SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 774

PRONALAŽENJE GRANICA OBJEKATA ULANČAVANJEM RUBNIH ELEMENATA

Siniša Louč

Zagreb, lipanj 2009.

Sadržaj:

1. Uvod

Jedna od najvećih potreba suvremenog čovjeka jest transport. Danas najvažniju ulogu u transportu ima osobni automobil. Tisuće inženjera i znanstvenika diljem svijeta svakodnevno rade na osmišljavanju i implementiranju raznih tehnoloških pomagala kako bi se omogućilo što ugodniju i sigurniju vožnju. I dok se većina bavi sustavima za efikasnije kočenje, osiguranje stabilnosti na kolniku i sl., jedna sasvim druga skupina istraživača krenula je u nešto drugačijem smjeru – autonomna vozila.

Autonomna vozila su automatizirani automobili kojima se ne mora ručno upravljati. Putnik jednostavno sjedne, izvadi sendvič ili prijenosno računalo, a automobil se sam pobrine za siguran put do odredišta. Neophodan uvjet za rad takvog sustava jest ispravna i brza obrada podataka koje prikupljaju senzori, posebice kamera. Na osnovu ulaznih slika potrebno je uspješno detektirati rub kolnika, prometne linije, razne zapreke na cesti i sl.

Ovaj rad se bavi detektiranjem trokutnih i pravokutnih prometnih znakova. Sustav koji prepoznaje prometne znakove ne mora nužno biti korišten u autonomnom vozilu. Takav sustav može se koristiti i kao pripomoć u vožnji budući da čovjek nerijetko ne primjećuje prometnu signalizaciju. U tom slučaju sustav nakon prepoznavanja određenog znaka može djelovati na automobil (npr. prilagoditi brzinu) ili naprosto obavijestiti vozača o određenom ograničenju, upozorenju i sl. U literaturi je moguće pronaći povelik broj raznih metoda za detektiranje prometnih znakova [7].

U ovom radu znakovi se pronalaze na sljedeći način.

Pretpostavka je da je na automobil pričvršćena kamera koja snima scenu ispred sebe. Svaka pribavljena slika prvo se obrađuje Cannyjevim algoritmom [1] koji detektira rubove i kao izlaz daje binarnu sliku u kojoj svaki piksel ili predstavlja rub (logička jedinica) ili ne (logička nula). Zatim se na binarnoj slici pronalaze povezani skupovi (lanci) rubnih elemenata [3,4,8] te se u njima pronalaze pravocrtni segmenti [4]. Pomoću tih segmenata se hipotetiziraju trokutni i pravokutni prometni znakovi. Hipoteze se optimiraju gradijentnim spustom [5] te se konačno evaluiraju i, u slučaju pozitivnog rezultata, iscrtavaju.

U poglavlju 2 opisan je algoritam kojim se detektiraju prometni znakovi. U poglavlju 3 opisana je programska izvedba algoritma. Eksperimentalni rezultati prikazani su u poglavlju 4. Konačno, u poglavlju 5 dan je završni komentar u kojem se ocjenjuje uspješnost algoritma te ističu njegove prednosti i nedostaci.

Pojedini koraci rada algoritma bit će prikazani na ispitnom primjeru (slika 2.1).

Slika 2.1 Ispitni primjer

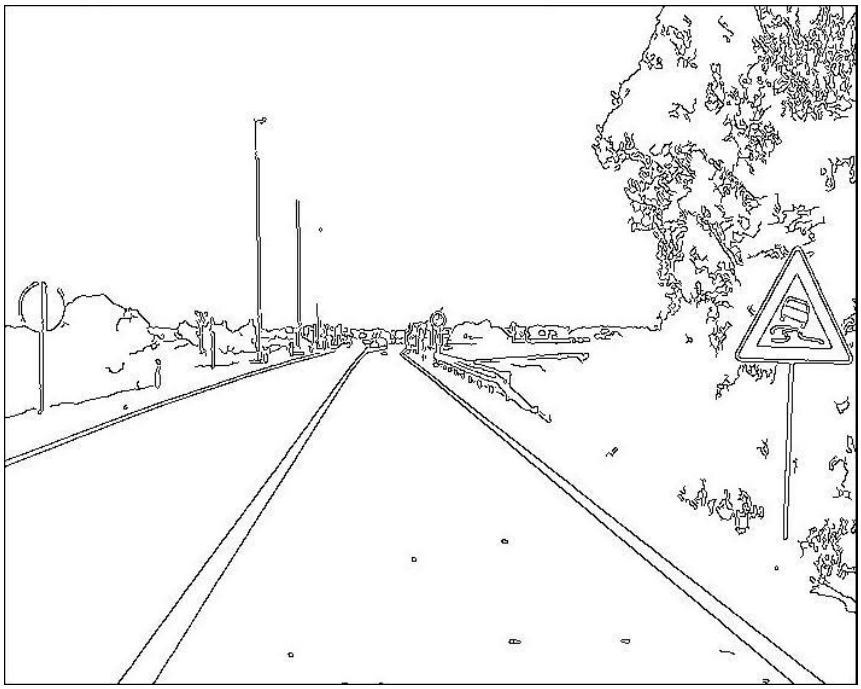
2. Korišteni algoritmi i metode

2.1. Cannyjev detektor rubova

Metoda detekcije rubova koju je razvio J. Canny jedna je od najšire korištenih i najpopularnijih jer predstavlja dobar omjer složenosti i efikasnosti. Naravno, efikasnost ovisi o domeni uporabe. Cannyjev algoritam je programski ostvaren u okviru projekta koji je prethodio ovom završnom radu. Detaljni opis algoritma može se pronaći u literaturi [1] i tehničkoj dokumentaciji spomenutog projekta [2].

Cannyev algoritam sastoji se od četiri faze.

U prvoj fazi se ulazna slika transformira u sivu sliku i obrađuje Gaussovim filtrom koji uklanja uvijek prisutni šum. Parametar Gaussovog filtra je standardna devijacija (σ). Rezultat je blago zamućena slika. Druga faza algoritma za svaki piksel određuje iznos (amplitudu) vektora gradijenta i njegov smjer pružanja. Računanje gradijenta za pojedini piksel obavlja se na temelju razlike u intenzitetu susjednih piksela. U trećoj fazi potrebno je stanjiti rubove na slici gradijenta na debljinu od jednog do najviše dva piksela. Četvrta faza pomoću dva praga, visokog i niskog (t_{high}, t_{low}), za svaki piksel određuje da li on pripada rubu ili ne. Time se postiže binarizacija (nema svjetlijih i tamnijih piksela, piksel ili jest rub ili nije). Algoritam detekcije prometnih znakova na ulaz prima binarnu sliku rubova koju daje Cannyjev algoritam i dalje ju obrađuje.



Slika 2.2 Izlaz Cannyjevog algoritma

Izlaz Cannyjevog algoritma za odabranu sliku prikazan je na slici 2.2. Parametri s kojima je slika obrađena su: σ=1.5, t_{high} =0.9, t_{low} =0.7. Pragovi su relativni, označavaju udio filtriranih rubnih elemenata kao što je opisano u [1,2]. Sva tri parametra se zadaju od strane korisnika.

2.2. Ulančavanje rubnih elemenata

Cannyjev algoritam kao izlaz daje binarnu sliku rubova. Ti rubovi nisu ni na koji način uređeni – jedino što je moguće napraviti jest ispitati da li je pojedini piksel rub ili ne. Da bi se omogućilo određene proračune koji su potrebni u nastavku rubove je potrebno povezati u neprekinute lance i spremiti ih u memoriju računala [3,8].

Postoje dva načina kako od postojećih rubova načiniti takve lance.

Jednoprolazno - slika se u petlji obilazi piksel po piksel pri čemu se svaki piksel provjerava samo jednom. Koristeći posebne strukture podataka pamti se svaki lanac te se za svaki promatrani piksel provjerava da li pripada nekom od lanaca.

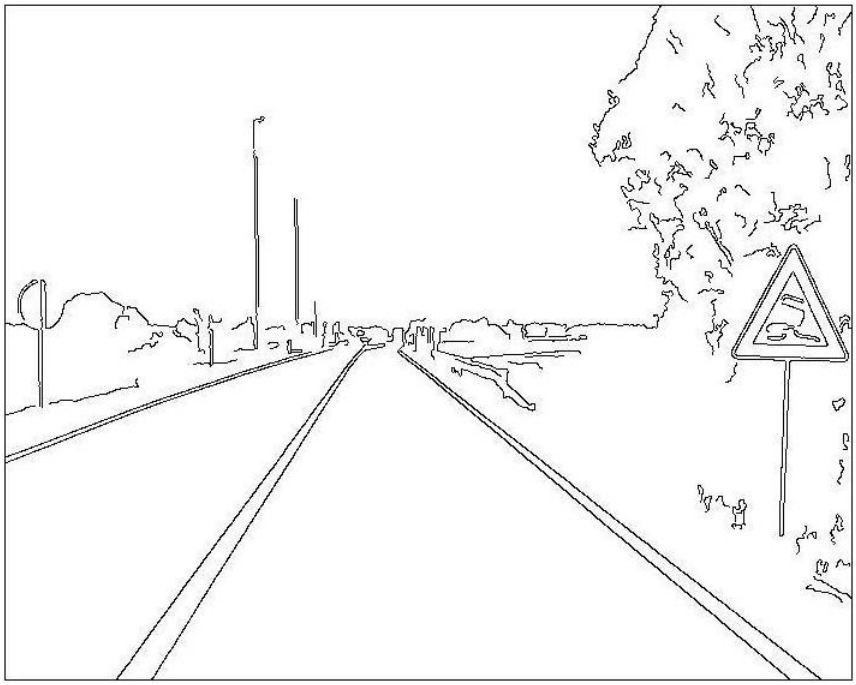
Rekurzivno - slika se također prolazi piksel po piksel, ali prilikom nailaska na piksel ruba funkcija se rekurzivno poziva za njegove susjede. Ako se za neki od susjeda ustanovi da je rub, funkcija se opet poziva za njegove susjede i tako sve dok za promatrani piksel postoje susjedni rubovi.

U ovom radu korištena je druga metoda.

U slučaju da je broj točaka konačnog lanca manji od neke unaprijed definirane granice, lanac se odbacuje. Naime, kratki lanci od nekoliko piksela ne samo da nisu korisni, nego i dodatno usporavaju rad algoritma (što se brzine tiče, što manje lanaca to bolje).

Ova faza je neka vrsta prekretnice u kojoj prestaje rad sa sirovom slikom, tj. njezinim pikselima, a počinje rad s lancima rubnih elemenata. Time se iz područja obrade slike prelazi u područje računalnog vida, odnosno simboličke obrade. Osnovne principe računalnog vida, od filozofskog razmatranja vida kao takvog do detaljnih algoritama korištenih u računarskoj znanosti, moguće je pronaći u literaturi [9].

Slika 2.3 prikazuje ispitnu sliku nakon ove faze. Valja uočiti da su uklonjeni lanci koji nisu bili dulji od zadanog praga koji iznosi 20 točaka.



Slika 2.3 Lanci piksela

Popis parametara programa u ovoj fazi: minimalna duljina lanca (dc_{min}) (20.0).

2.3. Pronalaženje pravocrtnih segmenata

U prethodnim koracima pronađeni su lanci rubnih elemenata. Idući korak je pronalaženje pravocrtnih segmenata u pronađenim lancima. Time se eliminiraju krivulje koje se ne koriste pri detekciji trokutnih i pravokutnih znakova. Pronalazak pravocrtnih segmenata ostvaren je metodom koju je opisao P. Kovesi u svojim radovima vezanim uz računalni vid [4]. Ideja je sljedeća.

Povuče se imaginarni pravac *p* kroz početnu i završnu točku lanca. Zatim se za svaku točku lanca računa njezina udaljenost od pravca *p* pomoću jednadžbe:

$$d = \left| \frac{A \cdot x + B \cdot y + C}{D} \right|$$

gdje se koeficijenti A,B,C i D računaju kao:

$$A = y_1 - y_2$$

$$B = x_2 - x_1$$

$$C = y_2 \cdot x_1 - y_1 \cdot x_2$$

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

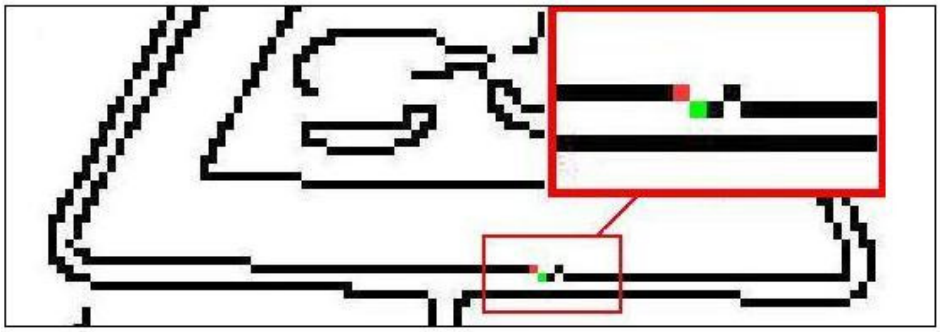
x_1, x_2, y_1 i y_2 su koordinate početne i završne točke lanca.

Ako dobivena udaljenost premašuje dozvoljenu granicu, lanac se dijeli na mjestu točke s najvećom udaljenosti (najvećom devijacijom) od pravca *p*. Postupak se zatim rekurzivno obavlja za obje polovice lanca. Primjerice, neka lanac ima *n* točaka. Maksimalna devijacija od pravca *p* jest u točki *k*. Neka omeđenost lanca označava par indeksa početne i završne točke lanca. Postupak se tada poziva za lanac omeđen točkama (1, k) te za lanac omeđen točkama (k+1, n). Kad u jednom trenutku maksimalna devijacija od pravca bude manja od zadanog praga primljenog kao parametar, lanac se sprema u memoriju.

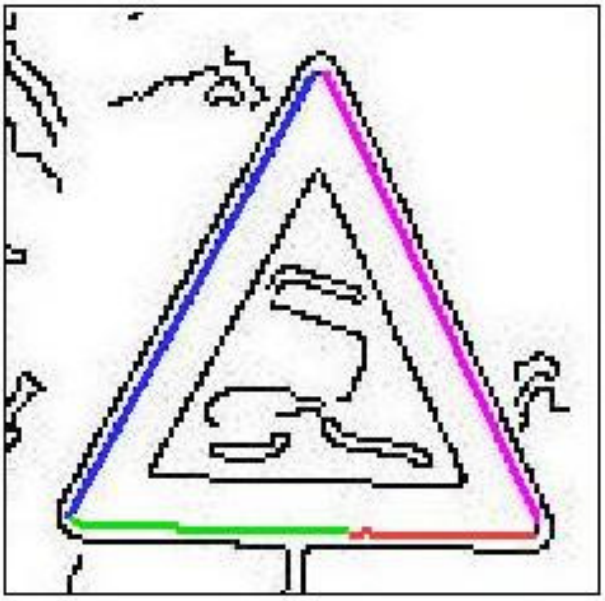
Postoji još jedan parametar – minimalna duljina pravocrtnog segmenta. Ako pravocrtni segment nije dovoljno velik odbacuje se kao što je bilo i u slučaju lanaca. Minimalna duljina segmenta također se prima kao parametar programa.

U ovom koraku se pojavila poteškoća (slika 2.4). Problem je u tome što se prilikom pronalaženja lanaca po pikselima iterira s lijeva na desno i odozdo

prema gore. Tako će na primjeru sa slike početak lanca biti detektiran u pikselu označenom zelenom bojom, a završetak u pikselu označenom crvenom bojom. Nakon opisanog postupka pronalaženja pravocrtnih segmenata ovo bi rezultiralo sa četiri segmenta (slika 2.5).



Slika 2.4 Problem kosog segmenta



Slika 2.5 Posljedica problema kosog segmenta

Ovakvo dijeljenje pravocrtnog segmenta ima negativne posljedice po detektiranje znakova jer time utječe na stvaranje hipoteza (poglavlje 2.5). Rješenje je jednostavno - za svaki piksel koji je zadnji u nekom horizontalnom

segmentu valja provjeriti da li se ispod ili dolje desno od njega nalazi piksel koji je početak nekog drugog horizontalnog segmenta. Ako jest, valja ih spojiti u jedan segment.

Opisani postupak primjenjuje se samo za horizontalne segmente. Naime, da bi neki piksel bio prepoznat kao početak segmenta on mora imati najmanju y koordinatu od svih piksela u segmentu i najmanju x koordinatu od svih piksela tog segmenta koji su u horizontalnoj ravnini s njim (na slici 2.5 zeleni piksel). Razlog tome je činjenica da se pikseli provjeravaju odozdo prema gore i s lijeva na desno, redak po redak, pa je jedina situacija u kojoj takav piksel nije ujedno i početak segmenta ona upravo opisana.

Popis parametara programa u ovoj fazi: minimalna duljina segmenta (ds_{min}) (zadaje se od strane korisnika), maksimalna dozvoljena devijacija od pravca (δ_{max}) (zadaje se od strane korisnika).

2.4. Grupiranje pravocrtnih segmenata

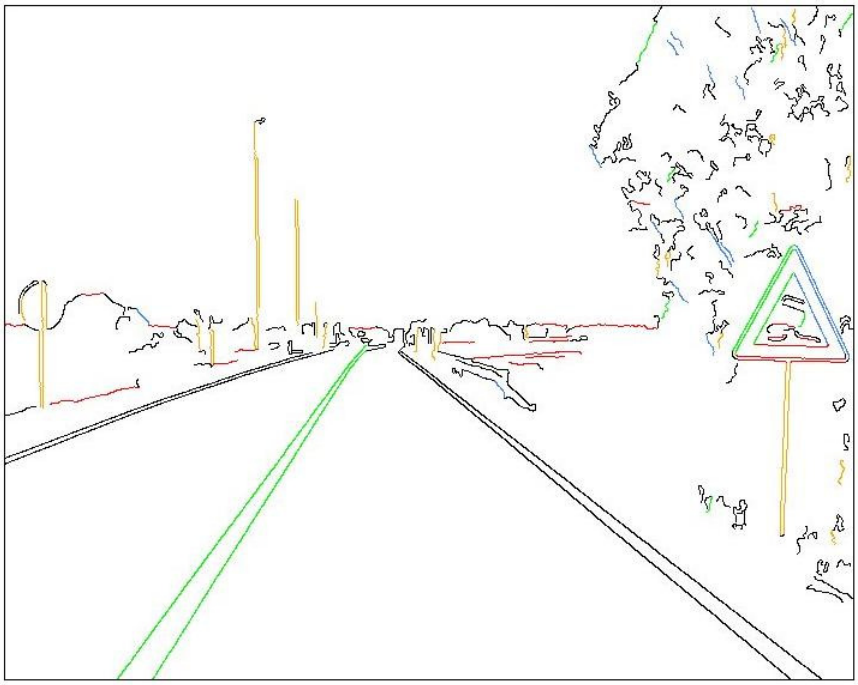
U ovoj fazi se općenitost algoritma naglo smanjuje. Segmenti se grupiraju onako kako je potrebno za algoritam detekcije trokutnih i pravokutnih prometnih znakova. Za neku drugu primjenu grupiranje se može ostvariti na drugačiji način.

Pravocrtni segmenti grupiraju se po kutovima. Grupiranje se obavlja neposredno nakon što se ustanovi da je maksimalna devijacija od pravca manja od zadane granice i da se segment prihvaća. Budući da se znaju koordinate početne i završne točke, pomoću funkcije arkus tangens lako se dođe do kuta. Zatim je još potrebno smjestiti dobiveni kut u jednu od četiri kategorije: 0º, 60º, 90º, 120º.

Dozvoljeno je odstupanje od 12º. Ako kut segmenta ne spada ni u jednu od kategorija segment se odbacuje.

Dvanaest stupnjeva predstavlja prilično veliko odstupanje, možda naizgled preveliko, ali jedina posljedica toga jest nastajanje većeg broj hipoteza. Vrijeme trajanja obrade će se zanemarivo povećati, a nema ni opasnosti od lažno pozitivnih rezultata jer se hipoteze provjeravaju. S druge strane, postoji vjerojatnost da će negdje upravo veće odstupanje omogućiti detektiranje nekog segmenta koji inače ne bi bio detektiran.

Slika 2.6 prikazuje pravocrtne segmente obojane ovisno o kutu.



Slika 2.6 Pravocrtni segmenti obojani ovisno o kutu

Popis parametara programa u ovoj fazi: maksimalno dozvoljeno odstupanje od kuta (φ_{max}) (12º).

2.5. Detekcija prometnih znakova na temelju hipoteza

2.5.1. Pronalaženje potencijalnih hipoteza

Dobiven je niz pravocrtnih segmenata koji se nalaze pod kutovima 0º, 60º, 90º i 120º. Potrebno je na osnovu tih segmenata pronaći trokutne i pravokutne oblike, ako postoje u slici.

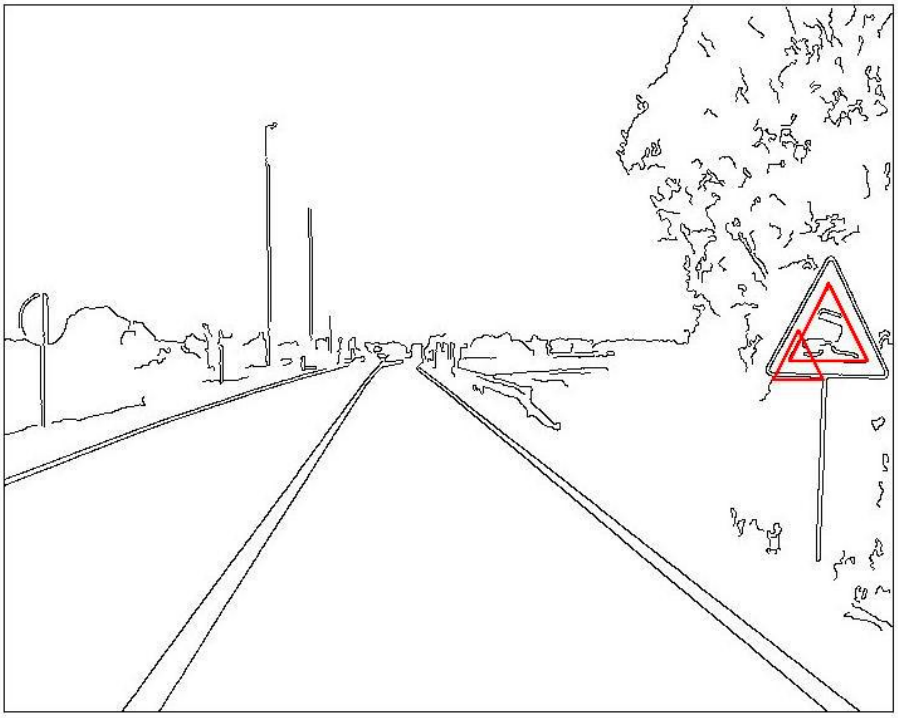
Princip rada temelji se na hipotezama koje postavljamo na temelju para segmenata, horizontalnog i kosog (60º) za trokutne odnosno horizontalnog i vertikalnog za pravokutne znakove. Kod stvaranja trokutnih hipoteza za svaki horizontalni segment se provjerava da li je udaljenost lijeve krajnje točke pronađenog segmenta od lijeve krajnje točke najbližeg kosog (60º) segmenta manja od zadanog parametra. Ako jest, stvara se trokutna hipoteza čija je duljina stranice jednaka duljini pronađenog horizontalnog segmenta. Za pravokutne znakove vrijedi ista analogija, uz vertikalni umjesto kosog segmenta. Alternativa detekciji znakova na temelju hipoteza jest detekcija znakova na temelju zatvorenih lanaca piksela, no takav pristup ima jedan veliki nedostatak: neotpornost na i najmanje prekide u lancu. Do prekida može doći uslijed nesavršenosti detektora rubova, a mogu ga izazvati i razne smetnje kao što su sjena, odbljesci svjetlosti ili djelomično zaklanjanje znaka drugim objektom.

Prije nego se stvori hipoteza provjeravaju se kutovi gradijenta (dostupni iz 2. faze Cannyjevog algoritma [2]) u svim pikselima horizontalnog segmenta. Ako segment sadrži manje od 75% piksela čiji je kut gradijenta približno 90º s odstupanjem od 30º, hipoteza se ne stvara. Time se sprječava detekcija vanjskog ruba znaka – cilj algoritma je detektiranje unutarnjeg ruba koji omeđuje sliku na prometnom znaku. Obrub znaka je uvijek crvene boje, a pozadina slike bijele pa zato pikseli donje stranice trokutnog obruba koji omeđuje sliku na

znaku imaju kut gradijenta približno 90º, dok pikseli donje stranice vanjskog ruba znaka imaju kut gradijenta približno 270º.

Svakako treba napomenuti da stvarni kut kod trokutne hipoteze nije točno 60º. Kut između katete i hipotenuze nešto je veći od 60º, a kut među katetama nešto manji. Hipoteze su jednakokračnog oblika jer su i sami znakovi na slikama jednakokračnog oblika zbog fotografiranja pod određenim kutom (znak se ne nalazi točno ispred objektiva već desno od njega). Radi jednostavnosti u radu se govori o kutovima 60º i 120º.

Na slici 2.7 mogu se vidjeti trokutne hipoteze na ispitnom primjeru.



Slika 2.7 Trokutne hipoteze na ispitnom primjeru

Pravokutne hipoteze imaju duljine stranica jednake duljinama dvaju segmenata na temelju kojih su stvorene. Trokutne hipoteze imaju duljinu donje stranice jednaku duljini pronađenog horizontalnog segmenta, a kose stranice se rade prema njoj. Ako se prilikom stvaranja pravokutne hipoteze ustanovi da je duljina jedne od stranica manja od praga (50 točaka za pravokutne, 25 za trokutne), hipoteza se odbacuje.

Popis parametara programa u ovoj fazi: minimalna udaljenost para segmenata na temelju kojih se radi hipoteza ($dseg_{min}$) (10.0), postotak piksela gradijenta na horizontalnom segmentu čiji je kut 90º (t_{grad90}) (0.7), odstupanje od kuta gradijenta na horizontalnoj stranici trokutnih hipoteza (δ_{kuth}) (30º), minimalna duljina stranice pravokutne hipoteze (str_{Tmin}) (50.0), minimalna duljina stranice trokutne hipoteze (str_{Pmin}) (25.0).

2.5.2. Optimizacija stvorenih hipoteza

Stvorene hipoteze moguće je optimirati kako bi što preciznije aproksimirale položaj prometnog znaka. U tu svrhu korištena je metoda gradijentnog spusta [5]. Algoritam gradijentnog spusta za skalarnu funkciju vektorske varijable F(x) može se opisati kako slijedi.

Neka je $F(x)$ derivabilna funkcija definirana u okolini točke a. Tada funkcija $F(x)$ opada najbrže krećući se od točke a u smjeru negativnog gradijenta $-\nabla F(x)$. Krenuvši od početne pretpostavke za rješenje x_0, razmatra se niz x_0, x_1, x_2... takav da vrijedi $x_{n+1} = x_n - \gamma_n \nabla F(x_n), n \geq 0$. Za prikladni parametar γ_n dobivamo $F(x_0) \geq F(x_1) \geq F(x_2)$, što znači da x_n konvergira prema minimumu funkcije F.

Pojednostavljeno rečeno, gradijentni spust za neku funkciju od *n* varijabli pronalazi takve vrijednosti varijabli za koje je ta funkcija minimalna. Zadana funkcija se naziva funkcijom cilja. Programska implementacija gradijentnog spusta korištena u ovom radu [6] minimizira sumu kvadrata *m* nelinearnih funkcija s *n* varijabli.

Domena funkcije su parametri hipoteze – koordinate početnog položaja hipoteze i duljine stranica. Pravokutne hipoteze stoga imaju četiri parametra (x,y,a,b), a trokutne tri (x,y,a). Kodomena funkcije cilja se može opisati kao mjera (ne)preklapanja pojedinih diskretnih elemenata hipoteze s gradijentom u slici. Što se hipoteza bolje poklapa s gradijentom, to je norma funkcije cilja $|F(x)|$ manja. Potrebno je, dakle, diskretizirati hipotezu i svaku diskretnu točku prikazati pomoću varijabli hipoteze te oblikovati funkciju cilja kao udaljenost diskretne točke hipoteze od najbliže točke visokog gradijenta. Točke gradijenta računaju se posebno za svaku iteraciju algoritma.

U prvoj iteraciji spusta diskretizirane točke se ne računaju, već se postavljaju na vrijednosti izračunate prije poziva funkcije spusta. Time se određuje početni položaj hipoteze.

Dane su formule za diskretizaciju pravokutne hipoteze. Za trokutne hipoteze vrijedi ista logika, samo se razlikuju formule. Hipoteze se diskretiziraju na 10 točaka po stranici.

x_p, y_p = početne koordinate

d_h, d_v = duljina horizontalne, odnosno vertikalne stranice hipoteze

$$\begin{aligned} x(i) &= x_p + \frac{i}{10} \cdot s_h \\ y(i) &= y_p \end{aligned}, \qquad i \in [1,10]$$

$$\begin{aligned} x(i) &= x_p + s_h \\ y(i) &= y_p + \frac{i-10}{10} \cdot s_v \end{aligned}, \qquad i \in [11,20]$$

$$\begin{aligned} x(i) &= x_p + s_h - \frac{i-20}{10} \cdot s_h \\ y(i) &= y_p + s_v \end{aligned}, \qquad i \in [21,30]$$

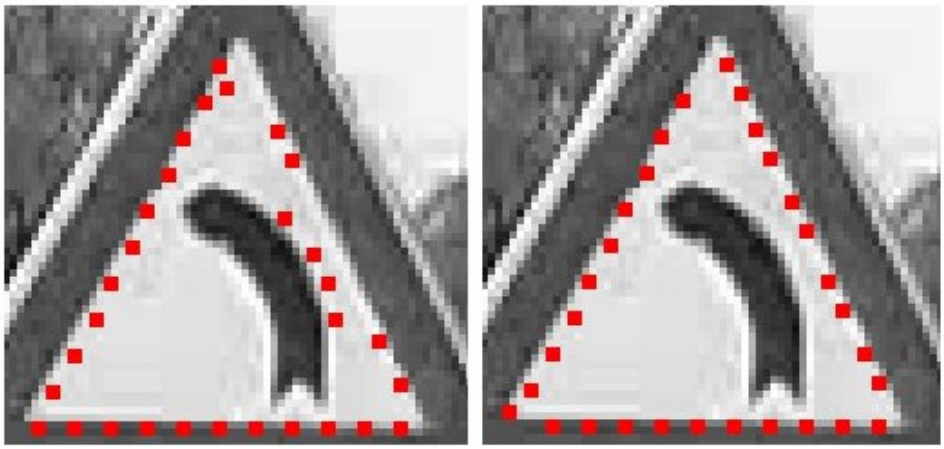
$$\begin{aligned} x(i) &= x_p \\ y(i) &= y_p + s_v - \frac{i-30}{10} \cdot s_v \end{aligned}, \qquad i \in [31,40]$$

Svaka točka je opisana pomoću četiri parametra hipoteze koji predstavljaju varijable u gradijentnom spustu. U svakoj iteraciji spusta potrebno je za svaku od diskretnih točaka hipoteze pronaći najbližu točku s odgovarajućim gradijentom. Postoje dva moguća kriterija za "odgovarajući gradijent":

a) maksimalni gradijent čija je udaljenost manja od neke granice;
b) najbliži gradijent čiji je intenzitet veći od neke granice.

U praksi se drugo rješenje pokazalo kao bolje jer je važnije da piksel gradijenta bude najbliži nego da bude maksimalan. Prag intenziteta u konkretnoj implementaciji iznosi 18.0 (eksperimentalno utvrđeno kao optimalno).

Prilikom traženja najbliže točke gradijenta potrebno je voditi računa i o kutu gradijenta te točke. Bez provjere kuta rezultat će biti kao na slici 2.8 lijevo.



Slika 2.8 Pikseli maksimalnog gradijenta sa i bez provjere kuta gradijenta

Slika prikazuje piksele gradijenta koji su najbliži diskretnim točkama hipoteze. Vidljivo je da četiri piksela leže na objektu koji predstavlja lijevi zavoj umjesto na okviru slike znaka. Zbog toga se za diskretne točke donje stranice hipoteze traže najbliži pikseli gradijenta čiji je kut gradijenta u okolini 90º, za točke desne stranice hipoteze traže se pikseli gradijenta u okolini -150º dok se za točke lijeve stranice traže pikseli s kutom u okolini -30º. Dozvoljeno odstupanje iznosi 15º. Kod pravokutnih znakova nema obruba na znaku pa nije moguće znati hoće li, primjerice, kut gradijenta piksela na donjoj stranici biti 90º ili -90º. Zbog toga se za pravokutne znakove dozvoljava i pozitivni i negativni smjer gradijenta.

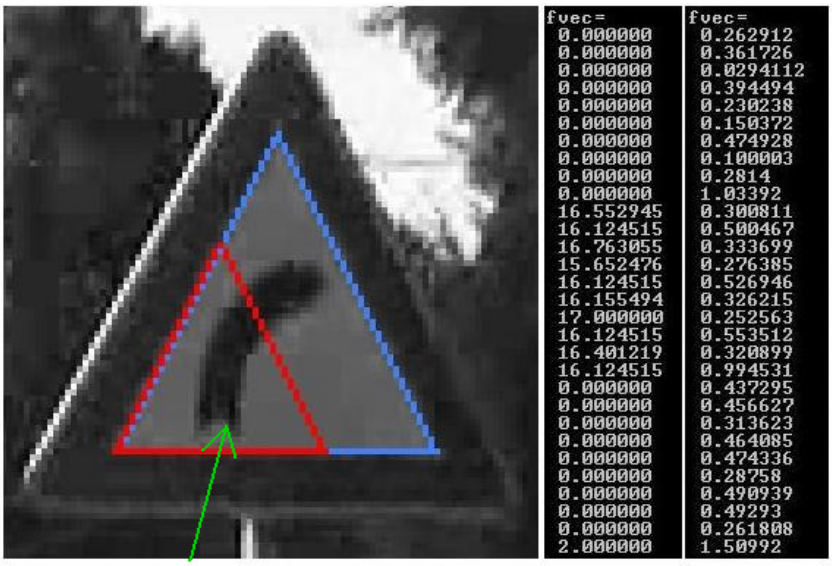
Ispravna detekcija najbližih piksela gradijenta od vrlo je velike važnosti za postupak optimizacije. U slučaju na slici 2.8 lijevo desna stranica poboljšane hipoteze bila bi pozicionirana nekoliko piksela lijevo od ruba jer bi je privlačila

četiri "zalutala" piksela s objekta koji predstavlja zavoj. Detekcija piksela kao na slici 2.8 desno daje ispravan rezultat.

Pikseli gradijenta traže se u okolini od 20 piksela u svim smjerovima za svaku točku hipoteze, osim kod točaka koje su na desnoj stranici trokutne hipoteze. Naime, nerijetko se događa da dođe do pukotine u lancu, a u praksi se pokazalo da su horizontalni segmenti posebno podložni ovakvim pukotinama. Teško je utvrditi točan razlog – ponekad je razlog nagib znaka, ponekad uklanjanje premalenih segmenata, a ponekad smetnje. Zbog toga trokutne hipoteze često znaju izgledati kao na slici 2.9 (crveno). Zato se prilikom traženja najbližeg piksela gradijenta kod diskretnih točaka desne stranice trokutne hipoteze provjera obavlja 2 piksela dolje i lijevo te 30 piksela gore i desno. Time je izbjegnuto da u primjeru na slici 2.10 najbliži pikseli gradijenta budu pronađeni na objektu koji predstavlja desni zavoj (konkretno, na desnoj polovici repa zavoja, na slici 2.9 označeno zelenom strelicom). Dovoljno je da samo jedan piksel gradijenta bude detektiran na spomenutom objektu i konačan rezultat će već biti kudikamo lošiji jer će taj piksel privlačiti hipotezu prema sebi. Ako je hipoteza toliko mala da se i objekt sa slike znaka nalazi iznad nje, tada je nemoguće spriječiti detekciju piksela gradijenta na tom objektu.

Provjera od samo dva piksela dolje i lijevo nije problem jer znak neće nikad biti ispod hipoteze – segmenti se mogu samo izgubiti/smanjiti, nikako stvoriti/produljiti pa hipoteza ne može biti veća od znaka.

Slika 2.9 prikazuje primjer rezultata optimizacije. Crvenom bojom označena je početna hipoteza, a plavom bojom optimirana hipoteza. S desne strane prikazane su redom elementi vektora funkcije cilja na početku optimizacije i elementi vektora funkcije cilja na kraju optimizacije. Dodatni primjeri prikazani su u poglavlju 4.2.



Slika 2.9 Primjer optimizacije i vrijednosti funkcija cilja u diskretiziranim točkama

Slika 2.10 prikazuje rezultat za sliku na kojoj je znak rotiran udesno. Vidljivo je odstupanje hipoteze od znaka. Moguće je ostvariti i rotiranje hipoteza, pri čemu bi kut rotacije bio dodatni parametar funkcije cilja u gradijentnom spustu. U ovom radu zbog uštede vremena i složenosti to nije ostvareno.

Slika 2.10 Neprecizna detekcija kao posljedica rotacije

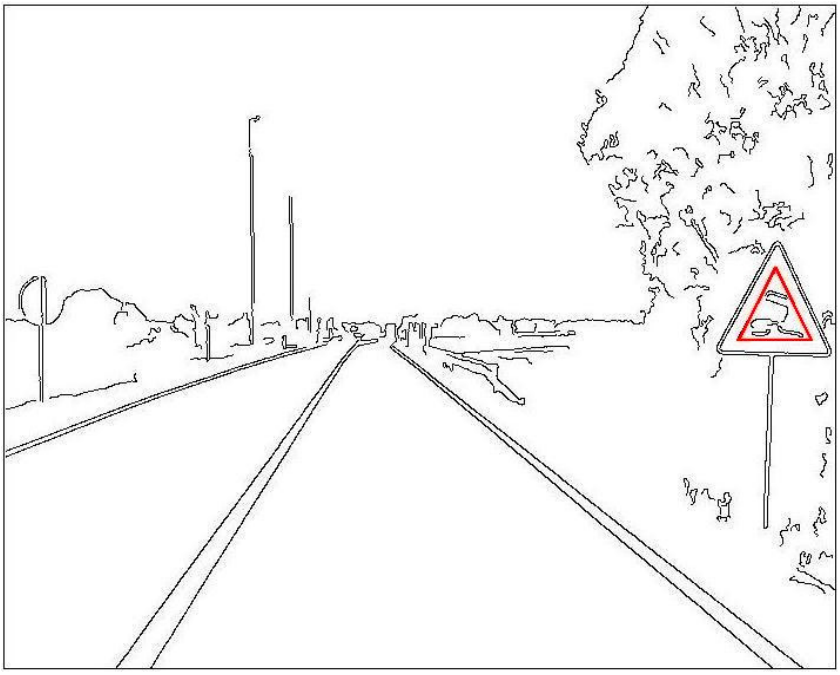
Popis parametara programa u ovoj fazi: prag intenziteta piksela gradijenta (t_{grad}) (18.0), odstupanje od kuta gradijenta (δ_{kutg}) (15º), broj točaka diskretizacije za trokutne i pravokutne znakove (nd_t, nd_p) (30,40), okolina točke desne stranice trokutne hipoteze u kojoj se traži piksel gradijenta (n_{ok1}) (-2,30), okolina točke ostalih stranica hipoteze u kojoj se traži piksel gradijenta (n_{ok2}) (-20,20).

2.5.3. Provjera valjanosti i iscrtavanje

Hipoteze su optimirane te je potrebno još provjeriti svaku hipotezu i odlučiti da li ona uistinu aproksimira prometni znak (ili bilo koji drugi trokutni ili pravokutni objekt – algoritam ne može odrediti radi li se uistinu o znaku). Provjera se obavlja vrlo jednostavno – provjerom reziduala funkcije cilja. Zahtijeva se da zbroj elemenata vektora funkcije cilja bude manji od nekog praga koji se zadaje kao ulazni parametar programa. Ukoliko hipoteza zadovoljava navedeni uvjet, radi se poboljšani uzorak pomoću vrijednosti varijabli početnih koordinata i duljina stranica koje su pronađene gradijentnim spustom. Naposljetku se iscrtavaju svi poboljšani uzorci.

Slika 2.11 prikazuje konačni rezultat za ispitni primjer. Popis ulaznih parametara programa dan je u tablici 2.1.

Popis parametara programa u ovoj fazi: maksimalni rezidual funkcije cilja za trokutne (t_{max}) (zadaje se od strane korisnika), odnosno pravokutne znakove (p_{max}) (zadaje se od strane korisnika).



Slika 2.11 Konačni rezultat

Oznaka	Značenje	Početna vrijednost
σ	parametar Gaussovog filtra	1.8
t_{high}	visoki prag Cannyjevog algoritma	0.5
t_{low}	niski prag Cannyjevog algoritma	0.3
ds_{min}	minimalna duljina pravocrtnog segmenta	15.0
δ_{max}	maksimalna devijacija od pravca	2.5
t_{max}	maksimalni rezidual funkcije cilja za trokutne hipoteze	40.0
p_{max}	maksimalni rezidual funkcije cilja za pravokutne hipoteze	80.0

Tablica 2.1. Popis ulaznih parametara

3. Programska implementacija

3.1. Struktura programske implementacije

U programskoj implementaciji algoritma korištena je ljuska `cvsh`[1] (computer vision shell). Ljuska je razvijena kako bi se olakšalo i pojednostavnilo eksperimentiranje s algoritmima računalnog vida u programskom jeziku C++. Dodatne informacije i detalji o korištenju ljuske dostupni su u tehničkoj dokumentaciji projekta koji je prethodio ovom radu (Cannyjev detektor rubova) [2].

Programski kôd je raspodijeljen u sedam komponenti:

alg_link.cpp je glavna komponenta koja izvršava algoritam detektiranja znakova. Komponenta, između ostalog, sadrži klasu `alg_link` koja posjeduje metodu `alg_link::process()`:

```
void alg_link::process(
  const img_vectorAbstract& src,
  const win_event_vectorAbstract&,
  int)
```

Povezivanje ljuske s korisničkim postupkom obavlja se preko virtualnog poziva funkcije `process()`. Ljuska poziva funkciju `process()` aktivnog algoritma svaki put kad korisnik pokrene obradu slike.

[1] Izvorni kôd ljuske dostupan je na adresi http://www.zemris.fer.hr/~ssegvic/src/cvsh_src_090310.tar.gz

Komponenta alg_link.cpp obavlja algoritam detekcije znakova tako što poziva pojedine funkcije smještene u datotekama ext_linkSegment.cpp, ext_linkLines.cpp i ext_signDetection.cpp. Funkcije koje poziva su:

```
void edgeSegmentation(  int margin,
                        const img_wrap& imgThreshold,
                        img_wrap& imgSegment);

void findStraightLines (  win_ann& annLines,
                          double lineSize, double maxDev);

void locateSigns (  win_ann& annSigns,
                    int margin,
                    const img_wrap& imgPhi,
                    const img_wrap& imgGrad,
                    double threshold);
```

Svaka funkcija izvršava jednu od tri faze algoritma:

1. ulančavanje rubnih elemenata
2. pronalaženje pravocrtnih segmenata
3. detekcija prometnih znakova na temelju hipoteza

Prije poziva navedenih funkcija, potrebno je pozvati i funkcije koje obavljaju pojedine faze Cannyjevog algoritma, što se izvršava na isti način kao i u samoj programskoj izvedbi Cannyjevog algoritma [2].

ext_linkSegment.cpp je komponenta u kojoj su smještene funkcije koje obavljaju prvu fazu algoritma – ulančavanje rubnih elemenata (poglavlje 3.2.1).

ext_linkLines.cpp sadrži funkcije kojima se pronalazi pravocrtne segmente u lancima iz prethodne faze (poglavlje 3.2.2).

ext_signDetection.cpp se sastoji od jedne funkcije koja obavlja detekciju prometnih znakova na temelju pravocrtnih segmenata pronađenih u prethodnoj

fazi (poglavlje 3.2.3) te još dvije funkcije koje algoritmom gradijentnog spusta pronalaze najbolje moguće vrijednosti ulaznih parametara (v. poglavlje 2.5.2).

Svaka od tri navedene datoteke ima pripadajuću datoteku s nastavkom **.hpp** u skladu s konvencijom o pravilnom strukturiranju programskog kôda.

Osim ovih sedam komponenti, koriste se i dvije vanjske komponente koje sadrže algoritam gradijentnog spusta: **lmdif.cpp** i **lmdif.hpp**.

3.2. Opis programske implementacije

3.2.1. Ulančavanje rubnih elemenata

Lanac povezanih piksela u memoriji je pohranjen sljedećom strukturom:

```
struct Edgel{
      int x;
      int y;
};

std::vector<
      std::vector<Edgel> >chains;
```

Struktura `Edgel` predstavlja jednu točku lanca koja ima svoju x i y koordinatu. Niz takvih točaka sprema se u `vector` (spremnik STL-a) koji tako predstavlja jedan lanac. Budući da na slici postoji više lanaca, oni se također spremaju u `vector`. Dakle, postoji jedan spremnik lanaca koji se sastoji od više spremnika od kojih svaki pojedini predstavlja jedan lanac i sastoji se od više točaka. Tako se, primjerice, x koordinati devete točke četvrtog lanca pristupa sa `chains.at(3).at(8).x` (indeksi počinju od 0).

Funkcije koje obavljaju ulančavanje rubnih elemenata su:

```
void findSegments( int margin,
                   int i,
                   int j,
                   img_wrap& imgSegment);

void edgeSegmentation(  int margin,
                        const img_wrap& imgThreshold,
                        img_wrap& imgSegment);
```

Funkcija `edgeSegmentation()` prima marginu slike (detalji u dokumentaciji Cannyjevog algoritma [1,2]), binarnu sliku rubova te još jednu, u tom trenutku praznu, sliku koja će po završetku sadržavati piksele povezanih lanaca. To je glavna funkcija koja iterira po pikselima slike i za svaki piksel koji predstavlja rub poziva rekurzivnu funkciju `findSegments()`.

Funkcija `findSegments()` prima marginu, sliku povezanih lanaca u koju će crtati te koordinate piksela za koji se poziva. Funkcija se dalje poziva rekurzivno za svaki susjedni piksel koji predstavlja rub. Svaki piksel za kojeg se funkcija pozove mora se označiti kako se ne bi jedan te isti piksel spremio u više lanaca. Ako više nema susjednih rubova za koje bi se rekurzivno pozvala, a duljina lanca je manja od zadane granice, funkcija postavlja zastavicu `erase` koja će na koncu uzrokovati uklanjanje pronađenog lanca iz memorije.

3.2.2. Pronalaženje pravocrtnih segmenata

Oblik u kojem je pravocrtni segment prikazan u memoriji ne razlikuje se od onog u kojem je prikazan lanac:

```
std::vector<
      std::vector<Edgel> >lines;
```

Ova faza se sastoji od tri funkcije:

```
MaxDev findMaxDev (int a, int first, int last);

void findLines (int a, int first, int last);

void findStraightLines (  win_ann& annLines,
                          double lineSize,
                          double maxDev);
```

Funkcija `findStraightLines()` je glavna funkcija koja iterira po točkama lanaca. Prima adresu anotacije u koju će se spremiti pikseli pronađenih segmenata. Anotacija se na kraju iscrtava preko neke od postojećih slika (konkretno za ovu fazu - preko slike lanaca).

Za svaki lanac se poziva funkcija `findLines()` koja provjerava da li je maksimalna devijacija od pravca manja od zadane granice. Ako nije, funkcija se rekurzivno poziva za dvije polovice lanca kao što je opisano u poglavlju 2.3. Maksimalna devijacija računa se pomoću funkcije `findMaxDev()`.

Funkcija `findMaxDev()` vraća objekt tipa `MaxDev`:

```
struct MaxDev{
      double value;
      int x;
      int y;
      int index;
};
```

koji sadrži podatke o maksimalnoj devijaciji – iznos devijacije, indeks točke u lancu u kojoj je devijacija maksimalna te njezine koordinate.

Kutovi se pamte u posebnom spremniku:

```
std::vector<int> angles;
```

pri čemu indeks kuta odgovara indeksu lanca. Tako npr. sedmi lanac ima kut spremljen u `angles.at(6)`.

3.2.3. Detekcija prometnih znakova na temelju hipoteza

Glavni dio posla se obavlja u funkciji:

```
void locateSigns (  win_ann& annSigns,
                    int margin,
                    const img_wrap& imgPhi,
                    const img_wrap& imgGrad,
                    double threshold);
```

koja na temelju horizontalnih segmenata stvara hipoteze i sprema ih u memoriju, a potom za svaku pojedinu hipotezu provjerava da li je valjana.

Hipoteze se spremaju u spremnike STL-a:

```
std::vector<
            std::vector<Edgel> >patterns;
```

Struktura `Turn`

```
struct Turn{
            int fst;
            int snd;
            int trd;
      };


std::vector<Turn> turns;
```

za svaku hipotezu sadrži indekse svih vrhova osim prvog. Bolje rečeno, to su indeksi onih točaka u kojima dolazi do odstupanja od pravocrtnog smjera kretanja ako se krećemo po uzorku počevši od početne točke. Tako će u

pravokutnom uzorku varijabla `fst` sadržavati indeks donjeg desnog vrha, varijabla `snd` indeks gornjeg desnog vrha, a varijabla `trd` indeks gornjeg lijevog vrha. Za trokutaste hipoteze varijabla `trd` je prazna. Indeksi se pohranjuju u `vector` na isti način kao i kutevi – npr. peti element vektora `turns` sadrži podatke vezane uz peti uzorak.

Funkcije

```
void fcnPrav ( void* context,
               int ndst,
               int nsrc,
               double xgradp[],
               double fvecp[],
               int* iflag);

void fcnTro (  void* context,
               int ndst,
               int nsrc,
               double xgradt[],
               double fvect[],
               int* iflag);
```

predstavljaju funkcije cilja za algoritam gradijentnog spusta. Sâm algoritam se poziva pomoću funkcije

```
bool lmdif_entry(
  int ndst,
  int nsrc,
  double* x,
  double* fvec,
  lmdif_fcn* fcn,
  void* context,
  bool logdata)
```

definirane u `lmdif.cpp`. Parametri ndst, nsrc, x, fvec i fcn predstavljaju redom: broj nelinearnih funkcija (ulaz algoritma), broj varijabli u funkciji cilja, pokazivač na polje varijabli, pokazivač na polje ulaznih funkcija, pokazivač na funkciju cilja (`fcnPrav` za pravokutne, `fcnTro` za trokutne znakove).

Postoje još dvije funkcije za stvaranje trokutnih, odnosno pravokutnih hipoteza:

```
void makeRecHyp(  std::vector<std::vector<Edgel> >&patterns,
                  int length, int length_90, Edgel *start,
                  std::vector<Turn>&turns);

void makeTriHyp(  std::vector<std::vector<Edgel> >&patterns,
                  int length, Edgel *start,
                  std::vector<Turn>&turns);
```

Funkcije primaju vektor u kojega će spremat uzorke, duljinu stranice (za trokutne) odnosno stranica (za pravokutne), varijablu tipa `Edgel` koja označava koordinate početne točke hipoteze, te pripadajući vektor `turns` opisan ranije.

4. Eksperimentalni rezultati

4.1. Organizacija rezultata

Kod testiranja algoritama koji služe za prepoznavanje objekata, eksperimentalni rezultati se dijele na tri skupine:

- Eksperimenti u kojima se algoritam dobro ponaša (pozitivni rezultati)
- Eksperimenti u kojima algoritam nalazi objekte koje nismo tražili (lažno pozitivni rezultati)
- Eksperimenti u kojima algoritam ne pronalazi tražene objekte (lažno negativni rezultati)

Testiranje je obavljeno na 96 ispitnih slika (100 znakova). Sve ispitne slike obrađene su sa sljedećim parametrima:
σ =1.8, t_{high} = 0.5, t_{low} = 0.3, ds_{min} = 15.0, δ_{max} = 2.5, t_{max} = 40.0, p_{max} = 80.0.

4.2. Ilustracija toka postupka optimizacije

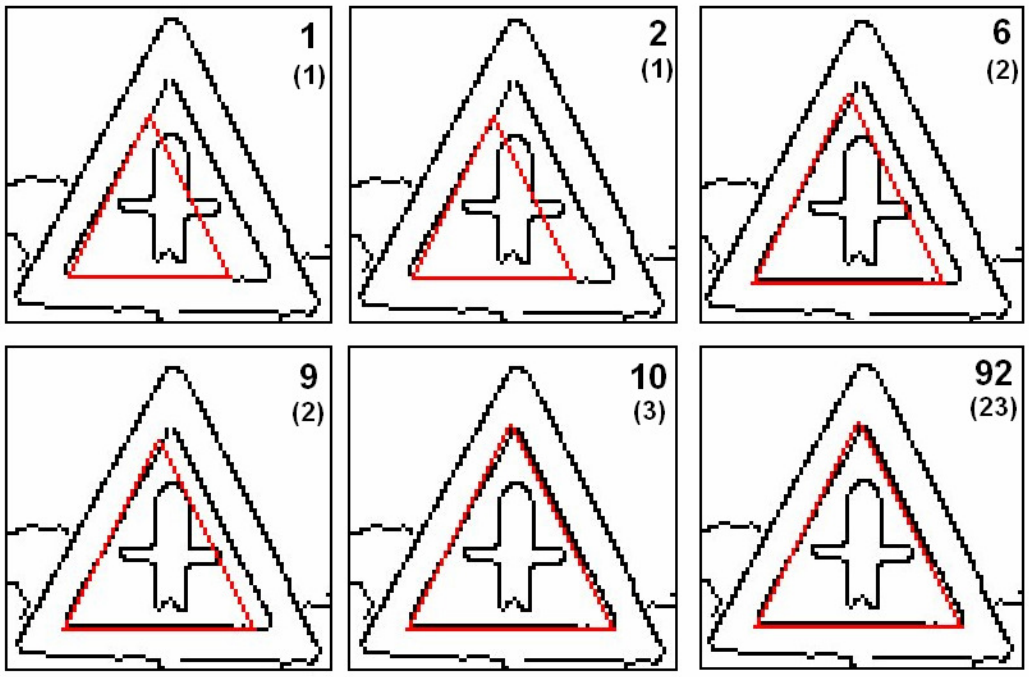
Postupak optimizacije gradijentnim spustom ilustriran je slikama 4.1. i 4.2. Slika 4.1 prikazuje nekoliko iteracija optimizacije na stvarnoj slici. Pojedina iteracija se sastoji od četiri koraka jer se u svakoj iteraciji evaluacija funkcije cilja obavlja četiri puta:

$$f(x, y, a)$$
$$f(x + \Delta x, y, a)$$

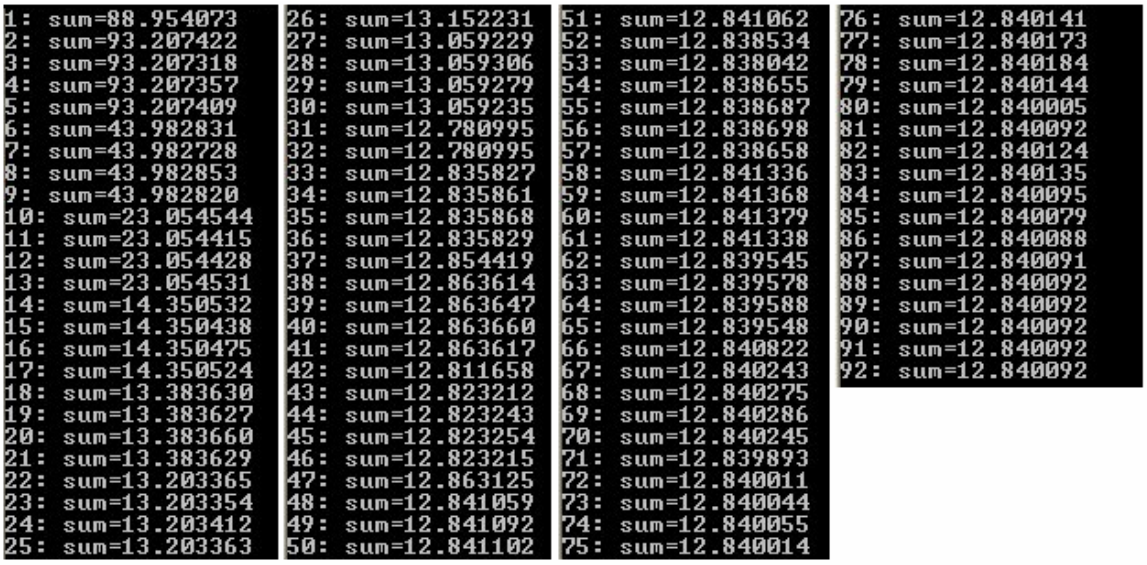
$$f(x, y + \Delta y, a)$$

$$f(x, y, a + \Delta a)$$

Pojedini koraci optimizacije naznačeni su u gornjem desnom uglu pojedine slike, izvan zagrada, a iteracije unutar zagrada. Valja primijetiti kako je već u 10. koraku (početak 3. iteracije) dobiven rezultat vizualno identičan onom iz 92. (zadnjeg) koraka. Međutim, reziduali funkcije cilja se razlikuju (slika 4.2). Rezultat je vizualno isti u oba slučaja jer se parametri hipoteze koje vraća funkcija cilja moraju zaokružiti na cjelobrojnu vrijednost kako bi se hipoteza mogla iscrtati.

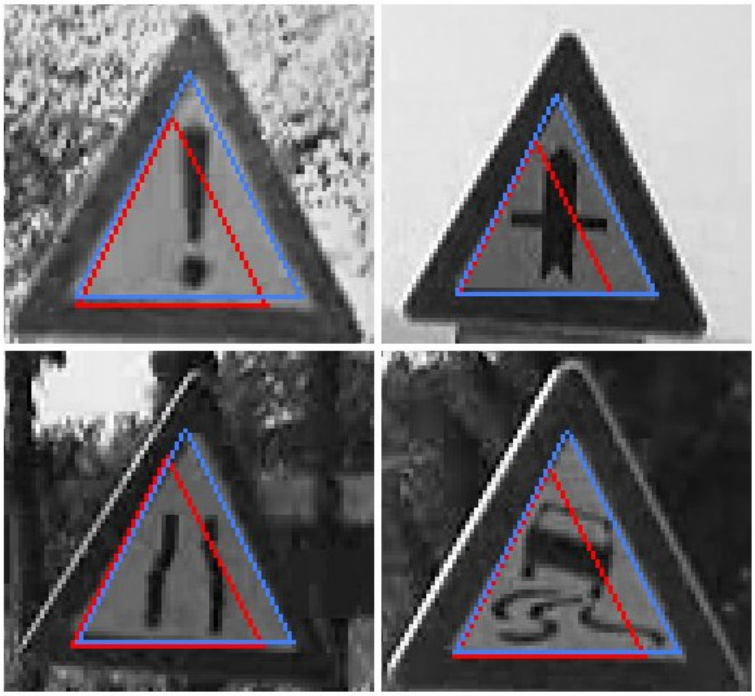


Slika 4.1 Ilustracija toka optimizacije na stvarnom primjeru



Slika 4.2 Reziduali funkcije cilja za primjer sa slike

Nekoliko dodatnih primjera prikazano je na slici 4.3. Crvenom bojom je označena početna hipoteza, a plavom optimirana.



Slika 4.3 Nekoliko primjera optimizacije

4.3. Pozitivni rezultati

Pozitivni rezultati su oni u kojima je prometni znak ispravno detektiran. Prikazana su četiri pozitivna rezultata (slike 4.4 - 4.7).

Slika 4.4

Slika 4.5

Slika 4.6

Slika 4.7

4.4. Lažno pozitivni rezultati

Na slici 4.8 prikazan je primjer lažno pozitivnog rezultata. Ovo je primjer situacije u kojoj ni provjera kuta prilikom traženja najbližeg gradijenta (primjer uz sliku 2.8) ni smjer pretrage (primjer uz sliku 2.9) nisu pomogli. Ovakav rezultat je u principu pozitivan, ali je odstupanje na gornjem znaku toliko veliko da se mora kvalificirati kao pogrešno prepoznavanje. Ovakav rezultat teško je evaluirati kao strogo lažno pozitivan, odnosno lažno negativan. U ovom radu ovakvi rezultati su kvalificirani kao lažno pozitivni budući da je došlo do detekcije (za razliku od lažno negativnih), mada ona nije uistinu *lažno* pozitivna nego samo neispravno pozicionirana.

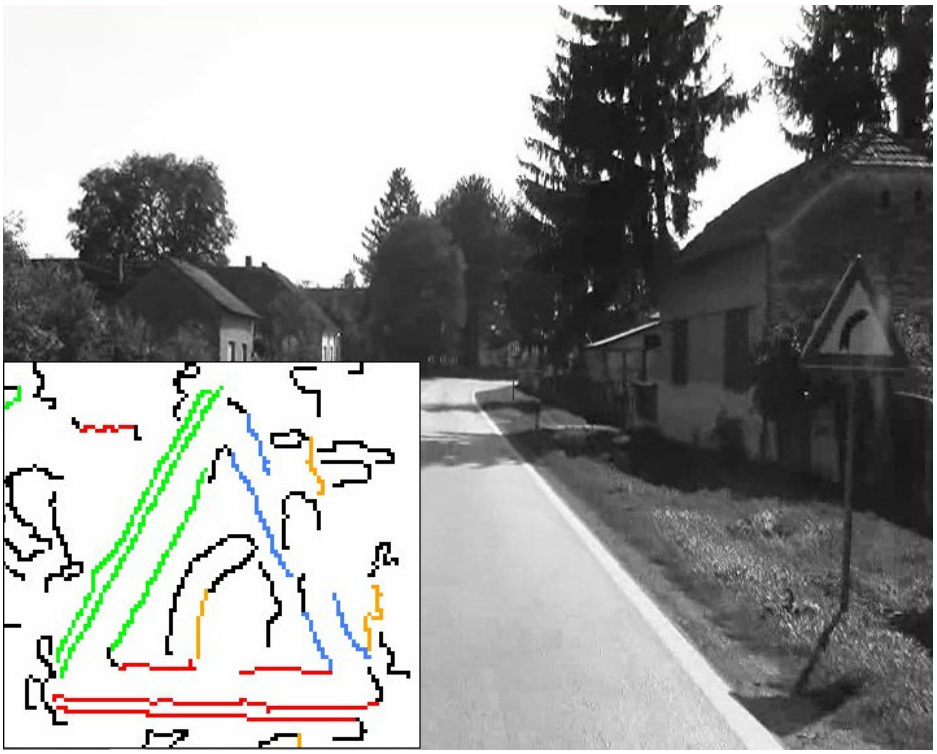
Slika 4.8

Slika 4.9

Na slici 4.9 detektiran je vanjski rub znaka, što nije cilj algoritma. Detekcija vanjskih rubova je u većini slučajeva uspješno spriječena provjerom kuta gradijenta piksela na donjoj stranici trokutne hipoteze (poglavlje 2.5.1), ali u ovom slučaju je donji rub znaka svjetliji, a obrub slike znaka je standardno tamno siv (crven u originalu) pa je zbog toga kut gradijenta 90º, a ne 270º.

4.5. Lažno negativni rezultati

Lažno negativni rezultati su oni u kojima znak ne biva detektiran. Postoje dva glavna uzroka lažno negativnih rezultata: loš rezultat detektora rubova i smetnje na slici (sjene, odbljesci, zaklanjanje drugim objektom).



Slika 4.10

Na slici 4.10 prikazan je lažno negativan rezultat kojem je uzrok loša detekcija rubova. U donjem lijevom uglu slike vidljiva je pukotina u donjem rubu slike znaka. Ovaj problem moguće je riješiti na nekoliko načina. Najefikasnije bi bilo postaviti hipotezu samo na temelju lijeve stranice slike znaka. Međutim, to ne rješava problem ako se pukotina pojavi upravo na lijevoj stranici. Druga mogućnost jest poboljšanje detekcije rubova, bilo izmjenom Cannyjevog algoritma ili samo odabirom drugih vrijednosti parametara. Pritom treba imati na umu da parametri koji daju dobre rezultate na jednoj slici daju loše rezultate na nekoj drugoj. Nemoguće je pronaći univerzalne parametre.

Na slici 4.11 znak nije detektiran jer smetnje u slici (obrasla biljka i njezina sjena) onemogućavaju ispravnu detekciju rubova u Cannyjevoj fazi algoritma. Za razliku od prošlog primjera, ovdje nema jednostavnog rješenja problema.

Slika 4.11

Znak koji je djelomično zaklonjen drugim većim objektom također vjerojatno neće biti uspješno detektiran (npr. drugim znakom). Mala zaklanjanja nisu nužno problem, pogotovo ako je zaklonjena stranica znaka koja ne sudjeluje u stvaranju hipoteze (kod trokutnih desna, kod pravokutnih desna i gornja). Ipak, to ovisi od slike do slike.

Postoji još nekoliko uzroka lažno negativnih rezultata, npr. eliminacija hipoteze zbog nezadovoljenosti uvjeta o kutu gradijenta u pikselima donje stranice znaka (ako znak na slici ima tamni element koji gotovo dodiruje donji rub slike) ili npr. zaglavljivanje algoritma optimizacije u lokalnom minimumu (osnovni nedostatak gradijentnog spusta [5]).

4.6. Statistika uspješnosti algoritma

Testiranje je obavljeno na 96 slika (ukupno 100 prometnih znakova), pri čemu su slike odabrane tako da nisu zamućene i da su znakovi dovoljno veliki. Treba naglasiti da se testiraju detekcije znakova, a ne slike, tako da se npr. rezultat kao na slici 4.9. klasificira kao jedan pozitivan i jedan lažno pozitivan.

Pozitivni rezultati	90
Lažno pozitivni rezultati	8
Lažno negativni rezultati	8
Preciznost	91.8%
Odziv	91.8%

Tablica 4.1. Statistika rezultata testiranja

5. Zaključak

Razvijeni algoritam za detekciju prometnih znakova pokazao se sukladan očekivanjima. Činjenica da hipotetiziranje ovisi o Cannyjevom detektoru rubova čini ga manje robusnim zbog mogućih pukotina u rubovima. Osim toga, na rezultat negativno utječu i smetnje na slici i rotacija znaka koja smanjuje preciznost detekcije. Ipak, valja naglasiti da u odsustvu smetnji i rotacije odziv i preciznost algoritma prelaze 90%.

Naravno, uvijek ima mjesta za poboljšanja. Tako bi, primjerice, bilo moguće ostvariti naprednije hipotetiziranje, npr. dodati i hipotetiziranje na temelju pojedinačnih stranica, a ne samo parova. Također, moguće je poboljšati optimizaciju uvođenjem kuta rotacije kao dodatnog parametra. Potencijalan parametar je i koeficijent sažimanja donje stranice trokutne hipoteze, čime bi se omogućilo stvaranje "uskih" hipoteza (korisno ako se znak ne nalazi ispred promatrača nego izrazito koso u odnosu na njega).

Ostvareni algoritam bio bi primjenjiv u sustavima gdje brzina i robusnost nisu od presudne važnosti i dozvoljava se pokoja greška u detekciji. Također, postupak predstavlja dobru osnovu za razvijanje naprednijih postupaka detekcije oblika. Nije idealan za primjene u kojima samo jedna od tisuću pogrešnih detekcija može imati vrlo negativne posljedice.

6. Literatura

[1] Canny, J. *A Computational Approach To Edge Detection'*, 1986., PAMI, 8. svezak, 6. izdanje

[2] Bašić, Š., Čepo, P. Š., Dodolović, I., Dostal, D., Grbić, S., Gulić, M., Horvatin, I., Louč, S., Sučić, I. *Cannyjev detektor rubova*, tehnička dokumentacija, Fakultet elektrotehnike i računarstva, 2008.

[3] Morse, B.S., Segmentation (Edge Based, cont'd), 26.02.2000., *CVonline: Edge linking,* http://homepages.inf.ed.ac.uk/cgi/rbf/CVONLINE/entries.pl?TAG408*,* 22.03.2009.

[4] Kovesi, P.D., Edge linking and line segment fitting, godina 2000, *MATLAB and Octave Functions for Computer Vision and Image Processing,* http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/, 22.03.2009.

[5] Turk S., Budin L. *Analiza i projektiranje računalom*, 2. izdanje, Zagreb, Školska knjiga, 1989.

[6] Netlib Repository at UTK and ORNL, http://www.netlib.org/minpack/lmdif.f, 03.06.2009.

[7] Liu Yangxing, *Reliable classification system for diverse traffic sign patterns*, 2007, doktorska disertacija

[8] Ballard, D. H., Brown C. M. *Computer Vision*, 2. izdanje, Prentice-Hall, Inc. 1982. http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/toc.htm, 21.03.2009.

[9] Marr, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, 1. izdanje, W. H. Freeman and Company 1996. http://homepages.inf.ed.ac.uk/rbf/BOOKS/MARR/toc.htm, 21.03.2009.

Pronalaženje granica objekata ulančavanjem rubnih elemenata

Sažetak

Opisuje se postupak detekcije trokutnih i pravokutnih prometnih znakova na slici metodom ulančavanja rubnih elemenata slike. Do rubnih elemenata se dolazi primjenom Cannyjevog algoritma detekcije rubova. Rubni elementi se ulančavaju te se u njima pronalaze pravocrtni segmenti. Položaji prometnih znakova se hipotetiziraju na temelju dobivenih pravocrtnih segmenata. Hipoteze se optimiraju gradijentnim spustom pri čemu su parametri funkcije cilja položaj i veličina znaka. Opisana je razvijena programska implementacija u programskom jeziku C++. Konačno, prikazani su i komentirani dobiveni eksperimentalni rezultati primjene algoritma na stvarnim slikama.

Ključne riječi

Cannyjev detektor rubova, rubna segmentacija, pravocrtni segmenti, računalni vid, detekcija objekata.

Object boundary recognition using edge linking

Summary

We describe a procedure for triangular and rectangular traffic sign detection based on edge linking. Edge segments are extracted by Canny edge detector. The extracted edges are grouped by edge linking. Line segments are fitted to the obtained edge chains. Positions of traffic signs are hypothesized starting from the extracted line segments. Hypotheses are optimized using gradient descent, with location and size of the traffic sign being function parameters. The developed software implementation in C++ is described. Finally, experimental results of applying the algorithm on real images are provided and discussed.

Keywords

Canny edge detector, edge segmentation, line segment fitting, computer vision, object detection.