Univerza na Primorskem

Fakulteta za matematiko, naravoslovje in informacijske tehnologije

Simon Mezgec

Zaznavanje prometnih znakov za omejitev

Zaključna naloga

Koper, september 2011

Mentor: doc. dr. Peter Rogelj

Zahvala

Zahvaljujem se svoji družini - očetu Silvotu za testne vožnje, mami Vlasti za potrpežljivost in bratu Alešu za vir svežih idej v najtežjih trenutkih razvoja programa. Predvsem pa se zahvaljujem mentorju doc. dr. Petru Roglju, ki je bil skozi celoten postopek razvoja programa in pisanja zaključne naloge s pogostimi razlagami, pojasnili in nasveti v izredno pomoč. Brez vseh vas nastanek te zaključne naloge ne bi bil mogoč!

Kazalo

Povzetek

Zaključna naloga je razdeljena na dva glavna dela: teoretični in praktični del. V teoretičnem delu se nahaja opis teorije postopkov, uporabljenih v praktičnem delu, torej pri izdelavi programa za zaznavanje prometnih znakov za omejitev. Ti postopki so barvna segmentacija, zaznavanje robov in zaznavanje krogov. Pri barvni segmentaciji je poleg segmentacije opisan tudi barvni prostor HSV, ki je pomemben pri implementaciji programa. Za tem je opisano zaznavanje robov in razširjeni gradientni robni operatorji: Robertsov, Sobelov in Prewittov operator. Na koncu se nahaja še zaznavanje krogov, pri katerem je največ pozornosti namenjeno Houghovem transformu in njegovim optimizacijam, poleg tega pa je opisan tudi postopek zaznavanja krogov s pari gradientnih vektorjev. V praktičnem delu je najprej podrobno opisana koda izdelanega programa za zaznavanje prometnih znakov, nato postopki in načini testiranja ter določanja mejnih vrednosti, kot zadnje pa so predstavljeni še rezultati testiranja. V zaključku je zaključna naloga na kratko povzeta in predstavljene so pomanjkljivosti ter možne optimizacije programa za zaznavanje prometnih znakov, poleg tega pa so opisane tudi možne nadgradnje programa.

Ključne besede: prometni znaki za omejitev, barvna segmentacija, zaznavanje robov, zaznavanje krogov, barvni prostor HSV, robni operator, Houghov transform.

Abstract

This thesis is divided into two main parts: the theoretical and practical part. In the first part the theory behind the procedures, used in the practical part for the development of the speed limit traffic sign detection program, is presented. These procedures are: color segmentation, edge detection and circle detection. Along with general concepts of segmentation there is a description of the HSV color space which is important for the implementation of the program. After that three widely used gradient edge operators are described: the Roberts, Sobel and Prewitt operator. At the end of the theoretical chapter there is the description of circle detection, where the emphasis is on the Hough transform and its optimizations. Circle detection using gradient pair vectors is also presented here. In the practical part the code of the speed limit traffic sign detection program and testing process for the setup of necessary boundary values are thoroughly described. After that, the results of the testing process are presented. In the final part of the thesis there is a brief summary of the thesis and then ideas are presented about how the speed limit traffic sign detection program could be optimized and improved.

Keywords: speed limit traffic signs, color segmentation, edge detection, circle detection, HSV color space, edge operator, Hough transform.

Poglavje 1

Uvod

Zaznavanje prometnih znakov je eden od pomembnih mejnikov na poti k avtonomnim avtomobilom. Program za zaznavanje prometnih znakov, ki je v realnem času zmožen obdelati in prikazovati zadnje zaznane prometne znake, pomaga razbremeniti voznika in s tem posledično zveča varnost in udobje pri vožnji. Če se k temu doda še razpoznavanje prometnih znakov, lahko v primeru prometnih znakov za omejitev program v navezi s tempomatom vozila samodejno regulira hitrost vožnje glede na zadnjo zaznano omejitev, kar še dodatno olajša vožnjo [1].

To je bila tudi glavna motivacija pri odločanju glede teme zaključne naloge. Po izbiri teme je sledila opredelitev ciljev. Že na začetku se nismo hoteli osredotočiti le na teorijo, ampak smo začeli s ciljem samostojno ustvariti praktični ter delujoči program za zaznavanje in morda tudi razpoznavanje prometnih znakov. Kmalu pa se je pokazalo, da bi bila tovrstna tema daleč preveč obsežna za to zaključno nalogo. Zato smo se odločili implementirati samo zaznavanje prometnih znakov in se omejili na prometne znake za omejitev, ki imajo dve pomembni skupni lastnosti: vsi prometni znaki za omejitev so okrogli in imajo zunanji rob rdeče barve. Zaradi tega je implementacija programa za zaznavanje olajšana, saj je za zaznavanje teh znakov potrebno zaznati le eno barvo (rdečo) ter samo eno obliko (krog) namesto večih barv in oblik. S tem smo prišli na primeren obseg za zaključno nalogo.

Kljub temu pa je cilj ostal ustvariti hiter in zanesljiv ter posledično uporaben program za zaznavanje prometnih znakov za omejitev. Zaradi tega je že kmalu postalo jasno, da bo velik poudarek poleg na samem programiranju tudi na testiranju ter določanju pogojev za zaznavanje znakov. Na koncu pa je bilo potrebno še opisati celotno teorijo vseh uporabljenih postopkov, kot tudi temeljito razložiti delovanje programa in testiranje.

Preostali del zaključne naloge je zgrajen tako, da se v drugem poglavju nahaja opis teorije zaznavanja prometnih znakov za omejitev, v drugem pa opis kode izdelanega programa za zaznavanje kot tudi razlaga testnih postopkov ter rezultati testiranja. V zaključku sledi kratek povzetek zaključne naloge in kritičen pogled na ustvarjen program. Vse vključene slike, katerih izvor se ne nahaja v virih, so rezultat lastnega dela.

Poglavje 2

Teoretični del

2.1 Barvna segmentacija

Segmentacija slike je postopek razbitja in ponovnega združevanja pikslov v pomensko smiselne skupine za nadaljnjo obdelavo. V računalniškem vidu je segmentacija eden najstarejših in najbolj široko raziskovanih problemov [11]. Eden od načinov segmentacije je barvna segmentacija, ki uporablja barvni prostor HSV. Ta segmentacija bo v podrobnosti predstavljena v praktičnem delu - v tem poglavju sledi pregled splošne segmentacije s primeri ter obrazložitev barvnega prostora HSV.

2.1.1 Splošno o segmentaciji

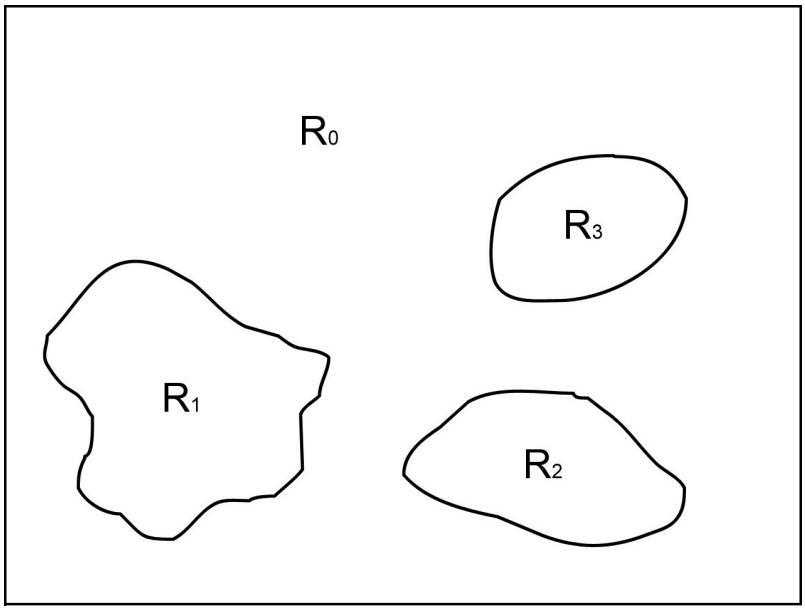
Definicija segmentacije

Segmentacija slike $f(x, y)$ je delitev $f(x, y)$ v manjše podslike $R_1, R_2, \ldots, R_n$ (kot je razvidno iz slike 2.1), tako da so zadovoljeni naslednji pogoji:

1. $\bigcup_{i=1}^{n} R_i = f(x, y)$.
2. $R_i \cap R_j = \emptyset, i \neq j$.
3. Vsaka podslika zadovoljuje trditev ali množico trditev. Par primerov teh trditev:
   - Vsi piksli v vsaki podsliki R_i imajo enak svetlostni nivo.
   - Vsi piksli v vsaki podsliki R_i se ne razlikujejo za več kot ΔX svetlostnih nivojev.
   - Vsi piksli v vsaki podsliki R_i se ne razlikujejo za več kot ΔX od povprečja svetlostnih nivojev v tej podsliki.
   - Standardna deviacija svetlostnih nivojev v vsaki podsliki R_i mora biti majhna.

Preprosta segmentacija

V preprostih primerih, kjer $f(x, y)$ vsebuje en objekt, se lahko svetlostno sliko spremeni v ustrezno binarno sliko $B(x, y)$. V tej sliki so piksli, ki ležijo na objektu, enice, ostali piksli



Slika 2.1: Slika, ki je razdeljena (segmentirana) v več podslik R_i. Za vsako posamezno podsliko velja neka trditev ali množica trditev, ki je lastna samo njej.

pa ničle. Za določanje binarne slike je potrebna neka meja M, tako da velja:

$$B(x,y) = \begin{cases} 1, & \text{če } f(x,y) < M \\ 0, & \text{sicer} \end{cases} \tag{2.1}$$

Nekatere variacije zgornjega primera vsebujejo dve meji, M_1 in M_2, ali celo polje mej, $Z = \{M_1, M_2, \ldots, M_k\}$ [9].

$$B(x,y) = \begin{cases} 1, & \text{če } M_1 < f(x,y) < M_2 \\ 0, & \text{sicer} \end{cases} \tag{2.2}$$

$$B(x,y) = \begin{cases} 1, & \text{če } f(x,y) \in Z \\ 0, & \text{sicer} \end{cases} \tag{2.3}$$

Meje in histogrami

Porazdeljenost svetlostnih nivojev se lahko uporabi za določanje meje za ustvarjanje binarnih slik. Histogram segmentirane slike predstavlja število pikslov v sliki z določenim svetlostnim nivojem. Za umetne slike z majhnim številom objektov histogram vsebuje jasne vrhove - če ima slika na primer en objekt, bo histogram segmentirane slike vseboval dva stolpca: eden v vrednosti, ki označuje piksle objekta (recimo za črno barvo je ta vrednost 0), drugi pa ponavadi pri 255, ki običajno predstavlja vrednost pikslov povsod drugje. Histogram realne slike ponavadi ne vsebuje tako strmih robov in jasnih vrhov. Namesto tega je sestavljen iz hribov in dolin, ki predstavljajo prehajanje iz ozadja na določen objekt in obratno.

Objekti s približno enakimi svetlostnimi nivoji sestavljajo razred. Histogram segmentirane slike ima vrh za vsak razred objektov in en velik vrh, ki predstavlja ozadje. Za razlikovanje med k različnimi razredi objektov je potrebno izbrati k mej - eno za vsako razmejitev sosednjih vrhov. V tabeli 2.1 in na sliki 2.2 je predstavljena slika, ki sestoji iz treh objektov. Na histogramu so štirje vrhovi - tri za objekte in eden za ozadje.

Tabela 2.1: Tabela, ki prikazuje vrednosti primera 8×8 slike s tremi objekti z vrednostmi 50, 100 in 200, vrednost 255 pa predstavlja ozadje.

255	255	255	255	255	255	255	255
255	100	100	255	255	255	255	255
255	100	100	255	255	255	255	255
255	255	255	255	255	255	255	255
200	200	200	255	255	255	255	255
200	200	200	255	255	255	255	255
200	200	200	255	255	50	50	50
255	255	255	255	255	50	50	50

S pomočjo mej se lahko ustvari tri binarne slike:

$$B_1(x,y) = \begin{cases} 1, & \text{če } 0 < f(x,y) < M_1 \\ 0, & \text{sicer} \end{cases} \tag{2.4}$$

$$B_2(x,y) = \begin{cases} 1, & \text{če } M_1 < f(x,y) < M_2 \\ 0, & \text{sicer} \end{cases} \tag{2.5}$$

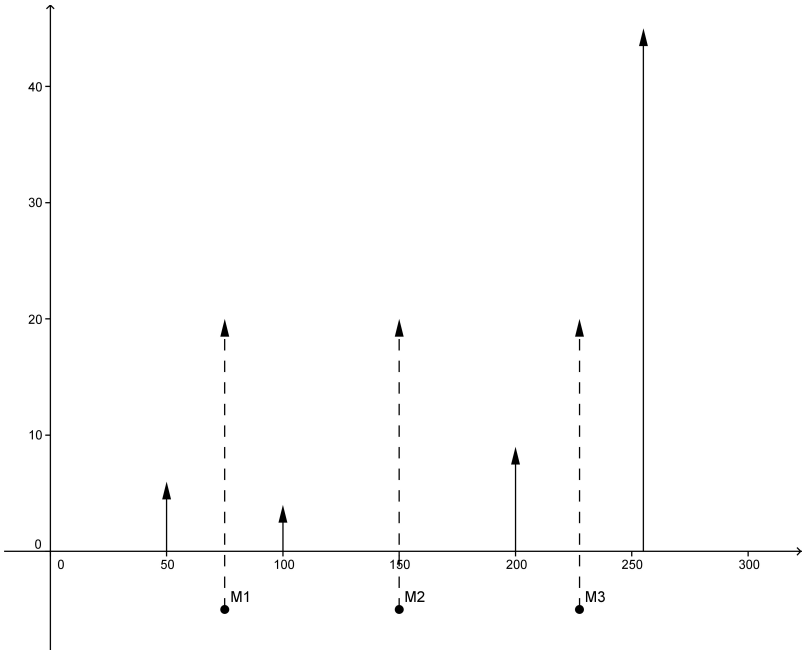
$$B_3(x,y) = \begin{cases} 1, & \text{če } M_2 < f(x,y) < M_3 \\ 0, & \text{sicer} \end{cases} \tag{2.6}$$

Pri histogramu segmentirane slike je potrebno poudariti, da lahko razlikuje le med razredi objektov in ne vsebuje nobene prostorske informacije. Naslednji primer treh slik prikaže posledico tega dejstva, saj imajo vse tri enak histogram [8]:

1. Slika, ki je do polovice črna in od polovice naprej bela,
2. Slika šahovnice z izmenjujočimi se kvadrati črne in bele barve v enakem številu,
3. Slika naključnih črnih in belih pikslov z enako porazdelitvijo.

2.1.2 Barvni prostor HSV

Kratica HSV pomeni “hue”, “saturation” in “value”, oziroma v slovenskem jeziku - odtenek, nasičenost in vrednost. Barvni prostor HSV se torej razlikuje od bolj razširjenega prostora RGB (“red”, “green”, “blue” - rdeča, zelena, modra) v zelo pomembni lastnosti - HSV loči intenzivnost barve, ki je podana v vrednosti, od barvne informacije, dobljene

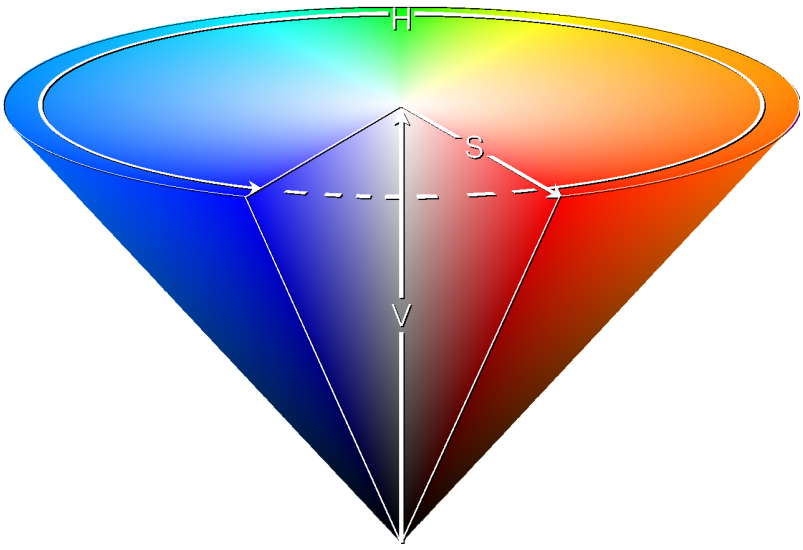


Slika 2.2: Histogram segmentirane slike za primer iz tabele 2.1. Vidni so štirje vrhovi - eden za vsak objekt na sliki in eden za ozadje. Prikazane so tudi meje, s katerimi so objekti ločeni v razrede.

iz odtenka in nasičenosti. To je posebej uporabno pri barvni segmentaciji, kjer je barvna informacija pomembnejša od intenzivnosti barve. Značilnost prostora HSV je tudi, da odtenek barve ni odvisen od jakosti osvetlitve. Barvni prostor HSV je predstavljen na sliki 2.3.

Sredinska navpična os prostora HSV predstavlja intenzivnost barve, medtem ko odtenek predstavlja kot med 0 in 2π glede na rdečo os, z rdečo barvo pri vrednosti 0, zeleno pri $2\pi/3$, modro pri $4\pi/3$ in rdečo spet pri 2π. Nasičenost predstavlja globino oziroma "čistost" barve in je definirana kot oddaljenost od sredinske osi, z vrednostmi od 0 v središču do 1 na zunanjem robu. Za $S = 0$ se s pomikanjem od najnižje točki proti najvišji po sredinski osi dobi od črne preko odtenkov sive do bele barve.

Če sta intenzivnost in odtenek dana, se z naraščanjem nasičenosti od 0 do 1 barva spreminja od odtenka sive do najbolj čiste barve, ki jo določa dan odtenek. Oziroma povedano drugače - vsaka barva v barvnem prostoru HSV je lahko spremenjena v odtenek sive z zadostnim znižanjem nasičenosti, kjer odtenek sive določa intenzivnost barve. Ko je vrednost nasičenosti blizu 0, so si vsi piksli podobni, tudi če imajo drugačen odtenek, z višanjem nasičenosti pa se ta podobnost manjša, dokler se ne pri vrednosti 1 vsi odtenki med seboj razlikujejo. Za nizke vrednosti nasičenosti se za približek barve vzame vrednost sive, ki je določena z nivojem intenzivnosti, pri visokih vrednostih pa se vzame odtenek barve. Meja, ki določa, kdaj vzeti katero od vrednosti, je spet odvisna od intenzivnosti. Pri nizki intenzivnosti je tako tudi pri visoki vrednosti nasičenosti barva bližje sivi vrednosti, pri visoki intenzivnosti pa ravno obratno [10].



Slika 2.3: Stožec, ki predstavlja vse tri dimenzije barvnega prostora HSV - odtenek (H) in nasičenost (S), ki določata barvno informacijo, ter vrednost (V), ki določa intenzivnost in po sredini stožca poteka od črne do bele barve [14].

2.2 Zaznavanje robov

Robovi objektov se ponavadi pokažejo kot prekinitve v intenzivnosti na sliki. Poskusi s človekovim vidom kažejo, da so robovi na sliki izredno pomembni, saj je lahko objekt pogosto prepoznan le s pomočjo grobega obrisa [2]. To dejstvo je glavna motivacija za predstavljanje objektov z njihovimi robovi. Poleg tega se lahko zaznavanje robov na preprost način integrira v široko paleto algoritmov za zaznavanje objektov.

Algoritmi pa le stežka najdejo robove objektov direktno iz črno-belega rezultata segmentacije, saj so robovi včasih kompleksnih oblik. Veliko večji uspeh dosežejo, če se sliko najprej spremeni v vmesno sliko lokalnih robov, in se nato te sestavi v bolj zapleten rob. Lokalni rob je majhno območje slike, kjer se svetlostni nivo hitro spremeni. Robni operator (ang. *edge operator*) je matematični operator (oziroma njegova računska ustreznica) z majhnim prostorskim obsegom z namenom zaznavanja prisotnosti lokalnega roba v funkciji slike [5].

Kot rob lahko algoritem zazna več različnih primerov:

- Dejanski rob objekta, ki razmejuje dva ali več objektov,
- Razmejitev dveh različnih lastnosti istega objekta (primer: barvne razlike),
- Razmejitev dveh različno osvetljenih delov istega objekta.

Zaradi tega je potrebno biti pazljiv pri zaznavanju robov, saj včasih ni zaželjeno, da se kot rob upoštevajo tudi razmejitve znotraj posameznega objekta, namesto le meje med več

različnimi objekti.

Kljub temu, da se robni operatorji med seboj razlikujejo, pa ima večina vsaj eno skupno lastnost, in sicer računanje smeri, v kateri je poravnana največja svetlostna sprememba, ter magnitude, ki opisuje stopnjo te spremembe. Ker je pogostost robov na slikah visoka, so algoritmi za iskanje robov ponavadi občutljivi za visoko-frekvenčni šum, kot je recimo "sneg" na televizijskem zaslonu.

Robni operatorji se delijo na tri glavne razrede:

1. Operatorji, ki aproksimirajo operator matematičnega gradienta,
2. Operatorji ujemajočih se predlog, ki uporabljajo več predlog za različne orientacije,
3. Operatorji, ki preverjajo lokalne intenzitete s parametričnimi modeli robov.

Parametrični modeli ponavadi zajamejo bolj natančno strukturo roba kot pa tisti, ki uporabljajo usmerjenost in vektor magnitude, ampak so ravno zaradi tega tudi bolj računsko zahtevni.

2.2.1 Tipi robnih operatorjev

Najbolj pogost robni operator je gradientni operator. Za funkcijo slike $f(x)$ izračuna magnitudo gradienta $s(x)$ in usmerjenost $\phi(x)$ kot:

$$s(x) = (\Delta_1^2 + \Delta_2^2)^{1/2} \tag{2.7}$$

$$\phi(x) = \arctan(\Delta_2, \Delta_1), \tag{2.8}$$

kjer je:

$$\begin{aligned} \Delta_1 &= f(x+n, y) - f(x, y) \\ \Delta_2 &= f(x, y+n) - f(x, y) \end{aligned} \tag{2.9}$$

n je majhno celo število, ponavadi v velikosti enega piksla slike, $\arctan(x, y)$ pa vrne $\tan^{-1}(x/y)$, prilagojeno za ustrezni kvadrant. Parameter n se imenuje "razpon" gradienta in mora biti dovolj majhen, da je gradient dober približek lokalnih sprememb funkcije slike, ampak vseeno dovolj velik, da se ne odziva na majhne variacije funkcije f oziroma šum.

Enačba (2.9) vsebuje le en operator razlike v svetlostni intenzivnosti, razdeljenega na dve komponenti Δ_1 in Δ_2 - eno za vsako ortogonalno smer. Osnovna verzija takega operatorja se imenuje Robertsov operator. Zaradi manjše občutljivosti na šum pa sta v večini praktičnih primerov bolj uporabna Sobelov in Prewittov operator, ki sta modificirana Robertsova operatorja [2].

Robertsov operator

Robertsov operator izračuna preprost približek magnitude gradienta:

$$G[f(i,j)] = |f(i,j) - f(i+1,j+1)| + |f(i+1,j) - f(i,j+1)| \tag{2.10}$$

Z uporabo sledečih konvolucijskih mask (ang. *convolution masks*):

$$G_x = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & \text{-1} \\ \hline \end{array} \tag{2.11}$$

$$G_y = \begin{array}{|c|c|}\hline 0 & \text{-1} \\ \hline 1 & 0 \\ \hline \end{array}$$

se enačba (2.10) spremeni v:

$$G[f(i,j)] = |G_x| + |G_y| \tag{2.12}$$

Razlike se kot pri splošnem gradientnem operatorju izračunajo v interpolirani točki $[i + \frac{1}{2}, j + \frac{1}{2}]$. Robertsov operator je približek kontinuiranega gradienta v tej točki in ne v točki $[i, j]$.

Sobelov operator

Z uporabo 3×3 okolja se je možno izogniti računanju gradienta z interpolirano točko. Z naslednjo matriko okolja:

a_0	a_1	a_2
a_7	$[i,j]$	a_3
a_6	a_5	a_4

(2.13)

se Sobelov operator izračuna kot magnituda gradienta:

$$M = \sqrt{s_x^2 + s_y^2}, \tag{2.14}$$

kjer se delni odvodi izračunajo kot:

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \tag{2.15}$$

$$s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4) \tag{2.16}$$

s konstanto $c = 2$. Kot pri ostalih gradientnih operatorjih, se lahko s_x in s_y implementira s pomočjo konvolucijskih mask [4]:

$$s_x = \begin{array}{|c|c|c|}\hline \text{-1} & 0 & 1 \\ \hline \text{-2} & 0 & 2 \\ \hline \text{-1} & 0 & 1 \\ \hline \end{array} \tag{2.17}$$

$$s_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

Potrebno je izpostaviti dejstvo, da ta operator zaradi konstante $c = 2$ daje poudarek pikslom, ki so bližje sredini konvolucijske maske. Sobelov operator je eden izmed najbolj razširjenih in uporabljenih robnih operatorjev [3].

Prewittov operator

Prewittov operator uporablja iste enačbe kot Sobelov operator, izjema je le konstanta c, ki je v tem primeru enaka $c = 1$. Konvolucijski maski za ta operator sta torej:

$$s_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \tag{2.18}$$

$$s_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

Prewittov operator za razliko od Sobelovega ne daje poudarka pikslom, ki se nahajajo blizu sredine konvolucijske maske.

Primerjava robnih operatorjev

Na sliki 2.4 je primerjava vseh treh opisanih robnih operatorjev - Robertsovega, Sobelovega ter Prewittovega.

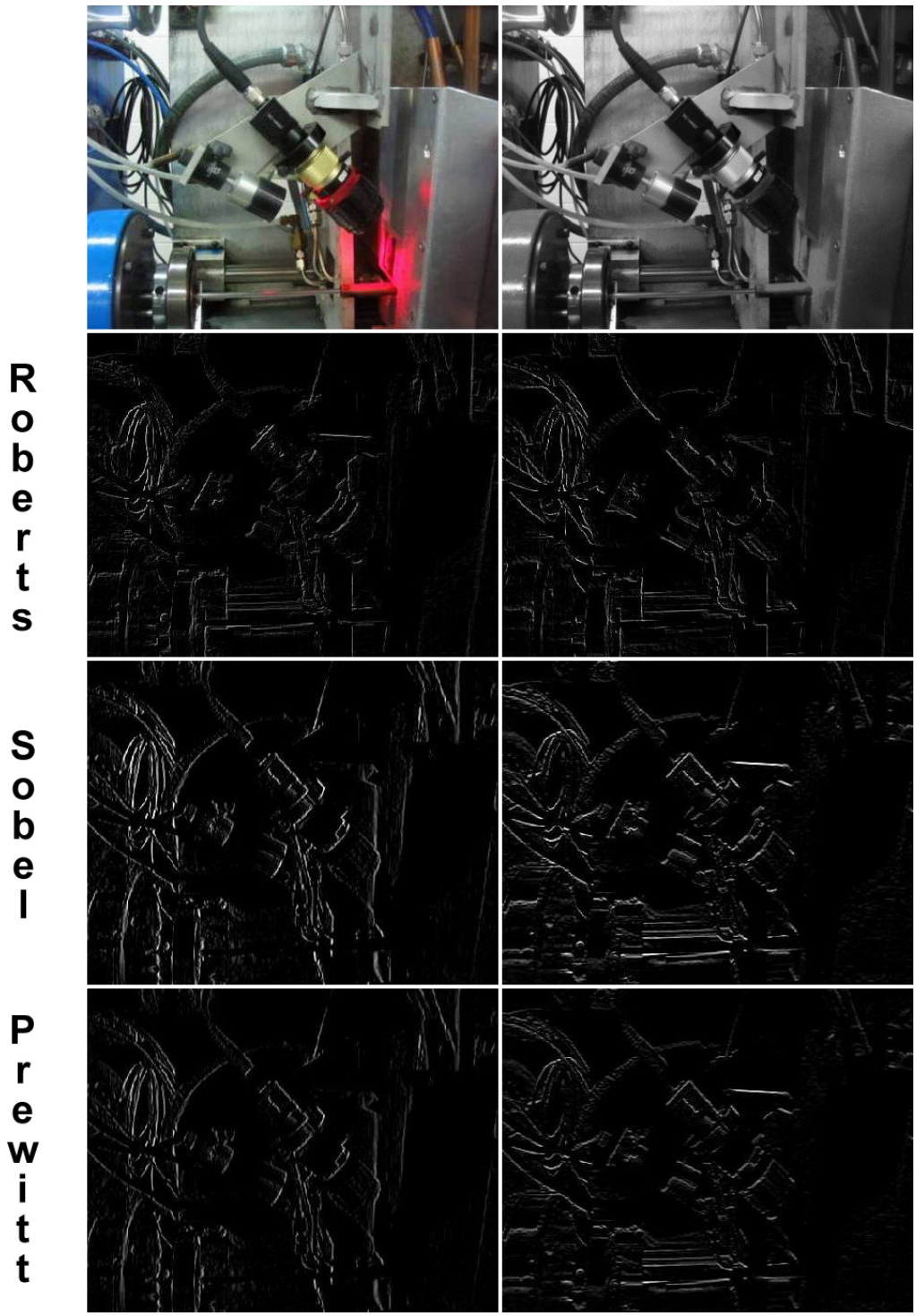
2.3 Zaznavanje krogov

Krogi so pogost predmet zanimanja v aplikacijah računalniškega vida. V tem odseku bodo predstavljeni postopki za iskanje krogov na segmentirani sliki z zaznanimi robovi. Najbolj pogosto uporabljen postopek je Houghov transform, ker pa spada med bolj računsko zahtevne algoritme, obstajajo optimizacije, ki postopek pohitrijo. Poleg tega pa obstajajo tudi drugačni postopki za iskanje krogov - primer takega postopka je zaznavanje krogov s pomočjo parov gradientnih vektorjev.

2.3.1 Houghov transform

Houghov transform se uporablja za določanje parametrov kroga, ko je znanih več točk na krožnici iskanega kroga. Krog s polmerom r in središčem (a, b) se lahko predstavi s parametričnima enačbama [12]:

$$x = a + r\cos(\theta) \tag{2.19}$$



Slika 2.4: Prikaz različnega delovanja treh gradientnih robnih operatorjev. Na vrhu levo se nahaja izvorna slika, desno od nje njena črno-bela različica, na kateri so aplicirani operatorji, potem pa po vrsti od zgoraj navzdol: Robertsov, Sobelov in Prewittov robni operator. V levem stolpcu so uporabljene konvolucijske maske operatorjev za spremembe v smeri koordinate x, v desnem pa v smeri koordinate y. Izvorna slika je pridobljena iz [13].

$$y = b + r\sin(\theta) \tag{2.20}$$

S pomikanjem kota θ čez celoten 360-stopinjski razpon, se iz točk (x, y) kot rezultat dobi veliko število različnih krožnic, podanimi s parametri (a, b, r). Med njimi je tudi iskana krožnica. Če slika vsebuje veliko število točk, med katerimi se nekatere nahajajo na krožnicah določenih krogov, druge pa ne, je naloga postopka iskanje parametrskih trojic (a, b, r), ki opišejo vsak krog. Posledica tega, da je polje iskanja trirazsežno, je dejstvo, da je implementacija Houghovega transforma bolj računsko zahtevna kot implementacija segmentacije ali zaznavanja robov.

Za iskanje krogov z znanim polmerom r se lahko iskalni prostor zmanjša na dvorazsežnega. Naloga je poiskati koordinate središč krogov (a, b) s pomočjo enačb (2.19) in (2.20). Houghov transform v tem primeru vzame vsako robno točko s koordinatama (x, y) za središče in nanjo nariše krožnico s polmerom r oziroma označi, kje bi se za to točko lahko nahajalo središče iskane krožnice. Vse dobljene krožnice se sekajo v točki s koordinatama (a, b), ki predstavlja središče iskanega kroga.

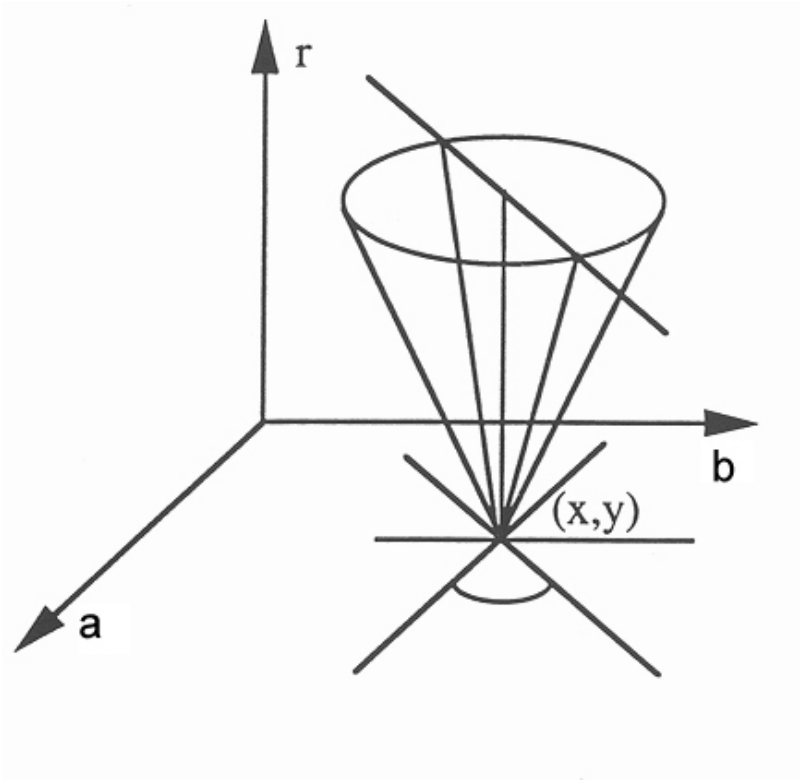
Če je naloga iskanje večih krogov z znanim polmerom r, se lahko krožnice s središči v robnih točkah (x, y) med seboj sekajo in prekrivajo. Posledica tega je lahko več zaznanih središčnih točk, kot pa je dejanskih krogov. Ta problem se lahko reši s primerjanjem najdenih krogov z izvorno sliko.

V večini primerov pa polmer r ni znan. Tedaj vsaka robna točka (x, y) proizvede površino stožca v parametrskem prostoru (a, b, r). Trojica (a, b, r) ustreza iskanemu krogu s središčem v točki trirazsežnega prostora, kjer se seka največje število stožcev. Slika 2.5 predstavlja generiranje površine stožca v parametrskem prostoru (a, b, r) za eno robno točko (x, y). Vsaka rezina površine stožca ustreza krogu z različnim iskanim polmerom [7].

Optimizacije Houghovega transforma

Možnih je več optimizacij Houghovega transforma. Pri rezultatu zaznavanja robov je znana tudi približna usmeritev roba. Zato se lahko risanje krožnic s središčem v robnih točkah (x, y) omeji na krožna loka, ki oklepata normalo usmeritve roba. S tem bo krog na sliki še vedno zaznan, postopek pa se občutno pohitri, saj ni več potrebno risati celotne krožnice za vsako robno točko. Poleg tega se s tem ponavadi zmanjša število napačno zaznanih (neobstoječih) krogov, saj je možnost, da se bodo krožnice s središči v robnih točkah (x, y), sekale v točkah, ki ni središče iskanega kroga (a, b), manjša.

Pogosta pa je še ena optimizacija Houghovega transforma. Skozi vsako robno točko kroga se namesto krožnice nariše bodisi navpično ali pa vodoravno premico. Če se na tej premici nahaja še kakšna druga robna točka, se na daljico med tema dvema točkama nariše simetralo. Središče iskanega kroga se tako nahaja na tej simetrali, ki se jo lahko omeji na dolžino $2r$, saj bo središče kroga oddaljeno kvečjemu za polmer r v obe smeri. Ko se postopek ponovi za vse robne točke, bo imelo središče kroga največ glasov. Prednost tega



Slika 2.5: Generiranje površine stožca za dano robno točko s koordinatama (x, y) v tri-razsežnem parametrskem prostoru (a, b, r), kjer sta (a, b) koordinati središča iskanega kroga, r pa njegov polmer [15].

postopka je večja hitrost, saj ni potrebno risati krožnice za vsako robno točko, ampak le simetralo. Obe optimizaciji sta prikazani na sliki 2.6.

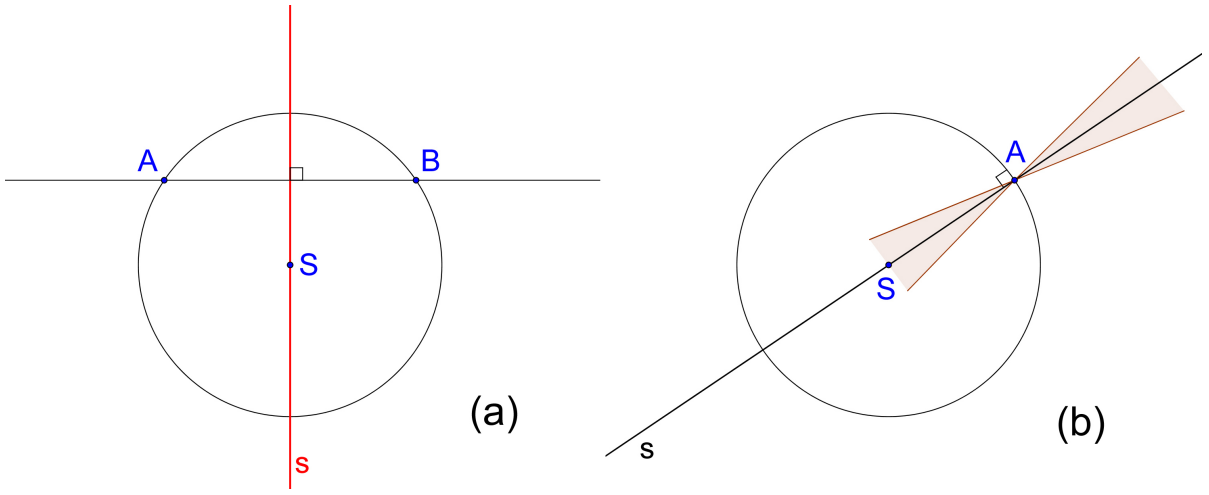
2.3.2 Zaznavanje krogov s pari gradientnih vektorjev

Glavna motivacija zaznavanja krogov s pari gradientnih vektorjev je, da se lahko s pomočjo predhodne obdelave dobi dodatne informacije o sliki in s tem pohitri delovanje algoritma zaznavanja krogov v primerjavi s Houghovim transformom.

Ob predpostavki, da se krog nahaja na svetlejšem ozadju, so izračunani gradientni vektorji kroga usmerjeni navzven, kar je razvidno iz slike 2.7. Ker je krog simetričen glede na svoje središče, vsak vektor tvori par z nasprotno usmerjenim vektorjem na nasprotni strani kroga. Na sliki 2.8a je predstavljen vektor $\overrightarrow{v_1}$, ki ima parni vektor $\overrightarrow{v_2}$, za katerega morata biti izpolnjena naslednja pogoja:

1. Kot α, definiran kot absolutna razlika med usmerjenostjo vektorja $\overrightarrow{v_1}$ in usmerjenostjo vektorja $\overrightarrow{v_2}$, mora biti blizu 180°,

2. Kot β med daljico, ki povezuje točki P_2 in P_1 (izhodiščcni točki za vektorja $\overrightarrow{v_2}$ in $\overrightarrow{v_1}$), ter vektorjem $\overrightarrow{v_1}$, mora biti skoraj 0°, kar pomeni, da mora biti vektor $\overrightarrow{P_2P_1}$ približno



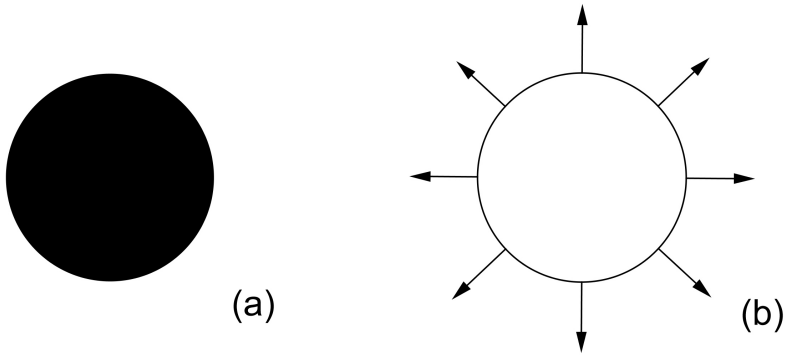
Slika 2.6: Obe sliki prikazujeta različni optimizaciji Houghovega transforma. Na sliki (a) je prikazana optimizacija s pomočjo iskanja nasprotne robne točke B točki A in risanjem simetrale na daljico med njima. Središče iskanega kroga se nahaja na simetrali s. Slika (b) pa prikazuje optimizacijo z omejevanjem risanja krožnice v robni točki A - namesto celotne krožnice s središčem v točki A je potrebno narisati le krožna loka (na vsako stran točke A enega), ki oklepata normalo usmeritve s robne točke A.

enako usmerjen kot vektor $\overrightarrow{v_1}$.

Drugi korak algoritma je apliciranje zgornjih dveh pogojev na vse pare vektorjev. Drugi pogoj znatno zmanjša šum s filtriranjem neuporabnih vektorjev. Kot je predstavljeno na sliki 2.8b, vektorja $\overrightarrow{v_1}$ in $\overrightarrow{v_2}$ nista obravnavana kot par vektorjev, saj kljub izpolnjenemu prvemu pogoju ne izpolnjujeta drugega pogoja. Za pohitritev postopka iskanja parnega vektorja se lahko polje dobljenih vektorjev uredi glede na usmerjenost, tako da je iskanje nasprotno usmerjenih vektorjev olajšano.

V tretjem koraku se za vsak par vektorjev vzame kandidata za iskani krog. Tak krog ima središče na sredini daljice, ki povezuje točki P_1 in P_2 in njegov polmer meri polovico dolžine te daljice. Slika 2.8a prikazuje kandidata za iskani krog. V posebnih primerih - če je polmer iskanega kroga znan, se lahko uporabi dodaten pogoj za filtriranje parnih vektorjev, in sicer v primeru, ko je razdalja med parnima vektorjema izven predvidenega polmera. Takrat se tak par vektorjev ne upošteva. Če je vsaj nekoliko znanega o razponu iskanih polmerov, se učinkovitost algoritma drastično zviša [6].

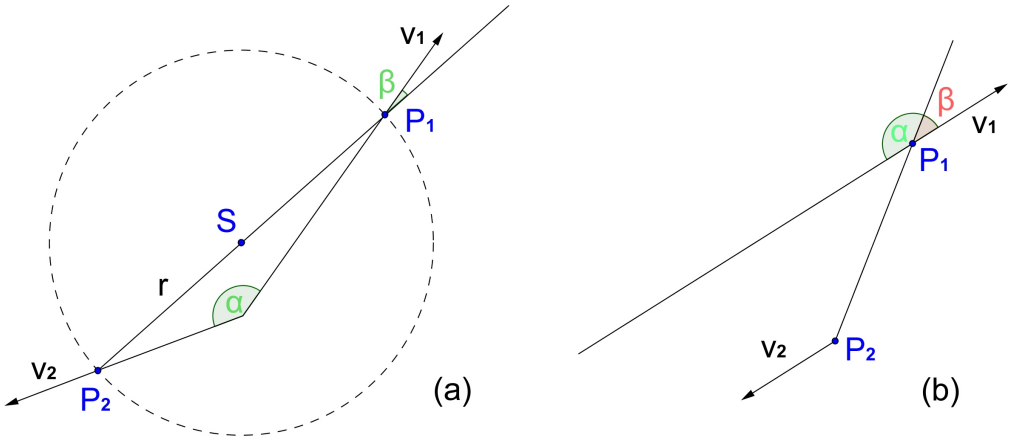
V zadnjem, četrtem koraku so želeni krogi pridobljeni iz kandidatov za kroge, ki so bili najdeni v prejšnjem koraku. Za to obstajata dva načina. Prvi je uporaba trirazsežne matrike za štetje pojavov krogov. Na ta način deluje prej opisani Houghov transform. Ker pa je seznam kandidatov za krog znan, je drugi in lažji način iskanje točnih krogov. Kandidate za krog se lahko shrani kot polje trojic (S_x, S_y, r) in razvrsti v skupine glede na evklidsko razdaljo med njimi. V tem primeru se lahko išče želeje kroge s pomočjo predstavnikov teh



Slika 2.7: (a) Črn krog na belem ozadju, (b) Gradientni vektorji za (a).

skupin. V primerjavi s klasičnim pristopom je ta postopek bolj učinkovit in manj prostorsko zahteven [6].

Pred implementacijo teh štirih glavnih korakov pa je priporočena predpriprava slike za hitrejše in bolj učinkovito delovanje algoritma. Kot je bilo omenjeno na začetku tega odseka, algoritem išče kroge, ki so bodisi temnejši ali pa svetlejši od ozadja - slednje se lahko doseže s pomočjo negativa slike ali pa z uporabo nasprotno usmerjenih gradientnih vektorjev. Zaradi tega je priporočljiva predhodna obdelava glede na značilnosti konkretne slike. Na področju zaznavanja zenice v slikah očesa je primer take obdelave uporaba Gaussovega filtra za mehčanje slike in izdelava binarne slike s pomočjo ustrezne meje svetlostnih nivojev. V tej binarni sliki zenica izgleda kot črn krog in ima značilnosti, primerne za algoritem parov gradientnih vektorjev. V splošnem pa metode predhodne obdelave za izboljšanje učinkovitosti algoritma na različnih področjih obsegajo mehčanje, zaznavanje robov, barvno segmentacijo in rezanje neuporabnih delov slike.



Slika 2.8: (a) Par vektorjev in njun kandidat za središče iskanega kroga, (b) Zavrnjena vektorja zaradi neizpolnjenega drugega pogoja.

Poglavje 3

Praktični del

3.1 Opis kode

V tem odseku bo predstavljena koda programa in razloženo njeno delovanje. Programski jezik pisanja kode kot tudi jezik vključene kode je C++ z zunanjo odprtokodno knjižnico za računalniški vid OpenCV, vmesnik 2.1. Za pisanje, testiranje in prikaz delovanja kode je bilo uporabljeno programsko okolje Microsoft Visual Studio 2010.

3.1.1 Barvna segmentacija v barvnem prostoru HSV

Na sliki 3.1 se nahaja koda, ki prikazuje prvi del programa po inicializaciji spremenljivk in branju iz konfiguracijske datoteke. Pred prvim ukazom na sliki naložimo še video iz kamere ter z zanko obdelujemo vsako sliko videa. Ukaz `cvtColor` je vgrajeni ukaz knjižnice OpenCV za spreminjanje barvne sheme podane slike. Prvi parameter je vhodna slika, drugi je izhodna slika, tretji pa metoda pretvorbe barvne sheme. V tem primeru iz vhoda kamere preberemo trenutno sliko (`frame`) in jo shranimo v sliko z imenom `hsv` s pomočjo ukaza `CV_RGB2HSV`, ki sliko iz barvnega prostora RGB spremeni v sliko barvnega prostora HSV - ta pa je v podrobnosti opisan v odseku 2.1.2.

Potem se sprehodimo skozi vse piksle pretvorjene slike v barvnem prostoru HSV, kjer spremenljivka i predstavlja trenutno vrstico slike, spremenljivka j pa stolpec. Ker gre za sliko v barvnem prostoru HSV, je vsak piksel slike predstavljen s tremi vrednostmi - na mestu (i, j) se nahaja odtenek (hue), na mestu $(i, j + 1)$ je nasičenost (saturation) in na mestu $(i, j+2)$ vrednost (value). Na mestu $(i, j+3)$ se tako nahaja že naslednji piksel slike.

Zaradi tega pri prvem pogoju ne preverimo piksla na mestu (i, j), kot bi bilo bolj intuitivno, ampak na mestu `i*stephsv + j*channelshsv`. Spremenljivka `stephsv` nam pove korak slike oziroma koliko bajtov vsebuje ena vrstica slike. Zato se z zmnožkom `i*stephsv` dejansko premaknemo na začetek i-te vrstice. Spremenljivka `channelshsv` pa nam pove število kanalov slike, kar v tem primeru znaša 3, kot je opisano v prejšnjem odstavku. S celotnim ukazom `i*stephsv + j*channelshsv` se torej nahajamo na podatku o odtenku piksla na mestu (i, j) in preverimo, ali je odtenek večji ali enak spodnji omejitvi za odtenek ter manjši ali enak zgornji omejitvi.

```
1  //Pretvorimo izvorno sliko v HSV sliko in jo shranimo.
2  cvtColor(frame, hsv, CV_RGB2HSV);
3
4  //Sprehodimo se po celotni HSV sliki.
5  for (i = 0; i < heighthsv; i++)
6  {
7    for (j = 0; j < widthhsv; j++)
8    {
9      //Ce je prva vrednost (odtenek) vsakega piksla slike znotraj dolocenih
           mejnih vrednosti...
10     if ( (datahsv[i*stephsv + j*channelshsv] >= hlower) && (datahsv[i*stephsv
           + j*channelshsv] <= hupper) )
11     {
12       //... in ce je druga vrednost (nasicenost) vecja od spodnje meje...
13       if ((datahsv[i*stephsv + j*channelshsv + 1]) > sthreshold)
14       {
15         //... shranimo ta piksel kot piksel bele barve v crno-beli sliki...
16         datamono[i*stepmono + j*channelsmono] = 255;
17       }
18       else
19       {
20         //... drugace pa shranimo piksel crne barve.
21         datamono[i*stepmono + j*channelsmono] = 0;
22       }
23     }
24   }
25 }
```

Slika 3.1: Prvi glavni postopek programa za zaznavanje prometnih znakov za omejitev, kjer se nahaja barvna segmentacija v barvnem prostoru HSV. Pred tem je potrebno trenutno sliko iz kamere pretvoriti iz barvnega prostora RGB v barvni prostor HSV.

Če je odtenek znotraj mejnih vrednosti, preverimo naslednji pogoj. Pri tem se pomaknemo naprej za 1 v sliki, kar pomeni, da se sedaj nahajamo pri vrednosti za nasičenost. Tokrat preverimo, ali je nasičenost piksla večja od prej določene spodnje meje za nasičenost. Zgornje meje ne preverjamo, saj nas zanimajo vsi piksli od spodnje meje do vrednosti 255.

V primeru, da sta oba pogoja izpolnjena, shranimo vrednost 255 na mesto `i*stepmono + j*channelsmono` črno-bele slike rezultata barvne segmentacije, kjer sta tako spremenljivka `stepmono` kot `channelsmono` trikrat manjši kot pri sliki v barvnem prostoru HSV, saj je za opis črno-bele slike za vsak piksel potreben le en bajt namesto treh, in ta nam pove vrednost svetlostnega nivoja piksla. Če vrednost nasičenosti piksla ni večja od spodnje meje, pa v piksel črno-bele slike shranimo vrednost 0.

Kot rezultat tega postopka se tako na sliki z imenom `mono` na mestih, kjer je bila zaznana rdeča barva, nahajajo piksli bele barve, povsod drugod pa piksli črne barve.

3.1.2 Zaznavanje robov s Sobelovim operatorjem

Na sliki 3.2 je del kode, v katerem poteka zaznavanje robov s pomočjo Sobelovega operatorja. Najprej dvakrat poženemo vgrajen ukaz knjižnice OpenCV za Sobelov operator. Prvi parameter tega ukaza je vhodna slika, drugi izhodna slika, nato je globina izhodne slike, na koncu pa še izberemo, ali želimo pognati Sobelov operator za spremembe v x ali y smeri - za x mora biti predzadnji parameter 1 in zadnji 0, za y pa ravno obratno. Za apliciranje Sobelovega operatorja višjih odvodov bi namesto števila 1 vstavili višje število.

```
1  //Apliciramo Sobelov operator na crno-bel rezultat segmentacije rdece barve.
2  Sobel (mono, rob_x, 3, 1, 0);
3  Sobel (mono, rob_y, 3, 0, 1);
4
5  //Ustvarimo polje za zdruzevanje slik Sobelovega operatorja.
6  int *sob;
7  sob = new int [rob_x.rows * rob_x.cols];
8
9  //Zdruzimo oba (x in y) rezultata Sobelovega operatorja v eno polje.
10 for (i = 0; i < rob_x.rows; i+=2)
11 {
12   for (j = 0; j < rob_x.cols; j++)
13   {
14     int vrednostX = datarobovi_x[i*steprobovi_x + 2*j*channelsrobovi_x];
15     int vrednostY = datarobovi_y[i*steprobovi_y + 2*j*channelsrobovi_y];
16
17     double skupaj = vrednostX * vrednostX + vrednostY * vrednostY;
18     int koren = int (sqrt(skupaj));
19
20     //Vrednost shranimo v polje le, ce je vecje od mejne vrednosti.
21     if (koren >= 255)
22     {
23       sob[i*rob_x.cols + j] = koren;
24     }
25   }
26 }
```

Slika 3.2: Drugi glavni postopek programa za zaznavanje prometnih znakov za omejitev je zaznavanje robov s pomočjo Sobelovega operatorja. Najprej uporabimo vgrajena ukaza za Sobelov x operator in y operator posebej, nato pa obe sliki združimo.

Vhodna slika je v tem primeru rezultat barvne segmentacije, katere koda je opisana v odseku 3.1.1. Izhodni sliki pa sta dve, in sicer `rob_x` in `rob_y` - za spremembe po x in y koordinatah posebej. Za tem sledi shranjevanje podatkov izhodnih slik. V spremenljivkah `datarobovi_x` in `datarobovi_y` imamo shranjeni mesti v pomnilniku, kjer se izhodni sliki nahajata, v spremenljivkah `steprobovi_x` in `steprobovi_y` njuna koraka (korak slike je opisan v prejšnjem odseku), ter v spremenljivkah `channelsrobovi_x` in `channelsrobovi_y` število barvnih kanalov obeh slik. Nato ustvarimo še novo polje `sob` (enakih razsežnosti kot izhodni sliki) za shranjevanje združene slike Sobelovega operatorja.

Na koncu se sprehodimo po izhodnih slikah, kjer podobno kot pri barvni segmentaciji spremenljivka i predstavlja vrstico slike, spremenljivka j pa stolpec. Glede na barvno segmentacijo pa je razlika pri sprehodu po vrsticah i, saj smo pri segmentaciji pregledovali vsako vrstico slike, tukaj pa vsako drugo vrstico - zato spremenljivko i v vsakem koraku povečamo za 2 namesto za 1. Razlog za to je, ker potem pri Houghovem transformu in naprej zaradi hitrosti (bolj podrobno opisano v odseku 3.1.3) uporabljamo samo vsako drugo vrstico slike, zato je bolj smiselno in tudi hitreje že tukaj uporabiti le vsako drugo vrstico.

V vsakem koraku zanke najprej v spremenljivki `vrednostX` in `vrednostY` shranimo vrednosti trenutnega piksla iz izhodnih slik Sobelovega operatorja. Razlog, da obravnavamo piksel na mestu `i*steprobovi_x + 2*j*channelsrobovi_x` in ne na `i*steprobovi_x + j*channelsrobovi_x` je, da vgrajeni ukaz knjižnice OpenCV za Sobelov operator shrani rezultat kot 16-bitno sliko, medtem ko je vhodna (ter vse ostale) slika 8-bitna. Zato je dejansko potrebno gledati vsak drugi piksel.

Podatek za vsak piksel iz izhodnih slik Sobelovega operatorja nam pove magnitudo spremembe svetlostnega nivoja v tistem pikslu - pri prvi sliki spremembe po x osi, pri drugi pa po y osi. Čim večja je ta številka, tem večja je razlika v barvi po posamezni osi. Če torej hočemo izvedeti, ali se piksel nahaja na robu objekta, moramo oba podatka združiti v enega - v skupno magnitudo spremembe svetlostnega nivoja. Zato v naslednjem koraku kvadriramo spremembi po x in y osi ter ju seštejemo. Koren seštevka na koncu shranimo v spremenljivko `koren` (spremenljivka `skupaj` je uporabljena le za vmesno računanje).

Če je prej shranjeni koren večji od vrednosti 255 oziroma povedano drugače - če je skupen podatek o magnitudi spremembe svetlostnega nivoja večji od mejne vrednosti, potem v polje `sob` shranimo ta ulomek. Tukaj je potrebno poudariti, da za združevanje celotnih slik Sobelovega operatorja ne bi bilo potrebno preverjati vrednost ulomka, ampak bi shranili kar podatke za vsak piksel. Ker pa dobivamo slike iz kamere v realnem času, je vsaka optimizacija hitrosti dobrodošla in glede na to, da v naslednjem koraku (pri Houghovem transformu) potrebujemo le piksle, ki se dejansko nahajajo na robovih objektov, bi bilo shranjevanje vseh pikslov nepotrebno in potratno.

Rezultat tega koraka je polje `sob`, ki vsebuje vrednosti magnitud spremembe svetlostnega nivoja v robnih pikslih, povsod drugod v polju pa vrednost 0.

3.1.3 Zaznavanje krogov s Houghovim transformom

Pri zaznavanju krogov s Houghovim transformom je potrebno najprej omeniti razlog, zakaj se nismo odločili za uporabo temu namenjene funkcije knjižnice OpenCV - `HoughCircles`. Pri tem ukazu je možno spreminjati vrsto parametrov za iskanje krogov, ima pa kritično pomanjkljivost, zaradi katere se na koncu nismo odločili za njegovo uporabo - funkcija namreč ne podpira iskanje krogov s središčem v isti točki. Uporaba ukaza `HoughCircles` torej ni prišla v poštev, saj z njim ne bi mogli uvesti optimizacije, opisane v odseku 3.1.4, kjer iščemo koncentrične kroge. Brez te optimizacije pa bi dosegli tako nižjo zanesljivost kot tudi nižjo hitrost, zato smo se raje odločili za lastno implementacijo Houghovega transforma, ki

je opisana v naslednjih odstavkih.

Prvi korak zaznavanja krogov s Houghovim transformom je priprava polja (akumulatorja) za shranjevanje podatkov, ki je predstavljena na sliki 3.3. V tem primeru je to polje `buf` razsežnosti `range` × `rob_x.rows * rob_x.cols`. Spremenljivka `range` je bila na začetku prebrana iz konfiguracijske datoteke in nam pove obseg iskanih polmerov, spremenljivki `rob_x.rows` in `rob_x.cols` pa sta dimenziji slik Sobelovega operatorja.

```
1  //Inicializiramo polje velikosti (razpon iskanih polmerov)x(velikost slike).
2  int **buf;
3  buf = new int* [range];
4
5  for (a = 0; a < range; a+=4)
6  {
7    buf[a] = new int [rob_x.rows * rob_x.cols];
8
9    int omejitev = a + low;
10
11   //Sprehodimo se po polju Sobelovega operatorja.
12   for (i = omejitev; i < (rob_x.rows - omejitev); i+=2)
13   {
14     for (j = omejitev; j < (rob_x.cols - omejitev); j++)
15     {
16       //Ce je bila na trenutnem pikslu zaznana rdeca barva, je tu kandidat za
             rob kroga.
17       if (sob[i*rob_x.cols + j] >= 255)
18       {
19         int vrednostX = datarobovi_x[i*steprobovi_x + 2*j*channelsrobovi_x];
20         int vrednostY = datarobovi_y[i*steprobovi_x + 2*j*channelsrobovi_y];
21
22                 //Koda na sliki 3.4.
23       }
24     }
25   }
26   //Ponavljamo za vse iskane polmere.
27 }
```

Slika 3.3: Inicializacija polja za Houghov transform in zunanje zanke za izvajanje Houghovega transforma. Koda za samo izvajanje Houghovega transforma se nahaja na sliki 3.4.

Tukaj je potrebno poudariti, da se z zunanjo zanko premikamo po akumulatorju s korakom 4, kar dejansko pomeni, da iščemo kroge s polmeri, ki se razlikujejo za 4 - če na primer iščemo polmere velikosti od 1 do 20, bomo upoštevali samo polmere velikosti 4, 8, 12, 16 in 20. Razlog za to je ogromen pospešek v hitrosti delovanja programa, saj namesto iskanja čisto vseh polmerov v obsegu jih iščemo štirikrat manj, kar pomeni štirikratno pohitritev. Po drugi strani pa preskoki v iskanih polmerih v manjši meri vplivajo na rezultat zaznavanja prometnega znaka za omejitev kot preskoki v vrsticah ali stolpcih slike. Ravno zato delamo največje preskoke v iskanih polmerih.

Poleg koraka 4 pri polmerih kroga pa delamo preskoke tudi v sprehajanju po vrsticah slike, ampak zaradi prej omenjenega razloga ta preskok znaša le 2. Kljub manjšim preskokom pa je zaradi tega pohitritev delovanja programa še dodatno zmanjšana za faktor 2 - skupaj s korakom iskanja polmerov je torej pohitritev osemkratna. S tema dvema spremembama v korakih je doseženo ravnotežje med hitrostjo in zanesljivostjo. Več o testiranju in iskanju primernih vrednosti se nahaja v odseku 3.2.

Omeniti velja tudi, zakaj se z notranjima zankama ne sprehajamo od začetne vrstice in stolpca do zadnje, ampak pustimo na vsaki strani slike rob v velikosti spremenljivke `omejitev`, ki jo izračunamo kot `a + low` - torej seštevek trenutnega indeksa v polju `buf` in vrednosti najmanjšega iskanega polmera, kar je skupaj velikost trenutno iskanega polmera kroga. Razlog za to je, da tudi če bi v tem robu zaznali središče kroga, se ta krog ne bi v celoti nahajal na sliki, mi pa želimo prikazati celoten prometni znak. Poleg tega pa do tega primera pride samo, ko smo z vozilom že skoraj vzporedno s prometnim znakom, znak pa program zazna veliko prej. Zato s to omejitvijo ne izgubimo nič, pridobimo pa nekoliko na hitrosti, saj za vsak iskan polmer preskočimo omenjen rob.

Ko se enkrat nahajamo v zanki, najprej preverimo, ali je vrednost polja `sob` v trenutnem pikslu večja od vrednosti 255, torej ali je bil v tem pikslu zaznan rob. V primeru, da se tam nahaja rob, si najprej shranimo magnitudi sprememb svetlostnega nivoja po x in y koordinati v spremenljivki `vrednostX` ter `vrednostY`. Ti spremenljivki bomo potem potrebovali za optimizacijo Houghovega transforma s pomočjo določanja usmeritve roba. Zaradi večje preglednosti se koda, ki sledi, nahaja na sliki 3.4.

V prvem pogoju preverimo, ali je tako spremenljivka `vrednostX` kot `vrednostY` višja od mejne vrednosti 250. To pomeni, da je bila zaznana velika magnituda spremembe tako po x osi kot po y osi. V tem primeru s prvima dvema zankama pregledujemo spodnjo desno četrtino krožnice, z drugima dvema pa nasprotno četrtino, torej zgornjo levo. V vsakem koraku zank najprej izračunamo oddaljenost trenutnega piksla od piksla, kjer je bil zaznan rob. Če je ta oddaljenost enaka vrednosti `a + low` oziroma polmeru iskanega kroga, se nahajamo na krožnici s središčem v robni točki (i, j) in povečamo vrednosti na tej krožnici v polju `buf` za 1. Na teh delih krožnice se torej nahajajo piksli, ki so kandidati za središče iskanega kroga.

Pri naslednjem pogoju preverimo, ali je samo ena od magnitud spremembe večja od mejne vrednosti 250. Če ta pogoj drži, potem je bila zaznana velika magnituda spremembe samo po eni od osi x in y ter ponovimo postopek, opisan v prejšnjem odstavku, le da najprej vzamemo spodnjo levo četrtino krožnice in potem še zgornjo desno. Skupaj s prejšnjim pogojem smo tako pokrili celotno krožnico. Namesto povečanja polja `buf` za točke celotne krožnice v vsakem koraku smo torej razdelili robne točke (i, j) v dva razreda glede na njihovo usmerjenost. V enem primeru obravnavamo en par nasprotnih četrtin kroga, v drugem pa ostala dva. Gre za optimizacijo Houghovega transforma, ki je opisana v odseku 2.3.1 ter predstavljena na sliki 2.6b.

Na koncu pa dodamo še varnostno zanko, v katero spadajo vsi primeri, ki niso bili zajeti

```
1  //Ce sta tako x in y rezultat Sobelovega operatorja nad mejo, gledamo samo
       dve (nasprotni) cetrtini kroznice.
2  if (vrednostX >= 250 && vrednostY >= 250)
3  {
4    for (k = i - 1; k < i + (a + (low+1)); k++)
5    {
6      for (l = j - 1; l < j + (a + (low+1)); l++)
7      {
8        //Razdalja sredisca do kroznice (po Pitagorovem izreku).
9        double razdalja = (l-j) * (l-j) + (k-i) * (k-i);
10       int koren2 = int (sqrt(razdalja));
11
12       //Na prej omenjenem pikslu vzamemo sredisce kroga in tockam na kroznici
             povecamo polje za ena.
13       if (koren2 == (a + low))
14       {
15         buf[a][k * rob_x.cols + l]++;
16       }
17     }
18   }
19
20     //Se enkrat ponovimo enako kodo kot pri prejsnji zanki, le da vzamemo za
         meje zank od k = i - (a + (low+1)) do k < i + 1 in od l = j - (a + (
         low+1)) do l < j + 1.
21 }
22
23 //Podobno kot prejsnji primer, le da tukaj vzamemo drugi dve cetrtini
       kroznice.
24 else if ( (vrednostX >= 0 && vrednostX <= 5 && vrednostY >= 250) || (
       vrednostX >= 250 && vrednostY >= 0 && vrednostY <= 5) )
25 {
26     //Koda je tukaj enaka kot pri prejsnjem pogoju, le da prvic vzamemo za
         meje zank od k = i - 1 do k < i + (a + (low+1)) in od l = j - (a + (
         low+1)) do l < j + 1, drugic pa od k = i - (a + (low+1)) do k < i + 1
          in od l = j - 1 do l < j + (a + (low+1)).
27 }
28
29 //Varnostna zanka - ce slucajno usmerjenost robne tocke ne ustreza prejsnjim
       primerom (morala bi), potem vzamemo celotno kroznico.
30 else
31 {
32     //Koda je tukaj enaka kot pri prejsnjih pogojih, le da vzamemo za meje
         zank od i - (a + (low+1)) do i + (a + (low+1)) tako za spremenljivko
         k kot za l.
33 }
```

Slika 3.4: Tretji glavni postopek programa za zaznavanje prometnih znakov za omejitev je zaznavanje krogov s Houghovim transformom. Več pogojev je zato, ker gre za optimiziran Houghov transform, kjer upoštevamo tudi usmeritev robov - ta optimizacija je prikazana na sliki 2.6b.

s prvima dvema pogojema. Zanka je varnostna zato, ker naj bi s prej opisanima pogojema obravnavali vse primere, zato v teoriji ne bi smel program priti v to zanko. Če pa je slučajno kak tak primer, ki ni bil zajet, povečamo polje `buf` za točke celotne krožnice s središčem v robni točki (i, j).

3.1.4 Optimizacije

Program, kot je bil opisan v prejšnjih odsekih, bi že lahko deloval. Pri testiranju pa se je izkazalo, da ima še nekaj težav, ki jih je možno odpraviti z dodatnimi optimizacijami. Največja težava je bila, da je program ne glede na spreminjanje konfiguracijskih vrednosti včasih razen samih prometnih znakov kot znak zaznal tudi bližje rdeče predmete nepravilnih oblik. Primer takšnega napačnega zaznavanja je bila rdeča majica, ki je z nepravilnim površjem tvorila sence in program je na njej tako napačno zaznal krog.

Prometni znak za omejitev pa ima lastnost, ki jo je bilo mogoče izkoristiti za optimizacijo delovanja programa. Sestavljen je iz dveh delov: notranjega, kjer se nahaja številčna omejitev in je bodisi rumene ali pa bele barve (oziroma vmesnih odtenkov - odvisno od ohranjenosti znaka), in zunanjega, kjer se nahaja rob rdeče barve. Kar posledično pomeni, da lahko zaznamo dva kroga - enega, ki razmejuje notranji del znaka od rdečega roba, ter drugega, ki razmejuje rdeči rob od okolice. Rob znaka je namreč dovolj velik, da omogoča zaznavanje dveh različno velikih krogov s središčem v isti točki.

Logičen zaključek tega dejstva je torej, da postavimo dodaten pogoj za zaznavanje prometnega znaka za omejitev - če smo zaznali najmanj dva kroga s središčem v isti točki, je to znak. S tem se v veliki meri znebimo prej omenjenega problema, da kot znak zaznamo tudi druge objekte, saj se zelo redko zgodi, da bi v kakem drugem objektu zaznali več krogov z istim središčem. To je bil tudi glavni razlog, da ni bil uporabljen vgrajen ukaz knjižnice OpenCV za Houghov transform, saj ta ne omogoča iskanje koncentričnih krogov na sliki.

Z uvedbo te optimizacije pa se pojavi nov problem. Pogosto se namreč zgodi, da pri prometnem znaku za omejitev zaznamo prvi krog s središčem v eni točki, drugi krog pa ima središče zamaknjeno za par pikslov. Gre za zelo majhno razliko, vendar dovolj veliko, da ne moremo uporabiti prej omenjene omejitve. Poleg tega nepravilni krogi (recimo elipse) proizvedejo večje število najdenih središč in polmerov kroga, kar je sicer tudi posledica vzorčenja slik. Pred tem je torej potrebno združiti sosednje piksle v skupine pikslov, tako da lahko zaznane kroge razvrstimo v skupine glede na medsebojno oddaljenost njihovih središč.

V našem primeru združujemo bloke slike v velikosti 10×10 pikslov, kot je predstavljeno na sliki 3.5. Najprej ustvarimo in inicializiramo polje `krogi` velikosti `rob_x.rows / korak * rob_x.cols / korak` - vsako razsežnost slike torej delimo z vrednostjo `korak`, ki je enaka 10. Nato se s potrebnimi zankami sprehodimo skozi celotno polje `krogi`, v vsakem koraku pa se z dvema dodatnima zankama sprehodimo po 10×10 pikslov velikem bloku slike, kjer levi zgornji piksel ustreza mestu v polju `krogi`, kamor bomo shranjevali podatke. Podobno kot prej tudi tukaj ne upoštevamo vseh pikslov, ampak zaradi hitrosti preskakujemo v iskanih polmerih za 4, v vrsticah pa za 2.

V pogoju preverimo, ali je posamezen piksel v sliki dobil zadostno število glasov, da ga lahko upoštevamo kot središče kroga. Glasove za piksle zbiramo z optimiziranim Houghovim transformom, opisanem v odseku 3.1.3. Če ta pogoj drži, potem na mesto v polju `krogi`, ki pripada ustreznemu 10×10 bloku slike, shranimo potrebne podatke. Vsako mesto v polju `krogi` sestoji iz štirih vrednosti: x in y koordinat središča kroga, polmera kroga in števila zaznanih krogov. Vsaki izmed teh štirih vrednosti prištejemo vrednosti kroga s središčem v pikslu, na katerem se trenutno nahajamo.

Ko imamo enkrat zaznane znake združene v 10×10 pikslov velike bloke, se sprehodimo po polju `krogi`, kjer imamo shranjene te bloke. Če sta bila v enem od blokov zaznana vsaj dva kroga, to pomeni, da smo zaznali prometni znak za omejitev. V tem primeru izberemo blok slike z največjim številom zaznanih krogov, kar preverimo s spremenljivko `max`, v katero smo predhodno shranili največje število zaznanih krogov po blokih. Blok z največjim številom zaznanih krogov izberemo zato, ker se v redkih primerih zgodi, da meja dveh sosednjih 10×10 pikslov velikih blokov poteka ravno skozi središče iskanih krogov, kar pomeni, da se nekateri zaznani krogi razvrstijo v enega od teh blokov, nekateri pa v drugega in včasih s tem povzročijo dva zaznana prometna znaka za omejitev. Oba sta sicer pravilno zaznana, upoštevamo pa tistega z več zaznanimi krogi.

Za posamezen blok slike imamo v polju `krogi` shranjene seštevke x in y koordinat ter polmerov, kakor tudi število najdenih krogov. Ko enkrat najdemo blok z največ zaznanimi krogi, najprej izračunamo povprečje x in y koordinat ter polmerov vseh najdenih krogov v tem bloku. Z računanjem povprečja namreč dosežemo boljši približek središču prometnega znaka za omejitev in ga lahko posledično tudi boljše prikažemo.

Za tem sledi še shranjevanje podatkov, potrebnih za prikaz znaka. Znak prikažemo v ločenem oknu, zato potrebujemo podatke za lokacijo in velikost tega kvadrata na izvorni sliki. V spremenljivki x in y si shranimo x in y koordinati leve zgornje točke kvadrata, kar izračunamo kot $x_s - r$ oziroma $y_s - r$, pri čemer x_s in y_s predstavljata koordinati središča prej izračunenega povprečja krogov, r pa povprečni polmer. Dolžino stranic kvadrata izračunamo kot $2r$. Pri x in y koordinatah odštejemo število 10, pri dolžini stranice pa prištejemo 25. Razlog za to je, da potem pri prikazovanju prometnega znaka za omejitev nimamo robov znaka tik ob robovih okna, ampak nekoliko oddaljene od njih, kar omogoča lepši pregled nad znakom.

Na samem koncu programa se nahaja še koda za prikaz zaznanih prometnih znakov, ki zaradi enostavnosti ni vključena v sliko 3.5. V primeru, da smo zaznali prometni znak za omejitev, ga na izvorni sliki obdamo z zelenim pravokotnikom, da je lažje viden, in shranimo v datoteko. Nato iz shranjenih datotek naložimo in prikažemo zadnja dva zaznana prometna znaka za omejitev.

```
1  //Sliko zdruzimo v 10x10 bloke.
2  for (a = 0; a < range; a+=4) {
3    for (i = 0; i < rob_x.rows / korak; i++) {
4      for (j = 0; j < rob_x.cols / korak; j++) {
5        for (k = korak * i; k < korak * i + korak; k+=2) {
6          for (l = korak * j; l < korak * j + korak; l++) {
7            //Ce je posamezen piksel na sliki prejel zadostno stevilo glasov,
                 pomeni, da smo nasli sredisce kroga.
8            if (buf[a][k * rob_x.cols + l] >= meja) {
9              //x koordinata.
10             krogi[i*rob_x.cols/korak + j][0] = krogi[i*rob_x.cols/korak + j
                   ][0] + l;
11             //Podobno se za y koordinato in polmer.
12             //Stevilo zaznanih krogov.
13             krogi[i*rob_x.cols/korak + j][3]++;
14           }
15         }
16       }
17     }
18   }
19 }
20
21 for (i = 0; i < rob_x.rows / korak; i++) {
22   for (j = 0; j < rob_x.cols / korak; j++) {
23     //Ko najdemo 10x10 blok z najvec zaznanimi krogi (najmanj dva), vzamemo
           povprecje x in y koordinat ter polmera.
24     if (krogi[i*rob_x.cols/korak + j][3] >= 2 && krogi[i*rob_x.cols/korak + j
           ][3] == max) {
25           //x koordinata.
26       krogi[i*rob_x.cols/korak + j][0] = krogi[i*rob_x.cols/korak + j][0] /
             krogi[i*rob_x.cols/korak + j][3];
27           //Podobno se za y koordinato in polmer.
28
29       //Shranimo podatke za prikaz znaka.
30       x = krogi[i*rob_x.cols/korak + j][0] - krogi[i*rob_x.cols/korak + j][2]
             - 10;
31       y = krogi[i*rob_x.cols/korak + j][1] - krogi[i*rob_x.cols/korak + j][2]
             - 10;
32       dolzina = 2 * krogi[i*rob_x.cols/korak + j][2] + 25;
33       radius = krogi[i*rob_x.cols/korak + j][2];
34     }
35   }
36 }
```

Slika 3.5: Koda, ki prikazuje združevanje zaznanih krogov z optimiziranim Houghovim transformom v 10×10 pikslov velike bloke. To združevanje omogoča boljše zaznavanje prometnih znakov za omejitev, saj so včasih središča zaznanih krogov med seboj oddaljena za par pikslov.

3.2 Testiranje in določanje mejnih vrednosti

Konfiguracijsko datoteko programa za zaznavanje prometnih znakov za omejitev sestavlja šest vrednosti, ki si po vrsti sledijo:

1. Spodnja meja nasičenosti (saturation),
2. Zgornja meja odtenka (hue),
3. Spodnja meja odtenka,
4. Razpon iskanih polmerov krogov,
5. Najmanjši iskan polmer kroga,
6. Spodnja meja glasov za zaznan krog.

Prve tri vrednosti potrebujemo za barvno segmentacijo v barvnem prostoru HSV, zadnje tri pa za zaznavanje krogov s Houghovim transformom. Opis, kako smo te algoritme uporabili, se nahaja v odseku 3.1 - v tem bodo opisani postopki, kako smo prišli do naštetih mejnih vrednosti.

Za testiranje in določanje mejnih vrednosti za barvno segmentacijo je bilo potrebno dobiti čimvečjo bazo slik prometnih znakov za omejitev v čimveč svetlobnih pogojih. Zato smo se lotili zbiranja testnih slik tako, da smo s kamero skozi celoten dan vsakih 5 minut zajeli sliko prometnega znaka za omejitev in na ta način prišli do 198 testnih slik. Tako smo torej dobili slike prometnega znaka za omejitev v jutranjem, dopoldanskem, popoldanskem in večernem času, poleg tega pa se je prometni znak nahajal pod ulično lučjo, kar lahko v večernih urah tudi vpliva na drugačno obarvanost znaka. S tem smo zajeli večino svetlobnih pogojev in s tem ustvarili primerno učno bazo za določanje mejnih vrednosti. Edina pogoja, ki ju nismo zajeli, sta prometni znak za omejitev v oblačnem vremenu in slabo ohranjen prometni znak. Na sliki 3.6 se nahaja vzorec 20 testnih slik, na katerih so vidni vplivi različnih svetlobnih pogojev na obarvanost znaka.

Testiranje je potekalo z nekoliko spremenjeno verzijo programa za zaznavanje prometnih znakov. V končnem programu namreč dobivamo slike iz kamere v realnem času, program za testiranje pa je v vsem enak, le da nalaga testne slike in med njimi se premikamo s pritiskom na katerokoli tipko. Poleg tega pri vsaki sliki izpisujemo število zgrešenih slik oziroma število zgrešenih prometnih znakov za omejitev, saj se znaki nahajajo na vseh testnih slikah, in čas, potreben za procesiranje vsake slike.

V tabelah 3.1, 3.2 in 3.3 se nahajajo vrednosti, ki so bile uporabljene za določanje mejnih vrednosti za zgornjo in spodnjo mejo odtenka ter spodnjo mejo nasičenosti, v vsaki tabeli pa je tudi število zgrešenih testnih slik (vseh je 198). Postopek testiranja je bil sledeč: najprej smo zelo omejili mejne vrednosti - odtenek smo iskali med vrednostima 120 in 130, nasičenost pa od 130 do 255. Pri tem so vrednosti odtenka barve podane v stopinjah, kjer celoten krog znaša 180°, ker je v knjižnici OpenCV predstavljen z 8 biti, zaradi česar bi bila predstavitev 360° nemogoča. Potem smo območja iskanja postopoma večali - najprej smo

Slika 3.6: Vzorec 20 od 198 testnih slik za določanje mejnih vrednosti barvne segmentacije v barvnem prostoru HSV. Slike so bile zajete vsakih 5 minut skozi celoten dan, prikazana pa je vsaka 10-ta slika. Na slikah so vidni različni svetlobni pogoji in posledično različno obarvan prometni znak za omejitev.

povečevali zgornjo mejo odtenka, nato nižali spodnjo mejo odtenka ter na koncu nižali še spodnjo mejo nasičenosti. Ko smo enkrat prišli do vrednosti, ki ni več prinašala manjšega števila zgrešenih testnih slik, smo kot končno mejno vrednost vzeli tisto vrednost, ki ima enako število zgrešenih slik kot zadnja testirana in hkrati najbolj omejuje območje iskanja. Na ta način smo karseda omejili iskana območja in s tem zaznani šum, po drugi strani pa smo zaznali skoraj vse slike prometnega znaka za omejitev.

Tabela 3.1: Vrednosti za določanje zgornje meje odtenka. Za spodnjo mejo odtenka in nasičenosti smo vzeli vrednosti 120 in 130, ki jih po tem dokončno nastavimo. Odebeljena vrstica predstavlja končno vrednost. Vseh testnih slik je 198.

Zgornji odtenek	Spodnji odtenek	Nasičenost	Št. zgrešenih slik
130	120	130	26
131	120	130	22
132	**120**	**130**	**10**
133	120	130	10
135	120	130	10
137	120	130	10
150	120	130	10

Za končne mejne vrednosti barvne segmentacije smo torej vzeli:

- Zgornja meja odtenka: 132,
- Spodnja meja odtenka: 106,
- Spodnja meja nasičenosti: 125.

Pri tem pa smo zgrešili le 3 od 198 testnih slik, pa še te so si sledile ena za drugo v trenutku, ko je sonce svetilo naravnost v kamero. Ti trije primeri so prikazani na sliki 3.7.

Slika 3.7: Edine 3 izmed 198 testnih slik, ki jih program ne zazna. Razlog je v zelo močni svetlobi sonca, ki sveti naravnost v kamero, zaradi česar kamera napačno zazna barve.

Tabela 3.2: Vrednosti za določanje spodnje meje odtenka. Za zgornjo mejo odtenka smo vzeli prej določeno optimalno vrednost 132, za spodnjo mejo nasičenosti pa vrednost 130, ki jo potem dokončno nastavimo. Odebeljena vrstica predstavlja končno vrednost. Vseh testnih slik je 198.

Zgornji odtenek	Spodnji odtenek	Nasičenost	Št. zgrešenih slik
132	120	130	10
132	110	130	9
132	108	130	7
132	107	130	7
132	**106**	**130**	**6**
132	105	130	6
132	104	130	6

Tabela 3.3: Vrednosti za določanje spodnje meje nasičenosti. Za zgornjo in spodnjo mejo odtenka smo vzeli prej določeni optimalni vrednosti 132 in 106. Odebeljena vrstica predstavlja končno vrednost. Vseh testnih slik je 198.

Zgornji odtenek	Spodnji odtenek	Nasičenost	Št. zgrešenih slik
132	106	130	6
132	106	128	4
132	106	127	4
132	106	126	4
132	**106**	**125**	**3**
132	106	120	3

Za ostale tri vrednosti, ki jih uporabljamo za zaznavanje krogov s Houghovim transformom, pa smo zajeli 12 testnih slik prometnih znakov za omejitev na različni oddaljenosti. Število slik je od testnih slik za segmentacijo manjše, ker ni potrebno zajeti toliko različnih pogojev. Pri segmentaciji je bilo potrebno zajeti prometni znak v vseh svetlobnih pogojih, pri zaznavanju krogov pa se spreminja samo oddaljenost prometnega znaka, zato ni potrebe po velikem številu testnih slik.

Začeli smo z vrednostjo 20 za razpon iskanih polmerov, 30 za najmanjši iskan polmer (oboje podano v pikslih) in 30 za spodnjo mejo glasov za zaznan krog. Spodnja meja glasov nam pove, kolikokrat moramo povečati akumulator pri Houghovem transformu za posamezen piksel, da ga obravnavamo kot središče kroga. Postopek testiranja se nekoliko razlikuje od testiranja vrednosti barvne segmentacije, saj so v tem primeru vrednosti bolj medsebojno odvisne. Če na primer znižamo razpon iskanih polmerov, moramo ustrezno spremeniti tudi vrednost najmanjšega iskanega polmera, kot tudi spodnjo mejo glasov, saj bolj oddaljeni prometni znaki prejmejo manj glasov kot bližnji.

Testiranje je potekalo tako, da smo postopoma večali razpon iskanih polmerov, nižali vrednost najmanjšega iskanega polmera ter manjšali spodnjo mejo glasov. Do optimalnega rezultata 3 zgrešenih slik smo prišli z vrednostmi 30 za razpon, 20 za najmanjši polmer in 15 za spodnjo mejo glasov, kot je prikazano v tabeli 3.4. Potrebno je omeniti, da so prometni znaki za omejitev na teh treh zgrešenih slikah toliko oddaljeni, da bi v primeru dodatnega nižanja kriterijev kot prometni znak zaznali tudi veliko šuma, zato smo se ustavili pri teh vrednostih.

Tabela 3.4: Vrednosti za določanje razpona iskanih polmerov, najmanjšega polmera in najmanjšega števila glasov za zaznan krog. Odebeljena števila predstavljajo končne vrednosti. Vseh testnih slik je 12.

Razpon	Najmanjši polmer	Najmanjše št. glasov	Št. zgrešenih slik
20	30	30	7
25	30	30	7
30	30	30	6
30	25	30	6
30	**20**	30	5
30	20	25	4
30	20	20	4
30	**20**	**15**	**3**

3.2.1 Rezultati

Ko smo enkrat dokončno nastavili mejne vrednosti, je bil program za zaznavanje prometnih znakov za omejitev pripravljen za končno testiranje. To je potekalo tako, da smo poskušali zajeti čimveč različnih prometnih znakov za omejitev in tako praktično preveriti delovanje programa. Merili smo število zaznanih oziroma zgrešenih prometnih znakov kot tudi hitrost delovanja programa.

Od vseh 50 zajetih prometnih znakov za omejitev smo s programom uspešno zaznali 45 prometnih znakov, kar pomeni 90% zanesljivost. Hitrosti obdelave posamezne slike so se gibale od 100 do 200 milisekund, kar znese 5 do 10 slik na sekundo, odvisno od količine zaznane rdeče barve - če je slika vsebovala veliko rdečih odtenkov, je program potreboval nekoliko več časa za obdelavo. 2 od 5 primerov z zgrešenimi prometnimi znaki je bilo posnetih, ko je sonce svetilo naravnost v kamero - takrat je zaradi avtomatskega izravnavanja nivoja beline uporabljene kamere ta napačno zaznala barve in zaradi tega zgrešila prometni znak.

Če bi bolj omejili pogoje za zaznavanje prometnih znakov za omejitev, bi program sicer deloval hitreje, ampak bi zaznali manjše število prometnih znakov. Obratno velja, če bi te pogoje omilili - v tem primeru bi program deloval počasneje, a bi zaznali več prometnih znakov za omejitev. Odločili smo se, da 90% zanesljivost skupaj s hitrostjo 5 do 10 slik na sekundo predstavlja primerno razmerje med hitrostjo in zanesljivostjo delovanja programa.

Poglavje 4

Zaključek

V poglavju 2 je bila opisana teorija zaznavanja prometnih znakov za omejitev. To obsega tri glavne postopke: barvno segmentacijo, zaznavanje robov in zaznavanje krogov. Pri barvni segmentaciji je bil poleg same segmentacije opisan tudi barvni prostor HSV, saj je zelo pomemben pri naši implementaciji barvne segmentacije. Za zaznavanje robov so bili opisani tri gradientni robni operatorji: Robertsov, Sobelov in Prewittov, kot tudi osnovna ideja zaznavanja robov, pri zaznavanju krogov pa je bil v podrobnosti opisan predvsem Houghov transform in njegove optimizacije, poleg tega pa tudi zaznavanje krogov s pari gradientnih vektorjev.

Poglavje 3 vsebuje temeljit opis in razlago delovanja kode samostojno izdelanega programa za zaznavanje prometnih znakov. Natančno so razložene implementacije barvne segmentacije v barvnem prostoru HSV, zaznavanja robov s Sobelovim operatorjem in zaznavanja krogov s Houghovim transformom, kot tudi lastne optimizacije programa. Poleg tega se tu nahajajo podrobnosti testiranja in določanja mejnih vrednosti za zaznavanje, na koncu pa še rezultati testiranja. Pri testiranju je poleg določanja mejnih vrednosti pomemben rezultat dela tudi sama zbirka slik prometnih znakov za omejitev v različnih svetlobnih pogojih.

Kot je bilo ugotovljeno v odseku 3.2.1, končni program za zaznavanje prometnih znakov za omejitev deluje dobro, saj je pri testiranju zaznal 90% prometnih znakov v realnem času s hitrostjo 5 do 10 slik na sekundo. Vseeno pa je prostora za izboljšave še veliko.

Najprej tu velja omeniti optimizacijo, ki je opisana v odseku 3.1.4, pri kateri s Houghovim transformom združujemo zaznane kroge v 10×10 pikslov velike bloke slike. Ta postopek je bil uveden zaradi lastnosti prometnega znaka za omejitev, da je zunanji krog rdeče barve sestavljen iz dveh robov - notranjega in zunanjega. Zato lahko uvedemo pogoj za zaznavanje prometnih znakov, da moramo zaznati najmanj dva kroga z istim središčem. Včasih pa so ta središča krogov med seboj zamaknjena za par pikslov in to je bila motivacija za uvedbo združevanja slike v bloke, saj s tem v veliki meri pridobimo na zanesljivosti zaznavanja. Ima pa ta optimizacija vseeno pomanjkljivost, in sicer v primeru, ko meja med dvema blokoma poteka ravno tudi med dvema zaznanima središčima. Takrat bo eno središče v enem bloku in drugo v drugem, kar pomeni, da bomo prometni znak za omejitev zgrešili. Tak primer je zelo redek, vendar bi bilo ob nadaljnjem razvoju programa primerno

poskrbeti tudi zanj.

Druga izboljšava programa bi bila večja učna baza prometnih znakov za omejitev. Za določanje mejnih vrednosti barvne segmentacije smo sicer zajeli 198 slik prometnega znaka v vseh trenutkih dneva, vendar smo s tem spustili nekaj različnih pogojev, ki vplivajo na različno obarvanost znaka, kot sta oblačno vreme in slaba ohranjenost znaka. Ena od testnih voženj je sicer potekala v oblačnem vremenu, so pa bili vsi prometni znaki za omejitev na testiranem območju dobro ohranjeni. Za določanje vseh šestih mejnih vrednosti bi bilo idealno pridobiti čimveč slik različnih prometnih znakov s področja celotne Slovenije v različnih vremenskih pogojih, ki bi bili tudi različno ohranjeni, in glede na njih nastaviti mejne vrednosti. S tem bi dosegli še večjo hitrost in zanesljivost delovanja.

Zadnja in morda največja omejitev razvoja programa je bila uporabljena kamera. Gre za spletno kamero Logitech HD Webcam C510, ki je bila izbrana zaradi svoje prilagodljivosti ter predvsem zaradi možnosti pošiljanja slike na računalnik v realnem času, zaradi česar niso prišle v poštev ročne kamere. Glede na ceno je tudi kakovost zajemanja videa zadovoljiva, vseeno pa bi veliko boljše rezultate dobili z industrijsko kamero.

Naslednji logični korak v razvoju programa bi bila implementacija razpoznavanja prometnih znakov za omejitev, kjer ne bi prometni znak za omejitev le zaznali, ampak tudi razpoznali številčno omejitev znotraj znaka. S tem bi lahko vozniku prikazali vrednost omejitve in ga ob prehitri vožnji morda tudi opozorili. Za tem bi lahko zaznavanje in razpoznavanje razširili še na ostale prometne znake in bi s tem imeli celovit program za zaznavanje in razpoznavanje prometnih znakov. Ker pa obseg te zaključne naloge zajema le zaznavanje prometnih znakov za omejitev, smo se na tem mestu ustavili.

Literatura

[1] C. Bahlmann, Y. Zhu, V. Ramesh: *A System For Traffic Sign Detection, Tracking, Recognition Using Color, Shape, And Motion Information*. Siemens Corporate Research, Inc., 2005.

[2] D.H. Ballard, C.M. Brown: *Computer Vision*. University of Rochester, 1982.

[3] R. Jain, R. Kasturi, B.G. Schunck: *Machine Vision*. McGraw-Hill, 1995.

[4] A. Koschan: *A Comparative Study On Color Edge Detection*. Technical University Berlin, 1995.

[5] J.R. Parker: *Algorithms For Image Processing And Computer Vision*. Wiley Publishing, Inc., 1997.

[6] A.A. Rad, K. Faez, N. Qaragozlou: *Fast Circle Detection Using Gradient Pair Vectors*. Amirkabir University of Technology, 2003.

[7] H. Rhody: *Hough Circle Transform*. Rochester Institute of Technology, 2005.

[8] M. Shah: *Fundamentals Of Computer Vision*. University of Central Florida, 1997.

[9] J. Shi, J. Malik: *Normalized Cuts And Image Segmentation*. Carnegie Mellon University, 2000.

[10] S. Sural, G. Qian, S. Pramanik: *Segmentation And Histogram Generation Using The HSV Color Space For Image Retrieval*. Michigan State University, 2002.

[11] R. Szeliski: *Computer Vision: Algorithms And Applications*. Springer, 2010.

[12] H.K. Yuen, J. Princen, J. Illingworth, J. Kittler: *A Comparative Study On Hough Transform Methods For Circle Finding*. University of Surrey, 1990.

Viri

[13] Southwest Research Institute: *Machine Vision Control Of Laser Weld Processing*. Pridobljeno avgusta 2011 iz: `http://www.swri.org/4org/d10/msd/automation/images/AWACS_system.jpg`.

[14] USC iLab: *HSV And H2SV Color Space*. Pridobljeno avgusta 2011 iz: `http://ilab.usc.edu/wiki/images/5/58/HSV_Color_Cone.png`.

[15] R. Wang: *Detection Of Circles*. Pridobljeno avgusta 2011 iz: `http://fourier.eng.hmc.edu/e161/lectures/hough_4.gif`.

Priloge

Zaključni nalogi je priložen CD, ki vsebuje zaključno nalogo v elektronski obliki (datoteka `zakljucna_naloga.pdf`), seznam uporabljene opreme (datoteka `oprema.txt`), izvorno kodo samostojno izdelanega programa za zaznavanje prometnih znakov za omejitev in verzije programa za testiranje (datoteki `prometni_znaki.cpp` in `prometni_znaki_test.cpp`), konfiguracijsko datoteko programa (datoteka `vrednosti.txt`), mapo za shranjevanje zaznanih prometnih znakov za omejitev (mapa `zaznava`), testne slike prometnih znakov za omejitev (slike v mapah `test_seg` in `test_krog`), ki so bile uporabljene za določanje mejnih vrednosti, ter predstavitveni video (datoteka `predstavitveni_video.wmv`), ki prikazuje praktično delovanje programa za zaznavanje prometnih znakov za omejitev.

Slike

Tabele

Stvarno kazalo