sa pikselom izvorne slike koji je transformiran za p od svog početnog položaja.

Mijenjanje parametara odvija se inverzno, odnosno matrica transformacije se invertira[4]:

$$W(x;p) \longleftarrow W(x;p) \circ W(x;\Delta p)^{-1} \tag{15}$$

Vremenska složenost ovog algoritma je $O(nN + n^3)$ zbog toga što se Hesseova matrica računa samo jednom i to prije početka iteracija. Isto tako, prije početka iteracije računa se Jakobijeva matrica $\frac{\partial W}{\partial p}$, gradijent predloška ∇T i njihov umnožak $\nabla T \frac{\partial W}{\partial p}$. Inverzni algoritam može obraditi manji skup slika nego algoritam Lukas-Kanade jer osim što što matica transformacije $W(x;p)$ mora biti diferencijabilna, ona mora imati svoj inverz $W(x;p)^{-1}$. Za više informacija o ovom algoritmu pogledati u [4], str. 13.

2.2.3 Inverzni aditivni algoritam

Ovaj algoritam ima isti cilj kao i algoritam Lucas-Kanade, minimizirati kvarat razlike intenziteta slike i preloška[4]:

$$\sum_x \left[I(W(x;p+\Delta p)) - T(x)\right]^2 . \tag{16}$$

Kada se izraz (16) linearizira dobiva se:

$$\sum_x \left[I(W(x;p)) + \nabla I \frac{\partial W}{\partial p} \Delta p - T(x) \right]^2 . \tag{17}$$

Sada se uloga predloška i izvorne slike mora zamijeniti:

$$I(W(x;p)) \approx \nabla T. \tag{18}$$

Derivacijom izraza (18) po varijabli x dobiva se:

$$\nabla I \frac{\partial W}{\partial x} \approx \nabla T. \tag{19}$$

Izraz (19) se uvrštava u izraz (17) i zamjenjuje se uloga predloška i slike primjenom izraza (18):

$$\sum_{x}\left[T(x)+\nabla T\left(\frac{\partial W}{\partial x}\right)^{-1}\frac{\partial W}{\partial p}\Delta p - I(W(x;p))\right]^2. \tag{20}$$

Parametri se u svakoj iteraciji mijenjaju na ovaj način:

$$p \longleftarrow p-\Delta p. \tag{21}$$

Prije iterativnog dijela algoritma, najprije se izračunavaju gradijent predloška ∇T, matrica $\Gamma(x)$ koja ima sličnu funkciju kao Jakobijeva matrica (za jednadžbu matrice $\Gamma(x)$ pogledati [4], str. 19). Vremenska složenost ovog algoritma je najčešće $O(n^2N+n^3)$. Inverzni aditivni algoritam se može primjenjivati na vrlo mali skup slika, budući da matrica transformacije mora biti diferencijabilna po varijablama x i p pri računanju matrice $\Gamma(x)$ [4].

3 Precizno lociranje prometnih znakova trokutnog oblika

Cilj programa je dohvatiti nijanse boje iz crvenih trokuta skupa prometnih znakova i napraviti histograme nijansi, zasićenja i osvjetljenja tih boja. Kako bi to bilo moguće, potrebno je precizno odrediti položaj crvenog trokuta znaka. To se postiže algoritmom gradijentne optimizacije. Postoji više vrsta algoritama gradijentne optimizacije, a ovdje je implementiran algoritam Lucasa i Kanadea. Osim kvantifikacije varijacija boja pomoću histograma ovim programom može se dodatno testirati detektor znakova koji tipično daje puno neispravnih detekcija. Ulazi programa su izlazi detektora Viole i Jonesa koji detektira 99,5% znakova, ali zato u prosjeku daje tri krive detekcije za svaki znak, gdje se svaki znak detektira pet ili više puta. [6]. Ovim programom testira se uspješnost detektora Viole i Jonesa. Računalo za daljnju obradu rezultata programa za poravnavanje slike s predloškom znakova će zaključiti da se radi o pogrešnoj detekciji po anomalijama u histogramu. Program za poravnavanje slike s predloškom moguće je nadograditi tako da odbacuje pogrešno detektirane znakove. Pogrešno detektirani znakovi su oni za koje bi algoritam gradijentne optimizacije izrazito brzo divergirao jer bi se predložak i pogrešno detektirani znak potpuno razlikovali. Kod pogrešno detektiranog znaka će u histogramu biti mnogo vrijednosti nijansi (*hue*) koje nisu oko nule ili oko 180. Takve vrijednosti predstavljaju boje koje nisu crvene pa je iz toga očito da u pogrešno detektiranom znaku nema crvenog trokuta pa ni znaka. Program za poravnavanje slike s predloškom može se integrirati zajedno sa programom za detekciju znakova u jednu cjelinu. Takav hibrid bi uz detekciju znakova omogućavao i njihovo precizno lociranje. Također, program za poravnavanje slike s predloškom mogao bi se koristiti kao pomoćni modul u postupcima raspoznavanja uzoraka. Postupci raspoznavanja uzoraka posebno su osjetljivi na "nesavršen" izgled znaka uslijed krivog postavljanja ili perspektivnog izobličenja. Kada bi se uspjeli detektirati i ispraviti takvi negativni efekti, postupci raspoznavanja uzoraka bili bi znatno uspješniji. Za verifikaciju prometne signalizacije također se može koristiti program za poravnavanje slike s predloškom[7]. Verifikacija prometne signalizacije temelji se na bazi znanja koja sadrži informaciju o tome koji znak se treba nalaziti na određenom mjestu u prometu. Vozilo u prometu snima znakove i svaki detektirani znak uspoređuje sa korespondentnim znakom u bazi znanja. Ako usporedba rezultira konvergencijom sa dovoljno malim rezidualom tada se može reći da je na tom mjestu pravilan znak. Pri verifikaciji prometne signalizacije potrebno je koristiti varijantu postupka sa specifičnim predloškom. Specifičan predložak također se koristi pri kvantifikaciji boje u skupu znakova određenog tipa. Program za poravnavanje slike s predloškom može se modificirati u program za određivanje tipa znaka. Specifični predlošci iz baze predložaka uspoređivali bi se sa ulaznim znakom i vratio bi se onaj predložak čija je korelacija sa ulaznim znakom najveća.

4 Precizno lociranje prometnih znakova trokutastog oblika

Program je implementiran u programskom jeziku C++. Za razvoj programa korišten je alat Microsoft Visual Studio 2008.

Precizno lociranje znakova implementirano je na način da se prvo parsira ulazna tekstualna datoteka koja sadrži informacije o položaju znakova na slici. Informacije o znakovima se zatim spremaju u memoriju za kasnije korištenje. Kako bi se uštedilo na računalnim resursima uzima se samo dio slike gdje se znak nalazi i dio oko znaka. Radi boljeg provođenja algoritma gradijentne optimizacije sliku je moguće binarizirati i zagladiti. Slika je binarizirana crveno-bijelim kontrastom, pretvorena u crno-bijelu i na kraju zaglađena. Crno-bijela binarizirana slika obrađuje se algoritmom gradijentne optimizacije koji ju uspoređuje sa predloškom i pokušava promijeniti predložak da što više bude nalik slici. Predložak je prekriven maskom zato da se ne obrađuje područje predloška koje nije bitno. Algoritam kao rezultat daje matricu transformacije. Matricom transformacije trasformira se predložak kojeg smo stavili na sliku tako da se on točno preklapa sa slikom. Svi pikseli na slici koji nisu u trokutu predloška zanemaruju se, a nad ostalima se izrađuje histogram. Algoritam preciznog lociranja znakova nalazi se u datoteci `alg_locateSigns.cpp`.

4.1 Priprema za implementaciju

Kako bi se olakšala implementacija i povećala apstrakcija, korištene su dodatne biblioteke *boost* i *OpenCV*. Biblioteka *boost* sadrži funkcije u direktoriju `regex` kojima se parsira tekst pomoću regularnih izraza. Biblioteka *OpenCV* sadrži širok spektar funkcija koje se koriste u računalnom vidu. Od posebne koristi su funkcije za množenje matrica, izračunavanje gradijenata, promjenu veličine slike i mnoge druge.

Projekt treba smjestiti u direktorij `msvc_zavrad\zavrad`. Za prevođenje ljuske potrebne su biblioteke *boost* [9], *Windows Media Format SDK* [10] i *OpenCV*[11]. Biblioteke je potrebno instalirati na lokacije: `C:\boost_1_39_0`, `C:\WMFSDK11` i `C:\ProgramFiles\OpenCV`. Informacije o ručnoj instalaciji i načinu korištenja biblioteka *boost* moguće je pogledati na [9] i [11]. Biblioteka *WMF* u direktoriju `C:\WMFSDK11` koristi se za čitanje formata .wmv, a vfw32 omogućuje čitanje formata .avi[11]. Potrebno je instalirati dodatne *boost* biblioteke[12] i kopirati `libboost_regex-vc90-mt-gd-1_38.lib` i `libboost_regex-vc90-mt-1_38.lib` u direktorij gdje se nalazi projekt.

Kreirani projekt otvara se odabirom File → Open → Project/Solution. Putanja do projekta je `C:\...\msvc_zavrad\zavrad\nazivProjekta.sln` (Slika 2).

Gorenavedene biblioteke moraju se uključiti u projekt. Desnim klikom na projekt prikazuje se padajući izbornik u kojem je potrebno odabrati opciju Properties(Slika 3).

Uključivanje kazala sa sučeljima dodatnih biblioteka obavlja se pozicioniranjem na Configuration Properties →C/C++ →General →Aditional Include Directories (Slika 4) i upisivanjem sljedećih putanja:

C:\Program Files\OpenCV_\cvaux\include

C:\Program Files\OpenCV_\cxcore\include

C:\Program Files\OpenCV_\cv\include

C:\Program Files\OpenCV_\otherlibs\highgui

C:\Program Files\OpenCV_\otherlibs\cvcam\include

C:\boost_1_38_0

C:\Program Files\Microsoft SDKs\Windows\v6.0A

Uključivanje dodatnih biblioteka obavlja se pozicioniranjem na Configuration Properties →Linker →Input →Aditional Dependencies (Slika 5) i upisivanjem sljedećih putanja:

„C:\Program Files\OpenCV_\lib\cv.lib“

„C:\Program Files\OpenCV_\lib\cxcore.lib“

„C:\Program Files\OpenCV_\lib\cvaux.lib“

„C:\Program Files\OpenCV_\lib\highgui.lib“

„C:\Program Files\Microsoft SDKs\Windows\v6.0A\Lib\wmvcore.lib“ vfw32.lib

Slike koje će se obrađivati moraju biti smještene u direktoriju `source/`. Za algoritam su također potrebni predložak znaka i maska koja određuje koji pikseli predloška se obrađuju u algoritmu gradijentne optimizacije, a koji ne. Tekstualnu datoteku u kojoj su smještene informacije o znakovima treba smjestiti u direktorij u kojem se nalazi projekt.

4.2 Parsiranje ulazne datoteke

Algoritmu za precizno lociranje znakova potrebne su slike koje su snimljene iz vozila koje se kreće cestom. Na svakoj slici nalazi se jedan ili više znakova. U tekstualnoj datoteci sadržane su informacije o znakovima. Zaglavlje tekstualne datoteke sadrži sve slike koje se nalaze u datoteci. Nakon zaglavlja slijedi lista slika gdje svaki redak predstavlja jednu sliku. Format retka izgleda ovako:

[A22_0005.bmp]:A22@(x=609, y=203, w=58,h=61)&A20@(x=598, y=265, w=62, h=58)

Oznaka "A22_0005.bmp" predstavlja naziv slike. Nakon graničnika ":" navedeni su znakovi na slici koji su razvojeni graničnikom "&". Format znaka započinje njegovim nazivom, na primjer "A22". Graničnik "@" označava početak zapisa parametara znaka. Koordinate x i y označavaju gornji lijevi kut pravokutnika u kojem je smješten znak, a w i h označavaju širinu i visinu pravokutnika. Gorenavedeni primjer opisuje sliku "A22_0005.bmp" koja sadrži znakove "A22" i "A20". Znak "A22" je omeđen pravokutnikom širine 58 piksela i visine 61 piksel čije su koordinate 609 i 203. Implementacija parsiranja nalazi se na početku glavnog programa koji se nalazi u datoteci `alg_locateSigns.cpp`. Za parsiranje tekstualnih datoteka koristi se klasa `boost::regex`. Parsirane slike pohranjuju se u klasu Slika, smještenu u datoteci `slika.hpp` . Klasa Slika definirana je kao:

```
public class slika
{
private:
     static const int MAX_BR_ZNAKOVA_SLIKE=10;
     std::string imeSlike;
     int brZnakova;
     int indeksSlike;
public:
     int getIndeksSlike();
     int getBrZnakova();
     char* slika::getIme();
     void setZnak(int indeksZnaka, char* oznakaZnaka, int x,
     int y, int sirina, int visina);
     slika(char* imeSlike_, int indeksSlike_);
     slika(void);
     virtual ~slika(void);
public:
     znak *znakoviSlike[MAX_BR_ZNAKOVA_SLIKE];
};
```

Svaka slika ima svoj indeks koji odgovara rednom broju pripadajućeg retka u tekstualnoj datoteci. Isto tako slika ima svoje ime čija je duljina ograničena na `MAX_DULJINA_IMENA` i svoje znakove čiji je broj ograničen na

`MAX_BR_ZNAKOVA_SLIKE`. Navedene varijable su privatne i mogu se dohvatiti pomoću pripadajućih javnih metoda.

Znakovi slike pohranjuju se u klasu znak koja je definirana kao:

```
public class znak {

public:
     string visinaTmplToString();
     string sirinaTmplToString();
     char* getOznaka();
     int getX();
     int getY();
     int getSirina();
     int getVisina();
     znak();
     znak(char* oznaka_, int x, int y, int sirina, int
visina_);
     virtual ~znak();
private:
     std::string oznaka;
     int x;
     int y;
     int sirina;
     int visina;
 };
```

Znak ima gornje lijevu koordinatu pravokutnika koji ga omeđuje, njegovu širinu i visinu. Varijable znaka su privatne pa su definirane javne metode koje ih dohvaćaju.

Pročitani redak parsira se pomoću funkcije `strtok(char*str, const char* delim)` koja uzima znakovni niz i vraća dio niza do graničnika `delim`. Pravilna sintaksa znakovnih nizova provjerava se funkcijom `boost::regex_match(char* znak_, boost::cmatch matches, boost::regex re)` koja uzima znakovni niz `znak_` kojeg će provjeravati regularnim izrazom `re`.

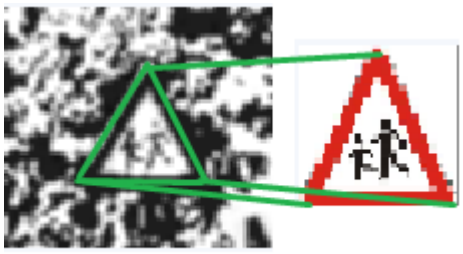
4.3 Pripremanje slike za algoritam gradijentne optimizacije

Kako bi se nad slikom mogao provesti algoritam gradijentne optimizacije potrebno ju je binarizirati u crveno-bijelu sliku. Binariziranu sliku potrebno je pretvoriti u crno-bijelu i nakon toga ju zagladiti. Ako se ne provedu ova tri koraka, algoritam vrlo često divergira i ne daje dobre rezultate. Uz navedena tri koraka pretvorbe slike, potrebno je imati pravilni predložak i pravilnu masku. Pravilni predložak je onaj koji ima ravnomjerno raspoređenu debljinu crvenog okvira, sliči jednakostraničnom trokutu i jednakog je tipa kao i znak na slici(Slika 6). Predlošci se izvlače iz baze znakova koja sadrži 48 različitih tipova trokutastih znakova. Pravilna maska je ona koja prekriva crnom bojom područje manjeg trokuta unutar trokuta i dijelove izvan trokuta (Slika 7).

Slika 1 Predložak koji označava znak za skretanje ulijevo

Slika 2 Maska za za predložak

Konvergencija algoritma također ovisi o početnom položaju i veličini predloška naspram slike. Predložak se treba smjestiti tako da njegovo središte bude jednako središtu znaka i da mu veličina bude 90% veličine znaka(Slika 8).



Slika 3 Položaj predloška na slici

Metoda koja pretvara izvornu sliku u crveno-bijelu nalazi se u datoteci `binaryRed.cpp` i definirana je kao:

```
void binaryRed(IplImage* pImgI, IplImage* pImgI_Dest,
char opcijaPrikaza);
```

Binarizacija slike u crveno-bijelu sliku temelji se na funkciji koja određuje je li neki piksel crveni ili nije. Ako je funkcija veća od nule piksel je crven, inače se pretvara u bijelog. Funkcija određuje je li udio crvene boje u pikselu veći nego udio plave i zelene zajedno. Definicija funkcije je:

$$f(x) = R(x) - G(x) - B(x) + D. \tag{22}$$

Varijabla x je piksel, a funkcije $R(x)$, $G(x)$ i $B(x)$ vraćaju vrijednost crvene, zelene, odnosno plave komponente piksela. Konstanta D označava mjeru strogoće kriterija kojega piksel treba zadovoljiti kako bi se proglasio crvenim. Što je konstanta D manja to piksel treba biti crveniji da bi se proglasio crvenim. Algoritam binariziranja odvija se iterativno. U svakoj iteraciji konstanta D se povećava za jedan počevši od -30. Kako se konstanta D povećava sve više piksela zadovoljava kriterij za crvenu boju. Iteracije se odvijaju sve dok udio crvenih piksela na slici ne dosegne 50%.

Pretvorba slike iz RGB modela u sivu sliku temelji se na formuli za pretvorbu[12]:

$$Y(x) = 0.3R(x) + 0.59G(x) + 0.11B(x). \tag{23}$$

Funkcija $Y(x)$ označava intenzitet piksela x na sivoj slici. Funkcije $R(x)$, $G(x)$ i $B(x)$ imaju jednaku ulogu kao u izrazu (22). Implementacija pretvorbe modela boje slike iz jednog u drugi ostvarena je funkcijom:

```
cvCvtColor(const CvArr* SourceImage, const cvArr*
DestinationImage, int code);
```

Varijabla `int code` označava tip pretvorbe. U ovom slučaju njena vrijednost je `CV_RGB2GRAY`. Izraz (23) ne predstavlja jedinu funkcija pretvorbe, ali se najčešće koristi[12].

Uklanjanje šuma postiže se glađenjem slike. Šum može naškoditi performansama programa jer vodi algoritam gradijentne optimizacije u krivom smjeru što može dovesti do divergencije algoritma. Glađenje se postiže konvolucijom matrice veličine 3x3 sa slikom. Matrica prolazi kroz sve piksele slike. Za piksel na slici koji se preklapa sa sredinom matrice računa se nova vrijednost. Ona se dobije izračunavanjem aritmetičke sredine vrijednosti piksela koji okružuju središnji piksel. Implementacija glađenja ostvarena je funkcijom[13]:

```
void cvSmooth(const CvArr* src, CvArr* dst,
              int smoothtype=CV_GAUSSIAN,
              int param1=3, int param2=0, double param3=0,
double param4=0 );
```

Varijable `param#` označavaju standardne devijacije σ, odnosno rang matrice konvolucije. Za više informacija o parametrima funkcije pogledati na [13].

4.4 Algoritam gradijentne optimizacije

Najvažniji dio algoritma za precizno lociranje znakova je algoritam gradijentne optimizacije koji nastoji minimizrati kvadrat razlike intenziteta slike i predloška prema izrazu (2). U ovom radu implementiran je algoritam Lucas-Kanade. Funkcija koja implementira algoritam nalazi se u datoteci `forwardsAdditive.cpp` i njena deklaracija glasi:

```
void align_image_forwards_additive(IplImage* pImgT, IplImage* maska, CvRect template_rect, IplImage* pImgI, CvMat* W, char opcijaPrikaza);
```

Parametar `IplImage* pImgT` označava predložak, a parametar `IplImage* maska` masku koja prekriva dijelove predloška koji nisu bitni i koji se neće obrađivati. `CvRect template_rect` je pravokutnik koji omeđuje predložak i poveznica je između koordinatnog sustava slike i predloška. Izvorna slika deklarirana je kao `IplImage* pImgI`. Matrica transformacije `CvMat* W` je izlaz algoritma i ona predstavlja način na koji će se predložak podesiti u odnosu znak. Opcija prikaza definira odabir hoće li se rezultati prikazivati na ekranu ili neće.

Algoritmi gradijentne optimizacije sadrže dva dijela: prediteracijski i iteracijski. Poželjno je obaviti što je više moguće posla u prediteracijskom dijelu jer se taj posao obavi samo jednom, dok bi se u iteracijskom dijelu obavljao više puta.

Prije izvođenja algoritma potrebno je definirati maksimalni broj iteracija i prag sličnosti. Prag sličnosti je vrijednost koja uvjetuje nastavak algoritma. Ako su predložak i slika najviše moguće slični, njihova promjena parametara Δp biti će ispod praga sličnosti i algoritam će stati sa izvođenjem. Što je prag sličnosti manji, to će maksimalni broj iteracija morati biti veći ako se želi postići podudaranje slike i predloška. Pri malom maksimalnom broju iteracija, prag sličnosti će morati biti veći kako bi algoritam što brže iskonvergirao i našao znak. Prije samog početka iterativnog dijela računaju se gradijenti u smjeru x i y osi pomoću funkcije:

```
cvSobel(IplImg* pImgI, IplImg*  pGradIx, int xDir, int yDir, int apertureSize);
```

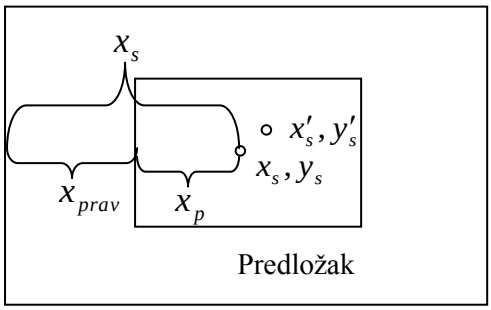
Gradijent u smjeru x osi računa se tako da se za svaki piksel izvorne slike (osim onih na rubovima) računa razlika između x koordinata njega i njegovog susjeda. Isti princip vrijedi i za gradijent u smjeru y osi.

Kao rezultat algoritma dobije se matrica transformacije $W(x;p)$ i ona je definirana kao:

$$W(x;p) = \begin{pmatrix} s_x & -w_z + h & t_x \\ w_z + g & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{24}$$

Matrica W omogućuje skaliranje predloška u smjeru x i y osi pomoću varijabli s_x i s_y, rotaciju pomoću varijable w_z, transformaciju pomoću varijabli t_x i t_y, te smik pomoću varijabli h i g.

Iterativni dio temelji se na prolasku kroz predložak i modifikaciji matrice transformacije. Kako bi bila moguća usporedba piksela predloška i slike, potrebno je na temelju koordinatepredloška saznati koordinatu slike. Koordinata slike dobije se zbrojem koordinate predloška i koordinate gornjeg lijevog kuta pravokutnika koji omeđuje predložak(Slika 9):



Slika

Slika 9 Odnos koordinata predloška i slike

Dobivene koordinate slike $x = (x_s, y_s)$ transformiraju se matricom matricom $W(x; p)$ u koordinate $x' = (x'_s, y'_s)$. Za svaki piksel predloška računa se razlika D:

$$D = I(W(x; p)) - T(x). \tag{25}$$

Funkcija $I(W(x; p))$ računa intenzitet u transformiranim koordinatama slike x', a funkcija $T(x)$ računa intenzitet u koordinatama predloška $x = (x_p, y_p)$.

Sljedeći korak algoritma izračunava slike takozvanog "najstrmijeg pada"(*steepest descent*). Broj slika "najstrmijeg pada" jednak je broju parametara, a za njihov izračun potrebno je znati derivaciju matrice transformacije po svim njenim parametrima. Derivacija matrice transformacije u izrazu (24) po parametrima p glasi:

$$\frac{\partial W}{\partial p} = \begin{pmatrix} x_s & 0 & -y_s & 1 & 0 \\ 0 & y_s & x_s & 0 & 1 \end{pmatrix}. \tag{26}$$

Piksel slike "najstrmijeg pada" dobije se umnoškom stupca matrice $\frac{\partial W}{\partial p}$ koji odgovara rednom broju slike i gradijenta slike u točki $x'' = (x_p, y_p)$, $\nabla I(x'')$. Na primjer, piksel prve

slike "najstrmijeg pada" na lokaciji x'' dobije se kao $\nabla I(x'')\begin{pmatrix} x_s \\ 0 \end{pmatrix}$, piksel druge slike kao $\nabla I(x'')\begin{pmatrix} 0 \\ y_s \end{pmatrix}$ itd. Izraz (8) definira slike "najstrmijeg pada" u općenitom obliku.

Vrijednost Hesseove matrice dobije se umnoškom svake slike "najstrmijeg pada" sa svakom što je opisano izrazom (9). Inverz Hesseove matrice implementiran je funkcijom:

```
double cvInvert( const CvArr* src, CvArr* dst, int method=CV_LU );.
```

Konačno, promjena parametara definirana je izrazom :

$$\Delta p = H^{-1}\sum_{x}\left[\nabla I \frac{\partial W}{\partial p}\right]^{\tau} D. \qquad (27)$$

Izraz (27) ekvivalentan je izrazu (10). Parametar D opisan je izrazom (25). Faktor H^{-1} je inverz Hesseove matrice, a faktor $\sum_{x}\left[\nabla I \frac{\partial W}{\partial p}\right]^{\tau}$ predstavlja slike "najstrmijeg pada".

Iterativni dio algoritma odvija se sve dok parametar Δp nije manji od granice sličnosti ε. Na kraju algoritma izmjereno je vrijeme izvođenja algoritma i inicijalizira se matrica tranformacije na temelju parametara $p = (s_x, s_y, w_z, t_x, t_y)$.

4.5 Izrada HSV histograma

Kada je poznata matrica transformacije $W(x; p)$ lako je moguće locirati položaj znaka. Znak se locira na način da se predložak koji se preklapa sa slikom (Slika 8) pomnoži sa $W(x; p)$. Mjesto gdje se slika preklapa sa trokutaskim rubom transformiranog predloška oboja se u crvenu boju(Slika 10). Funkcija koja implementira crtanje trokuta nalazi se u datoteci `drawTriangle.cpp` i deklarirana je kao:

```
void drawTriangle(IplImage* pImgSrc, CvMat* W, CvRect omega,
IplImage* pImgDest, IplImage* pImgTempl, float step);
```

Funkcija za svaki crveni piksel predloška traži pripadajući piksel u slici, trenformira ga i označi crvenom bojom. Varijabla `float step` označava koliko jedan piksel predloška ima pripadajućih piksela u slici. Ako, na primjer, postavimo vrijednost varijable `float step` na 0.1 to znači da svaki piksel ima deset pripadajućih piksela na slici. Razlog postojanja te varijable je u tome što je predložak manji od slike pa se jedan piksel predloška preslikava u više piksela na slici.

Slika 10 Znak sa označenim trokutom

Vrijednosti koje će se prikazivati u histogramu su HSV vrijednosti onih piksela koji se nalaze unutar crvenog trokuta. Pikseli koji se ne nalaze unutar crvenog trokuta zanemaruju se, odnosno pretvaraju u bijelu boju(Slika 11). Funkcija koja pretvara nebitne piksele u bijelo smještena je u datoteku `excludeIrrelevantPix.hpp` i deklarirana kao:

```
void excludeIrrelevantPix(IplImage* pImgSrc, IplImage*
pImgRedTriangle, IplImage* pImgDest);
```

Slika 11 Izolirani crveni trokut znaka

Područje koje nije označeno bijelom bojom razlaže se na tri komponente: nijansu(*hue*), zasićenje(*saturation*) i osvjetljenje(*lightness*/*value*). Vrijednosti nijansi kreću se od 0 do 180, gdje vrijednosti oko nule i oko 180 predstavljaju crvenu boju. Vrijednosti zasićenja i osvjetljenja kreću se od 0 do 255. Zasićenje predstavlja gustoću boje. Malo zasićenje predstavlja sivu boju, a veliko predstavlja jarke boje. Kada osvjetljenje ne postoji ili je maksimalno boja će biti crna odnosno bijela, bez obzira na ostale komponente. Razlaganjem slike na HSV komponente dobiju se tri slike od kojih svaka predstavlja pojedinu komponentu. Kako bi se raspodjela vrijednosti pojedinih komponenta mogla memorirati postoje tri polja (`int* vr_histograma_H, int* vr_histograma_S, int* vr_histograma_V`) u kojima se pojedini član povećava za jedan uvijek kada se naiđe na vrijednost piksela koja odgovara indeksu člana polja. Na primjer, ako se tokom prolaska kroz sliku komponente nijansi(*hue*) naiđe na vrijednost vr, tada će se član `vr_histograma_H[`vr`]` povećati za jedan. Kada se sve slike obrade, vrijednosti triju polja predaju se funkciji za crtanje histograma. Funkcije za

crtanje i računanje vrijednosti histograma nalaze se u datoteci `histogram.hpp` i deklarirane su kao:

```
void  drawHistogram(int* vr_histogramaH, int* vr_histogramaS,
int* vr_histogramaV);
void calcHistogram(IplImage* pImgI, int* vr_histograma_H,
int* vr_histograma_S, int* vr_histograma_V);.
```

5 Eksperimentalni Rezultati

Eksperimentalni rezultati obično se klasificiraju u tri skupine: ispravni rezultati, rezultati u kojima postupak nalazi objekte koje nismo tražili (*false positive errors*) i rezultati u kojima postupak ne nalazi tražene objekte(*false negative errors*)[14]. Pretpostavljena vrijednost praga sličnosti ε iznosi 0.005, a maksimalnog broja iteracija 1000. Kod znakova koji nisu zarotirani i dobrog su osvjetljenja algoritam gradijentne optimizacije iskonvergira u otprilike dvostruko manje iteracija nego u slučaju zarotiranih i/ili loše osvjetljenih znakova. Smanjivanjem praga sličnosti povećavaju se šanse za precizno lociranje znaka, ali se također povećavaju šanse za divergenciju algoritma. Ako algoritam divergira to može značiti i da će postupak locirati znak tek nakon maksimalnog broja iteracija. Povećavanjem praga sličnosti dobiva se suprotan efekt. Slika 12 pokazuje rezultat lociranja znaka za $\varepsilon = 5*10^{-3}$ i za $\varepsilon = 5*10^{-5}$. Pošto se za dvije različite vrijednosti ε dobije jednaki rezultat, prag sličnosti nije dobro previše smanjivati jer algoritam postaje sporiji. Za $\varepsilon = 5*10^{-3}$ algoritam iskonvergira u 73 iteracije (0.686 sekundi), a za $\varepsilon = 5*10^{-5}$ algoritam ne iskonvergira niti u 10000 iteracija (99 sekundi). Brzina detekcije također ovisi o veličini znaka. Veći znakovi se sporije detektiraju od manjih. Kvaliteta konvergencije ne ovisi o veličini jer se veličina predloška modificira u skladu sa veličinom znaka. Ako je znak perspektivno iskrivljen ili premalo/previše osvijetljen, moguća je slabija kvaliteta konvergencije.

5.1 Ispravni rezultati

Ispravan rezultat očituje se preklapanjem jarko crvenog trokuta sa crvenim trokutom znaka. Algoritam gradijentne optimizacije za 80% znakova iz direktorija `\source` dobro detektira crveni rub znaka. Dobra detekcija događa se kada je znak sličan predlošku (ima približno iste kuteve rubnog trokuta kao i predložak) te se ne nalazi u jako tamnoj okolini (Slika 12). Isto tako, znak ne smiju prekrivati nikakvi objekti.

Slika 12 Dobro osvjetljen znak i njegov predložak za vrijednosti praga $\varepsilon=5*10^{-3}$ i $\varepsilon=5*10^{-5}$

Znakovi snimljeni u tamnijoj okolini također se mogu dobro detektirati (Slika 13). Rotacija znaka ne utječe na kvalitetu detekcije.

Slika 13 Lošije osvjetljen zarotirani znak i njegov predložak

5.2 Neispravna konvergencija algoritma

Loša konvergencija zabilježena je u 20% znakova u direktoriju \source. U tim slučajevima algoritam gradijentne optimizacije divergira. Znakovi za koje je zabilježena divergencija perspektivno su iskrivljeni u odnosu na predložak (Slika 14), loše su osvjetljeni (Slika 15) ili ih jednim dijelom prekriva neki drugi objekt (Slika 16).

Slika 14 Perspektivno izobličeni znak i njegov predložak

Znak na slici 14 je perspektivno izobličen i postavljen je ukoso pa se dosta razlikuje od izvornog predloška. Njegova izobličenost razlog je divergenciji algoritma poravnavanja slike sa predloškom.

Slika 15 Loše osvjetljeni znak sa "dvosmislenim" crtežom stijene

Osim lošeg osvjetljenja drugi uzrok divergencije je vrlo specifičan. Radi se o crtežu stijene sa odronom koja se nalazi unutar znaka. Unutar nacrtane stijene na znaku može se pronaći

manja stijena istog oblika. Algoritam poravnavanja slike sa predloškom će prije naići na manju stijenu nego na veću pa će pomisliti da je upravo znak sa manjom stijenom traženi znak. Takav problem može se nazvati problem "dvosmislenog" objekta koji u sebi ima manji objekt istog oblika.

Slika 16 Znak prekriven ogradom

Svaki pogrešan rezultat postupka poravnavanja slike sa predloškom može se klasificirati kao *false negative* ili *false positive*, ovisno o interpretaciji. Klasifikacija se može opisati na primjeru iz stvarnog života. Primjer *false negative* rezultata bio bi test koji pokazuje da žena nije trudna, a zapravo jest. Analogno tome, *false positive* rezultat bio bi test koji pokazuje da je žena trudna, a zapravo nije. Moguće su dvije interpretacije rezultata sa slika 14, 15 i 16. Rezultati na slikama su tipa *false negative* jer postupak nije pronašao znak koji je trebao pronaći iako se on nalazi na slici. Isto tako, rezultati na slikama su tipa *false positive* jer je postupak pronašao znak (onaj označen crvenim trokutom), a on se ne nalazi na slici.

5.3 Analiza histograma

Rezultati vidljivi iz triju histograma u skladu su s očekivanjima. Histogrami prikazuju HSV komponente izračunate na skupu slika koje se nalaze u direktoriju \source. Vrijednosti nijansi (*hue*) grupirane su oko nule i oko 180 što govori da je većina boje u rubima znakova crvena(Slika 17). Pojavljivanje ostalih nijansi boja uzrok je loše detekcije znakova. Zelena boja pojavljuje se u histogramu nijansi zato jer je algoritam lociranja znakova detektirao znak na mjestu gdje se nalazi flora oko znaka. Histogram zasićenja (*saturation*) pokazuje kako je crvena boja unutar znaka izrazito gusta i prepoznatljiva iz razloga što prometni znakovi moraju biti uočljivi. Pojavljivanja vrijednosti slabog zasićenja također su razlog loše detekcije. Boje okoline znaka nisu toliko guste i izražene kao boje prometnih znakova. Osvjetljenje (*value*/*lightness*) izračunato nad skupom slika ravnomjerno je raspoređeno jer se slabije i bolje osvjetljene slike pojavljuju u podjednakom omjeru. Vrijednosti zasićenja i osvjetljenja kreću se od nule do 255.

Slika 17 Histogram nijansi boja

Slika 18 Histogram zasićenja

Slika 19 Histogram osvjetljenja

5.4 Rezultati različitih varijanti osnovnog postupka

Osim osnovnog postupka za poravnavanje slike sa predloškom, mogu se koristiti i njegove različite varijante. Tako osnovni postupak može koristiti binarizaciju izvorne slike, univerzalni predložak i/ili masku.

5.4.1 Varijanta sa korištenjem binarizacije slike

Binarizacija slike pomaže u slučaju kada je slika tamnija i/ili kada je unutrašnjost znaka žute boje (Slika 20). U tom slučaju, binarizacija raščišćava sliku koja je tamna i jasno definira crveni rub znaka koji je na originalnoj slici teže uočljiv. Isto tako, binarizacija može imati negativne efekte jer se njome gube informacije sadržane u originalnoj slici. Kada znak na slici nije taman niti ima žutu unutrašnjost, korištenjem binarizacije će algoritam poravnavanja slike i predloška sporije konvergirati. Ako je znak na slici taman i/ili ima žutu unutrašnost, algoritam poravnavanja slike sa predloškom će korištenjem binarizacije brže konvergirati jer mu je tada lakše prepoznati znak. Na temelju eksperimenata je pokazano da je binarizacija većinom korisna zbog nesavršenosti ulaznih slika. Originalna slika uvijek oslikava realne vrijednosti piksela dok ih binarna slika iskrivljuje.

a) Znak za nagib, postupak sa binarizacijom

b) Znak za nagib, postupak bez binarizacije

c) Znak opasnosti nailaska jelena, postupak sa binarizacijom

d) Znak opasnosti nailaska jelena, postupak bez binarizacije

Slika 20 Rezultati algoritma sa i bez binarizacije

5.4.2 Varijanta sa korištenjem univerzalnog predloška

Korištenje univerzalnog predloška umjesto predloška specifičnog tipa je generalno lošiji izbor (Slika 21). Predložak specifičnog tipa sadrži dodatne informacije o znaku. Svaki specifični predložak sadrži njemu specifičan crtež u sredini. Ti crteži pomažu algoritmu poravnavanja slike sa predloškom da ne divergira ili da brže iskonvergira. Eksperimenti sa univerzalnim predloškom koriste se pri verifikaciji rezultata detektora koji generira gomilu netočnih detekcija. U tom slučaju, zbog velikog broja netočnih detekcija, nije moguće koristiti specifični preložak jer nije moguće sa velikom sigurnošću odrediti o kojem se tipu znaka radi. Univerzalni predložak može se koristiti u postupku poravnavanja prije detekcije, kada je tip znaka nepoznat.

a) Znak za sporednu ulicu, postupak sa univerzalnim predloškom

b) Znak za sporednu ulicu, postupak sa specifičnim predloškom

c) Znak za oštri zavoj postupak sa univerzalnim predloškom

d) Znak za oštri zavoj postupak sa specifičnim predloškom

Slika 21 Rezultati algoritma sa univerzalnim i specifičnim predloškom

5.4.3 Varijanta bez korištenja maske

Uloga maske je zanemarivanje elemenata predloška koji nisu relevantni za algoritam gradijentne optimizacije. Pikseli predloška koji se zanemaruju su oni koji se poklapaju sa crnim pikselom maske. Maska uvelike poboljšava performanse algoritma poravnavanja slike sa predloškom. Bez maske se u obzir uzimaju nevažni bijeli pikseli koji se nalaze daleko od crvenog ruba. Takvi pikseli uvelike pridonose divergenciji algoritma(Slika 22). Da bi algoritam bez maske sigurno konvergirao, okoliš oko znaka bi u stvarnosti trebao biti potpuno bijel što je nerealno za očekivati.

a) Znak za sporednu ulicu, postupak bez korištenja maske

b) Znak za sporednu ulicu, postupak sa korištenjem maske

c) Znak za oštri zavoj, postupak bez korištenja maske

d) Znak za oštri zavoj, postupak sa korištenjem maske

Slika 22 Rezultati algoritma bez maske i sa maskom

6 Zaključak

Ovaj rad opisuje specifičan lokalizator prometnih znakova koji precizno lokalizira isključivo prometne znakove trokutaskog oblika. Na njegov ulaz šalju se rezultati algoritma detekcije prometnih znakova koji detektira općenitiji skup prometnih znakova sa ne tako velikom preciznošću. Detekcijom trokutastog prometnog znaka, nad njegovim crvenim rubom kreiraju se tri histograma koji opisuju komponente nijansi (*hue*), zasićenja (*saturation*) i osvjetljenja (*value*/*lightness*). Informacije o koncentraciji crvene boje i njenom zasićenju mogu biti izrazito korisne pri detekciji kuteva. Za neku točku može se sa velikom vjerojatnošću pretpostaviti da se nalazi u kutu trokuta znaka ako u jednom dijelu svoje okoline sadrži isključivo crvenu boju, a u drugom dijelu ne sadrži. Postoje više varijanti algoritma za precizno lociranje znakova. Osim na Lucas-Kanade algoritmu, precizno lociranje znakova može se temeljiti i na kompozicijskom, inverznom kompozicijskom i inverznom aditivnom algoritmu. Svaka od tih četiriju varijanti ima svoje podvarijante. Ako se analizira jednostavniji skup slika iz postupka se može izbaciti binarizacija slike. U slučaju da je baza predložaka prevelika, može se koristiti univerzalni predložak za sve tipove znakova što uzrokuje slabiju detekciju. Programska implementacija algoritma poravnavanja slike sa predloškom provodi se u četiri faze: parsiranje ulaznog niza, pripremanje slike za algoritam gradijentne optimizacije, provedba algoritma gradijentne optimizacije, izrada histograma. U prvoj fazi se u pohranjuju podaci o položaju i tipu detektiranih znakova. Nakon toga se u drugoj fazi iz cijele slike uzima samo relevantno područje gdje se nalazi detektirani znak. Slika relevantnog područja se zasivljuje i takva se spremi u sliku za obradu. Treća faza uspoređuje predložak sa slikom za obradu i kao rezultat vraća matricu transformacije. Ovdje je važno napomenuti da se za treću fazu uzima onaj predložak koji po tipu odgovara znaku koji se sa njim uspoređuje. Matricom transformacije se transformira predložak i dobije se točna lokacija detektiranog znaka. U zadnjoj fazi se pomoću histograma kvantificira varijacija boje unutar crvenog trokuta znaka.

Program poravnavanja slike s predloškom u 80% slučajeva daje zadovoljavajuće rezultate. Neispravni rezultati najčešće su posljedica perspektivne deformacije, premalog/prevelikog osvjetljenja i objekata koji zaklanjaju znak. Neispravna konvergencija također može biti uzrokovana problemom "dvosmislenog" objekta koji je opisan u poglavlju 5.2. Korištenjem specifičnog predloška i binarizacije, postotak ispravnih rezultata je oko 90%.

Precizno lociranje znakova važan je dio sustava za implementaciju inteligentnih sustava u prometu. Ono se može koristiti za kvantifikaciju varijacija boje unutar znaka, provjeru rada detektora znakova, pomoć pri raspoznavanju uzoraka ili za verifikaciju ispravnosti prometnih znakova.

7 Sažetak

Precizno lociranje znakova temelji se na algoritmu gradijentne optimizacije koji modificira predložak kako bi bio dovoljno sličan detektiranom znaku. Znakovi koji se uspoređuju sa predloškom detektirani su nekom drugom metodom. Tip predloška mora odgovarati tipu detektiranog znaka, a sam predložak se dohvaća iz baze predložaka. Prije obrade ulazna slika se zasivljuje. Algoritam gradijentne optimizacije uspoređuje predložak i detektirani znak i ako oni nisu dovoljno slični mijenja matricu transformacije $W(x; p)$. Mijenjanje matrice transformacije i usporedba predloška sa znakom iterativno se ponavljaju do trenutka kada su predložak i znak dovoljno slični. Dijelovi predloška koji nisu bitni za usporedbu zanemaruju se pomoću maske. Rezultat optimizacije je matrica tranformacije. Za precizno lociranje znaka potrebno je postaviti predložak na sredinu znaka i pomnožiti predložak matricom transformacije. Na temelju precizno lociranog znaka izrade se tri histograma koji predstavljaju nijanse (*hue*), zasićenje (*saturation*) i osvjetljenje (*value/lightness*) slike crvenog ruba znaka.

8 Summary

Precise sign detection is based on gradient optimization algorithm which modifies the template in order to be as similar as the detected sign. Signs which are being compared are detected with some other method. Template type must be equal as the type of the detected sign and the template is taken from the set of templates. Before processing the entry picture must be concerted to grayscale. Gradient optimization algorithm compares the template and the detected sign and if they are not similar enough it changes the transformation matrix $W(x; p)$. Transformation matrix is being modified and the template and the detected sign are being compared until they are similar enough. Parts of the template which are not relevant for the comparison are ignored with a mask. The result of the optimization is the transformation matrix. In order to locate the sign precisely the template should be placed in the middle of the sign and multiplied with the transformation matrix. Based upon the precisely located sign, three histograms are made. They represent hue, saturation and value of the sign's red edge image.

Ključne riječi: detekcija prometnih znakova, algoritam gradijentne optimizacije, HSV histogram, matrica transformacije, predložak, maska.

9 Literatura

[1] Artificial intelligence, *Wikipedia*, http://en.wikipedia.org/wiki/Artificial_intelligence, 15.04.2009

[2] Major areas of AI, *Pick Brains*, http://www.pickbrains.com/articles/major-areas, 16.04.2009.

[3] Computer Vision, *Wikipedia*, http://en.wikipedia.org/wiki/Computer_vision, 16.04.2009

[4] Baker S., Matthews I., The Quantity Approximated, theWarp Update Rule, and the Gradient Descent Approximation, 18.10.2008, *Lucas-Kanade 20 Years On: A Unifying Framework*, http://www.ri.cmu.edu/pub_files/pub3/baker_simon_2004_1/baker_simon_2004_1.pdf, 17.04.2009.

[5] S. Šegvić, Z. Kalafatić, Dinamička analiza 3D scena, *Geometrijski aspekti računalnog vida*, http://www.zemris.fer.hr/~ssegvic/pubs/das2.pdf, 14.06.2009.

[6] K. Brkić, A. Pinz, S. Šegvić, Traffic sign detection as a component of an automated traffic infrastructure inventory system, *Pronalaženje i prepoznavanje prometne signalizacije*, http://www.zemris.fer.hr/~ssegvic/pubs/oagm09.pdf, 14.06.2009

[7] S. Šegvić, K. Brkić, Z. Kalafatić, V. Stanisavljević, D. Budimir, I. Dadić, Towards automatic assessment and mapping of traf_c infrastructure by adding vision capabilities to a geoinformation inventory, *Pronalaženje i prepoznavanje prometne*, http://www.zemris.fer.hr/~ssegvic/pubs/mipro09.pdf, 14.06.2009.

[8] Š. Bašić, P.Š. Čepo, I. Dodolović, D. Dostal, S. Grbić,M. Gulić, I. Horvatin, S. Louč, I. Sučić, Korištenje programskog okruženja cvsh, 15.1.2008., *Cannyjev detektor rubova*, http://www.ri.cmu.edu/pub_files/pub3/baker_simon_2004_1/baker_simon_2004_1.pdf, 18.04.2009.

[9] boost_1_39_0.zip, *Boost C++ Libraries*, http://www.boost.org/doc/libs/1_39_0/more/getting_started/windows.html, 04.04.2009.

[10] Windows Media Format 11 SDK, Windows Media, http://msdn.microsoft.com/en-us/windowsmedia/bb190309.aspx, 04.04.2009.

[11] OpenCV_1.1pre1a.exe, OpenCV library, http://sourceforge.net/project/showfiles.php?group_id=22870&package_id=16937, 05.04.2009.

[12] Getting started on Windows, *Boost C++ libraries*, http://www.boost.org/doc/libs/1_39_0/more/getting_started/windows.html, 05.04.2009

[13] Convert RGB Image to Grayscale, *Fanning Consulting*, http://www.dfanning.com/ip_tips/color2gray.html, 12.06.2009

[14] Priprema glavnog teksta Završnog rada, dokumentacije Projekta, odnosno teksta seminara, *Detaljnije upute za predmete Seminar, Projekt, Završni rad*, http://www.zemris.fer.hr/~ssegvic/project/detalji.html, 13.06.2009.

10 Privitak

Uz dokumentaciju je priložen CD koji sadrži kod, ulazne slike, slike preloška i izvršnu datoteku programa za precizno lociranje znakova. Program se pokreće pozicioniranjem u direktorij \Precizno lociranje prometnih znakova_debug. Prije početka izvršavanja programa potrebno je odabrati broj slika nad kojima će se izvršiti kvantifikacija boje pomoću histograma i zatim način prikaza rada programa. Detaljan prikaz rada će za svaku sliku prikazivati njene meĎurazultate dok će običan prikaz na kraju rada programa iscrtati histogram.