SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD BR. 1392

SEGMENTACIJA SLIKE NA TEMELJU POKRETA

Goran Adrinek

Zagreb, srpanj 2003.

Zahvale

Prof. dr. Slobodanu Ribariću, na mentorstvu, uvođenju u, za mene, potpuno novi svijet i poticanju da dam sve od sebe, odnosno da pomaknem granice svojih mogućnosti.

Mr. Siniši Šegviću, na upoznavanju sa sklopovskom i programskom opremom nužnom za ovaj rad, pomoći, idejama i savjetima.

Dr. Zoranu Kalafatiću, na savjetima, pomoći i, najviše od svega, podršci i ohrabrivanju.

Sadržaj

Zahvale ..... **i**
Sadržaj ..... **iii**
Popis tablica i slika ..... **vii**
1 Uvod **1**
1.1 Analiza dinamičkih scena 1
1.2 Zadatak rada 2
1.3 Struktura rada 3
2 Pregled radova s područja segmentacije dinamičkih scena **5**
2.1 Postupci temeljeni na regijama 5
2.2 Postupci temeljeni na značajkama 6
2.3 Postupci temeljeni na aktivnim konturama 7
2.4 Postupci temeljeni na 3D modelima 7
3 Detekcija promjene **9**
3.1 Diferencija slika 9
3.1.1 Filter veličine 10
3.1.2 Robusna detekcija promjene [1] 11
3.2 Akumulirana diferencija slika 12
4 Segmentacija upotrebom pokreta **15**
4.1 Korištenje akumulirane diferencije slika 15
4.2 Detekcija vremenski promjenjivih rubova 15
4.3 Označena diferencija slika 16
4.4 Oduzimanje pozadine 18
5 Aktivni vid **21**
5.1 Pozornost 21
5.2 Fovealni vid 22
5.3 Upravljanje pogledom 23
5.4 Koordinacija oko-ruka 23
6 Sustav za aktivno praćenje pokretnih objekata - SAPPO **25**
6.1 Opis sustava 25
6.2 Prva faza: određivanje pomaka kamere 26
6.2.1 Detekcija točaka (značajki) za uparivanje 27

6.2.2 Uparivanje značajki 30
6.2.3 Pronalaženje dominantnog gibanja 31
6.3 Druga faza: detekcija pokreta i lociranje pokretnog objekta 32
6.4 Treća faza: usmjeravanje kamere prema pokretnom objektu 34
6.5 Opis implementacije sustava 35
6.6 Eksperimenti 36
6.6.1 Utjecaj parametra *t* 36
6.6.2 Utjecaj parametra *r* 39
6.6.3 Utjecaj parametra *d* 41
6.6.4 Utjecaj parametra τ 43
6.6.5 Utjecaj parametra *N* 43
6.6.6 Praćenje 44
6.7 Rezultati 48
7 Sustav za izlučivanje slike vozila iz scene - SIVS 49
7.1 Opis sustava 49
7.2 Prva faza: segmentacija i praćenje vozila 50
7.2.1 Detekcija promjene i segmentacija 50
7.2.2 Praćenje 54
7.3 Druga faza: izlučivanje slike vozila 55
7.3.1 Korištenje stacionarne kamere 56
7.3.2 Korištenje upravljive kamere 56
7.3.3 Korištenje dviju kamera 57
7.4 Opis implementacije sustava 58
7.5 Eksperimenti 59
7.5.1 Utjecaj parametra *m* 59
7.5.2 Utjecaj parametra *s* 61
7.5.3 Utjecaj parametra τ 62
7.5.4 Utjecaj parametra *N* 63
7.5.5 Utjecaj parametra *q* 64
7.5.6 Rješavanje zaklanjanja 64
7.5.7 Cjelokupni rad sustava 66
7.6 Rezultati 69
8 Zaključak 71
Literatura 73
Prilog A: Opis programske implementacije 75
Sustav za aktivno praćenje pokretnih objekata - SAPPO 75

Korištene klase i funkcije ... 75
Opis korištenih klasa i funkcija ... 76
Način rada ... 87
Verzije ... 89
Platforme ... 89
Sustav za izlučivanje slike vozila iz scene - SIVS ... 89
Korištene klase i funkcije ... 89
Opis korištenih klasa i funkcija ... 90
Konfiguracije ... 112
Način rada ... 112
Platforme ... 113
Prilog B: Upute za korištenje ... 115
Sustav za aktivno praćenje pokretnih objekata - SAPPO ... 115
Instalacija ... 115
Spajanje opreme ... 115
Upute za korištenje ... 116
Parametri postupka ... 117
Sustav za izlučivanje slike vozila iz scene - SIVS ... 117
Instalacija ... 117
Spajanje opreme ... 118
Upute za korištenje ... 118
Parametri postupka ... 120
Ljuska CVSH ... 120
Poziv programa ... 120
Formati ulaznih datoteka ... 122
Interaktivne naredbe ljuske ... 122
Prilog C: Korišteni radovi ... 125

Popis tablica i slika

Tablice

Tablica 6.1: Sklopovska i programska oprema korištena u implementaciji SAPPO-a. 35
Tablica 6.2: Broj neuspjeha za pojedine vrijednosti parametra *t*. 37
Tablica 6.3: Broj okvira s uspješno lociranim objektom u ovisnosti o parametru *r*. 40
Tablica 6.4: Broj okvira s uspješno lociranim objektom u ovisnosti o parametru *d*.41
Tablica 7.1: Sklopovska i programska oprema korištena u implementaciji SIVS-a. 58
Tablica 7.2: Ovisnost brzine detekcije zaustavljanja o parametru *s*. 61

Slike

Slika 3.1: Diferencija slika na primjeru umjetno generiranih okvira. 10
Slika 3.2: Diferencija slika na primjeru iz stvarnog svijeta. 10
Slika 3.3: Primjena filtera veličine na diferenciju slika. 10
Slika 3.4: Kreiranje područja za usporedbu. 11
Slika 3.5: Shematski prikaz akumulirane diferencije slika. 12
Slika 4.1: Detekcija vremenski promjenjivih rubova. 16
Slika 4.2: Princip označene diferencije slika. 17
Slika 4.3: Segmentacija korištenjem označene diferencije slika. 17
Slika 4.4: Principjelna shema postupaka oduzimanja pozadine. 18
Slika 4.5: Oduzimanje pozadine pomoću diferencije slika. 18
Slika 5.1: Ovisnost gustoće fotoosjetljivih stanica o udaljenosti od fovee u ljudskom oku. 22
Slika 6.1: Komponente sustava za praćenje pokretnih objekata..... 25
Slika 6.2: Područje preklapanja dvaju okvira. 26
Slika 6.3: SUSAN princip. 27
Slika 6.4: USAN područja. 28
Slika 6.5: Diskretna aproksimacija kružne maske korištene u implementaciji SUSAN detektora uglova. 28
Slika 6.6: Ovisnost broja značajki detektiranih SUSAN detektorom uglova o pragu *t*. 30
Slika 6.7: Primjer pronađenih vektora pomaka značajki. 31
Slika 6.8: Principjelna shema grupiranja na temelju teorije grafova. 33
Slika 6.9: Primjer detekcije i određivanja pozicije pokretnog objekta..... 33

Slika 6.10: Primjer ovisnosti broja uglova detektiranih SUSAN detektorom uglova o pragu sivih vrijednosti *t*. 37

Slika 6.11: Primjer označenih diferencija slika dvaju različitih okvira za različite vrijednosti parametra *t*. 39

Slika 6.12: Primjer ovisnosti lociranja objekta o parametru *r*. 40

Slika 6.13: Uspješnost grupiranja u ovisnosti o parametru *d* na primjeru dva okvira. 42

Slika 6.14: Izgled označene diferencije slika za različite vrijednosti praga τ..... 43

Slika 6.15: Izgled označene diferencije slika za različite vrijednosti praga *N*..... 44

Slika 6.16: Rezultati praćenja čovjeka u hodu. 46

Slika 6.17: Rezultati praćenja čovjeka u trku..... 48

Slika 7.1: Promjene intenziteta slikovnog elementa za uobičajene događaje. 51

Slika 7.2: Shematski prikaz praćenja spajanja i razdvajanja regija objekata. 55

Slika 7.3: Konfiguracija SIVS-a sa stacionarnom kamerom. 56

Slika 7.4: Konfiguracija SIVS-a sa upravljivom kamerom. 57

Slika 7.5: Konfiguracija SIVS-a sa dvije kamere. 58

Slika 7.6: Izgled izdvojenih regija za različite vrijednosti parametra *m* na primjeru jednog okvira ispitne sekvence. 60

Slika 7.7: Primjer prevelike osjetljivosti sustava na šum za male vrijednosti parametra *m*. 60

Slika 7.8: Primjeri izdvojenih regija za različite vrijednosti parametra *s*. 61

Slika 7.9: Primjeri utjecaja parametra τ na izdvajanje regija. 63

Slika 7.10: Utjecaj parametra *N* na eliminaciju šuma. 63

Slika 7.11: Utjecaj parametra *N* na eliminaciju presitnih objekata. 64

Slika 7.12: Primjer utjecaja parametra *q* na izdvajanje regija. 64

Slika 7.13: Ilustracija rješavanja zaklanjanja. 66

Slika 7.14: Primjer rada SIVS-a za sekvencu iz stvarnog svijeta..... 68

Slika 7.15: Primjer rada konfiguracije SIVS-a koja koristi upravljivu kameru..... 69

1

Uvod

U ovom poglavlju, nakon kratkog uvoda u problematiku - analizu dinamičkih scena, dan je zadatak rada i kratak pregled sadržaja rada.

1.1 Analiza dinamičkih scena

Svijet koji nas okružuje sastoji se od mnoštva objekata – predmeta i bića. Položaj i oblik tih objekata s vremenom se mijenja, kao i vremenske i svjetlosne prilike, tako da smo suočeni s promjenjivim svijetom. Ljudi, kao i mnoga druga bića doživljavaju svijet najviše kroz osjetilo vida, informacije dobivene vidom koriste za snalaženje u takvom svijetu – obavljanje svakodnevnih zadataka, preživljavanje.

Tehnološki razvoj omogućio je stvaranje tehničkih sustava koji oponašaju vid u biološkim sustavima. Tim sustavima bavi se računalni (strojni) vid (engl. *computer vision, machine vision*). U samom početku računalni vid bavio se samo analizom statičnih slika, uzetih iz svijeta u jednom trenutku. Daljni tehnološki razvoj, otvorio je mogućnost bavljenja promjenjivim svijetom, stvaranje sustava analizu scena dobivenih iz promjenjivog svijeta – sustava za analizu dinamičkih scena (engl. *dynamic scene analysis*). Analiza dinamičkih scena predstavlja korak bliže biološkim sustavima za vid, jer biološki sustavi kontinuirano dobivaju i analiziraju sliku okoline.

Ulaz u sustav za analizu dinamičkih scena je niz okvira (engl. *frame*), sekvenca. Pojedini okvir predstavlja sliku svijeta (odnosno dijela svijeta koji promatramo – scene) u određenom vremenskom trenutku. Sekvencu okvira možemo predstaviti funkcijom $F(x,y,t)$, gdje su x i y prostorne koordinate slikovnog elementa (engl. *pixel*) u okviru uzetom u trenutku t. U slučaju kad je vremenski razmak između okvira jednak, sekvenca se može predstaviti funkcijom $F(x,y,k)$, gdje k predstavlja redni broj okvira u sekvenci.

Vidljive promjene u sceni uzrokuju razlike između pojedinih okvira sekvence. Te promjene mogu biti posljedica kretanja promatrača (kamere), kretanja objekata scene, promjene (strukture, veličine, oblika) objekta, promjene osvjetljenja scene.

S obzirom na kretanje kamere i objekata razlikujemo četiri slučaja dinamičkih scena [1]:

1. Stacionarna kamera, stacionarni objekti (engl. *Stationary Camera, Stationary Objects - SCSO*)
2. Stacionarna kamera, pokretni objekti (engl. *Stationary Camera, Moving Objects - SCMO*)
3. Pokretna kamera, stacionarni objekti (engl. *Moving Camera, Stationary objects - MCSO*)
4. Pokretna kamera, pokretni objekti (engl. *Moving Camera, Moving Objects - MCMO*)

Prvi slučaj, analiza statičkih scena, najstarije je i najrazvijenije područje računalnog vida s mnoštvom razvijenih tehnika. Četvrti, najopćenitiji i najsloženiji slučaj najmanje je razvijeno područje računalnog vida.

Analiza dinamičkih scena metodama analize statičkih scena koje promatraju svaki okvir zasebno, nije praktična (previše je računski zahtjevna), a i ne uzima u obzir informacije sadržane u sekvenci okvira. Iz tog razloga potrebno je dinamičke scene analizirati drugim metodama, metodama koje uzimaju u obzir informacije dostupne iz sekvence. Često su metode analize dinamičkih scena jednostavnije i brže od metoda analize statičkih scena, jer uzimaju u obzir informacije iz više okvira. U prilog tome govori činjenica da i sami lakše zamjećujemo neki objekt ukoliko se on kreće.

Analiza dinamičkih scena sastoji se od tri faze [1][2][3]:

1. Periferna (engl. *peripheral*) faza - identificira područja vidnog polja koja pokazuju promjene od okvira do okvira (aktivna područja).
2. Pozorna (engl. *attentive*) faza - usmjeruje analizu na aktivna područja iz kojih izvlači informacije koje mogu poslužiti za raspoznavanje objekata, analizu pokreta i događaja u sceni.
3. Kognitivna (engl. *cognitive*) faza - primjenjuje znanje o objektima, pokretima i drugim konceptima koji ovise o području primjene, kako bi analizirala scenu s obzirom na prisutnost objekata i odvijanje događaja.

1.2 Zadatak rada

Ovaj rad obuhvaća dva područja računalnog vida - analizu dinamičkih scena i aktivni vid.

Zadatak rada je istraživanje i implementacija postupaka za detekciju promjena i segmentacije slike iz sekvence dinamičke scene te korištenje elemenata aktivnog vida u svrhu praćenja objekata u sceni i omogućavanja bolje detekcije i raspoznavanja značajki objekta.

Upotreba elemenata aktivnog vida (usmjeravanje kamere) za praćenje objekta u sceni vodi na slučaj MCMO (pokretna kamera, pokretni objekti) analize dinamičkih scena. Stoga ovaj rad implementira i postupke segmentacije vezane uz taj slučaj dinamičkih scena.

U ovom radu poseban naglasak je na prvoj, perifernoj fazi, analize dinamičkih scena. Svi postupci detekcije promjene i segmentacije slike temelje se na pojedinim slikovnim elementima.

1.3 Struktura rada

Rad se može podijeliti u dvije glavne cjeline.

Prva cjelina, koju čine poglavlja 2, 3, 4 i 5, opisuje neke od postupaka analize dinamičkih scena i daje pregled područja aktivnog vida.

Poglavlje 2 daje uvid u istraživanja vezana uz zadatak ovog rada. Daje se pregled pristupa segmentaciji dinamičkih scena te se ukratko opisuju radovi koji ilustriraju pojedini pristup.

U poglavljima 3 i 4 opisuju se postupci detekcije promjene i segmentacije slike na temelju detektiranih promjena. Opisani su postupci temeljeni na pojedinim slikovnim elementima.

Poglavlje 5 daje kratak uvod u područje aktivnog vida te opisuje pojedine važnije elemente aktivnog vida.

Druga cjelina, poglavlja 6 i 7, opisuje dva sustava koji su razvijeni u sklopu ovog rada.

Poglavlje 6 opisuje sustav za aktivno praćenje pokretnih objekata. Detaljno je opisan način rada sustava, postupci korišteni u realizaciji te implementacija. Prikazani su eksperimenti koji prikazuju ovisnost rada sustava o pojedinim parametrima i ilustriraju rad sustava.

Poglavlje 7 opisuje drugi sustav, sustav za izlučivanje slike vozila iz scene. Slično kao i u poglavlju 6, opisan je princip rada sustava, korišteni postupci i implementacija. Također su prikazani i eksperimenti koji ilustriraju rad sustava i pokazuju utjecaj pojedinih parametara na rad sustava.

2

Pregled radova s područja segmentacije dinamičkih scena

Dosad je razvijeno mnoštvo različitih postupaka analize dinamičkih scena. Po principu segmentacije, većina postupaka može se svrstati u četiri kategorije pristupa:

1. Postupci temeljeni na regijama (engl. *region based*).
2. Postupci temeljeni na značajkama (engl. *feature based*).
3. Postupci temeljeni na aktivnim konturama (engl. *active contour based*).
4. Postupci temeljeni na 3D modelima (engl. *3D model based*).

Osim na ovaj način, postupci se mogu još podijeliti i na pristupe odozgo prema dolje (engl. *top-down*) i odozdo prema gore (engl. *bottom-up*).

Pristup odozgo prema dolje podrazumijeva neko prijašnje znanje o sceni i objektima na temelju čega se traži prisustvo objekta u sceni. U pristup odozgo prema dolje spadaju postupci temeljeni na (3D) modelima.

Pristup odozdo prema gore ne zahtijeva nikakvo prijašnje znanje o sceni i objektima. Scena se particionira, vođena podacima, u regije koje odgovaraju promjenama (postupci temeljeni na regijama), kretanju (postupci temeljeni na značajkama) ili određenim cjelinama u sceni (postupci temeljeni na aktivnim konturama).

Ovdje će biti ukratko objašnjeni osnovni principi pojedinih pristupa. Uz svaki pristup navedeni su i neki radovi koji koriste taj pristup.

2.1 Postupci temeljeni na regijama

Postupci temeljeni na regijama nazivaju se još i postupci temeljeni na pojedinim slikovnim elementima (engl. *pixel based*). Glavna karakteristika ovih postupaka je detekcija razlike pojedinog slikovnog elementa trenutnog u odnosu na neke od

prošlih okvira, te grupiranje susjednih slikovnih elemenata u regije za koje se pretpostavlja da pripadaju pokretnom objektu.

Najčešće korišteni postupak segmentacije dinamičkih scena temeljen na regijama je oduzimanje pozadine (engl. *background subtraction*). Postupak se temelji na utvrđivanju razlike između trenutnog okvira i okvira koji predstavlja model pozadine. Najčešći pristupi modeliranju pozadine su korištenje Kalmanovog filtriranja (engl. *Kalman filtering*) [5] i korištenje statističkih modela [6][7][8].

U radu [9] korištena je kombinacija adaptivnog oduzimanja pozadine i diferencije slika kroz tri okvira kako bi se riješio problem pomicanja objekta koji je prethodno bio smatran pozadinskim. U istom radu predložen je postupak koji se naslanja na oduzimanje pozadine, ali uvodi i princip slojeva koji predstavljaju zaustavljene objekte u pozadini čime se može uspješno segmentirati slika i u uvjetima zaklanjanja objekata[1]. Oba postupka korištena su u sustavu za video nadzor za praćenje ljudi i vozila.

Prednost postupaka temeljenih na regijama je jednostavnost i brzina. Premda zbog svoje jednostavnosti općenito nisu dovoljni za segmentaciju i razumijevanje složenijih scena, mogu se uspješno koristiti u određenim ograničenim situacijama koje se mogu očekivati u pojedinim područjima primjene. Također, primijenjuju se kao pomoć u ranijim fazama složenijih postupaka segmentacije.

2.2 Postupci temeljeni na značajkama

Princip rada postupaka temeljenih na značajkama je slijedeći:

1. Pronađu se određene značajke u pojedinim okvirima.
2. Pronalaze vektori pomaka značajki iz jednog okvira u drugi (traži se korespodencija značajki dvaju okvira).
3. Značajke se grupiraju na temelju pomaka (brzine kretanja značajki) - svaka od takvih grupa značajki predstavlja jedan objekt.

Ovi postupci računaju slikovni tok (engl. *image flow*) u pojedinim točkama slike, te na temelju njega segmentiraju sliku. Za te točke obično se uzimaju uglovi dobiveni nekim detektorom uglova. Zbog smanjenog broja točaka u kojima se računa optički tok, ove metode su podesne za implementaciju u stvarnom vremenu.

Beymer *et al.* [10] koriste ovaj pristup za mjerenje parametara toka prometa na autocesti pomoću panoramske kamere. Smith i Brady [11] izgradili su sustav koji provodi segmentaciju čak i kada se promatrač kreće (slučaj MCMO) i njegovo kretanje nije poznato (automobil koji se kreće po autocesti).

Postupci temeljeni na značajkama primjenjivi su samo na segmentaciju pokretnih objekata, pošto značajke zaustavljenih objekata nije moguće razlikovati od značajki pozadine. Ovi postupci se uglavnom primijenjuju na praćenje automobila, što je razumljivo, jer automobili sadrže veliki broj uglova.

Velika prednost ovih postupaka jest neosjetljivost na promjene osvjetljenja i pomaka promatrača. Nedostatak je u tome što se ne dobiva potpuna slika pokretnog objekta, već samo skup značajki koje se nalaze na objektu. Smith i Brady [11] tome su doskočili tako da su koristeći detektor rubova računali i konture oko skupina značajki, ali tada takva verzija sustava nije bila primjenjiva u stvarnom vremenu.

---

[1] Ovaj postupak korišten je u sustavu za izlučivanje slike vozila iz scene, te je detaljno opisan u poglavlju 7.2.

2.3 Postupci temeljeni na aktivnim konturama

Aktivne konture (engl. *active contours*), još zvane i zmije (engl. *snakes*), su krivulje koje minimiziranjem energije pronalaze konture u sceni. Uveli su ih Kass *et al*. [12]. U oblikovanju aktivnih kontura sudjeluju tri sile [12]:

1. Interne sile (engl. *internal forces*) - elastične sile koje nameću glatkoću konture.
2. Sile slike (engl. *image forces*) - privlače konturu značajkama slike kao što su linije rubovi i sl.
3. Vanjske ograničavajuče sile (engl. *external constraint forces*) - postavljaju konturu u blizinu željenog lokalnog minimuma. Može ih zadati korisnik ili se određuju automatski na temelju nekog mehanizma pozornosti ili interpretacija više razine.

Ako se ove sile dobro postave, aktivna kontura trebala bi dobro opisivati obris određenog objekta u slici.

Aktivne konture imaju svojstvo da ako se postave u blizinu granice objekta konvergiraju toj granici objekta. Zbog tog svojstva mogu se koristiti za praćenje, odnosno segmentaciju u dinamičkim uvjetima. Pod uvjetom da su pomaci objekta između dva slijedna okvira maleni, aktivna kontura koja je opisivala objekt u prošlom okviru predstavlja dobru početnu konturu za pronalaženje obrisa objekta u trenutnom okviru.

Leymarie i Levine [13] koriste model aktivnih kontura za segmentaciju i praćenje objekata promjenjive strukture (pokretnih ljudskih stanica - *pseudopoda*). Sustav je zahtijevao od čovjeka da označi početnu poziciju objekta kako bi se dalje nastavilo praćenje.

Kalafatić [14] koristi kombinaciju aktivnih kontura i rijetkog optičkog toka za segmentaciju i praćenje (većeg broja) laboratorijskih životinja. Pomoću optičkog toka određuje se pomak konture iz prethodnog okvira, te pomoću modela aktivnih kontura popravlja tako dobivena kontura. Korištenjem takvog, kombiniranog, pristupa uspijeva riješiti probleme dodriravanja i zaklanjanja objekata. Za određivanje početnih kontura koristi se statička segmentacija.

Lefèvre *et al*. [15] koriste aktivne konture za praćenje većeg broja nogometaša na terenu. Prilikom praćenja javlja se problem kad se više objekata nalazi jedan do drugoga. Umjesto više kontura, za svaki pojedini objekt, javlja se samo jedna koja obuhvaća sve objekte. Iz tog razloga uvode korak rezanja (engl. *scission step*) konture ukoliko je kontura prevelika. Inicijalizacija kontura radi se ručno.

Aktivne konture predstavljaju dobar način za pronalaženje smislenih kontura u sceni, a time i dobar način za pronalaženje objekata u sceni. Najveći problem s aktivnim konturama je inicijalizacija jer one same nisu u stanju riješiti taj problem. O inicijalizaciji ovisi uspješnost daljnje segmentacije i praćenja i stoga je vrlo važno kvalitetno ju riješiti, poželjno nekim od drugih postupaka segmentacije slike.

2.4 Postupci temeljeni na 3D modelima

Postupci se temelje na *a priori* znanju o sceni i objektima. Znanje o objektima se predstavlja 3D geometrijskim modelima. Postupci traže dokaze prisustva nekog objekta u slici te se zbog toga svrstavaju u pristup odozgo prema dolje.

Sullivan [16] te Koller *et al.* [17] koriste isti pristup za praćenje vozila u sekvencama dobivenim pomoću stacionarne kamere. Pored znanja predočenog u

obliku 3D modela vozila, koristi se i znanje o kretanju vozila. Inicijalne hipoteze modela generiraju se segmentacijom temeljenom na značajkama. Nakon toga se uspoređuju linije u slici, dobivene detekcijom rubova, s linijama koje se dobivaju projekcijom određenog 3D modela kako bi se odredio odgovarajući model i položaj toga modela.

Prednosti postupaka temeljenih na 3D modelima proizlaze iz njihovog opisa scene na visokom nivou. Postupci nisu osjetljivi na promjene oblika, veličine, boje i osvjetljenja, probleme zaklanjanja rješavaju na prirodan način. Rezultat 3D razumijevanja scene omogućuje raspoznavanje objekata i interpretaciju njihovog ponašanja.

Kako bi imalo općenitije razumijevanje svijeta zahtijevalo veliki broj unaprijed zadanih modela čime bi i sam proces podudaranja bio izuzetno računski skup, postupci temeljeni na 3D modelima ograničeni su samo na određeno usko područje primjene.

3

Detekcija promjene

U ovom poglavlju obrađene su dvije metode detekcije promjene: diferencija slika i akumulirana diferencija slika. Premda jednostavne i na niskom nivou, te su metode vrlo korisne pri analizi dinamičkih scena jer pružaju informacije o tome gdje se u sceni događa promjena što uvelike smanjuje složenost daljnjih faza obrade.

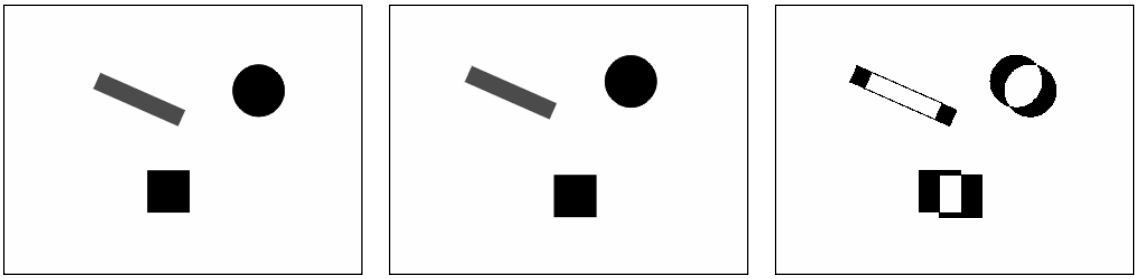
3.1 Diferencija slika

Kao što je već rečeno, vidljiva promjena u sceni izazvat će razliku između okvira sekvence koji odgovaraju toj sceni. Najjednostavniji i najprirodniji način detekcije promjene je međusobno uspoređivanje odgovarajućih slikovnih elemenata dvaju okvira. Ta se metoda naziva *diferencija slika* (engl. *difference pictures*). Diferencija slika definira se izrazom [1]:

$$DP_{jk}(x,y) = \begin{cases} 1 & \text{ako } |F(x,y,j) - F(x,y,k)| > \tau \\ 0 & \text{inače} \end{cases} \tag{3.1}$$

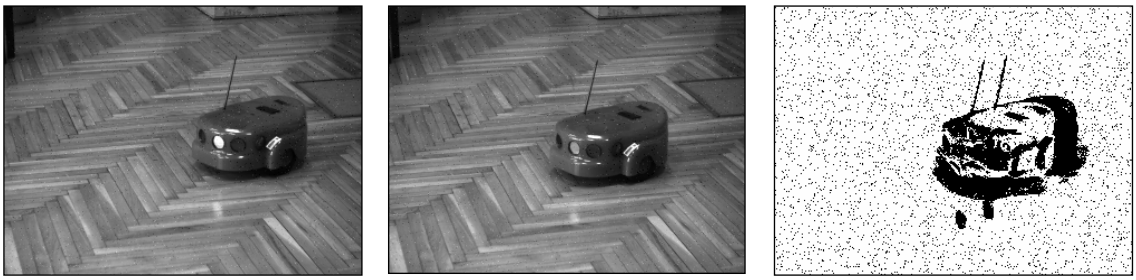
gdje su *F(x,y,j)* i *F(x,y,k)* okviri (definirani kao u uvodnom poglavlju), a τ prag. U daljnjem tekstu će se element diferencije slika koji je označen s 1 nazivati i *element diferencije slika*.

Slika 3.1 sadrži primjer diferencije slika dvaju okvira umjetno generirane scene, dok slika 3.2 sadrži primjer diferencije slika dvaju okvira uzetih iz stvarnog svijeta (u obje slike su elementi označeni s 1 u diferenciji slika prikazani crno).



(a) (b) (c)

Slika 3.1: Diferencija slika na primjeru umjetno generiranih okvira.
Prikazana su dva umjetno generirana okvira (a) i (b), te njihova diferencija slika (c).

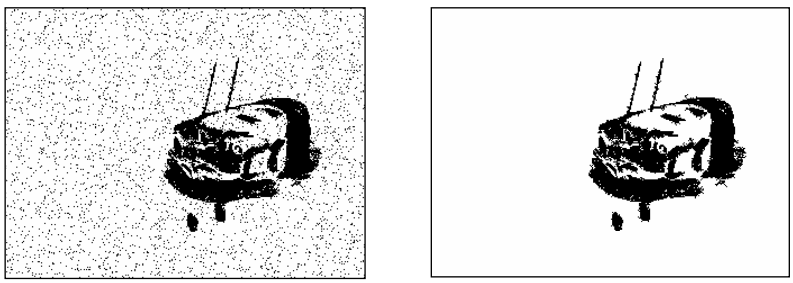


(a) (b) (c)

Slika 3.2: Diferencija slika na primjeru iz stvarnog svijeta.
Prikazana su dva okvira sekvence iz stvarnog svijeta (pokvarena šumom) (a) i (b), te njihovu diferencija slika (c) uz prag $\tau = 15$.

3.1.1 Filter veličine

Pojedini elementi diferencije slika označeni su ako se pripadajući slikovni elementi u promatranim okvirima dovoljno razlikuju. Razlika između okvira može biti posljedica kretanja u sceni, promjene osvjetljenja, ali također i posljedica šuma. Šum rezultira prevelikim brojem elemenata u diferenciji slika. Jedan od načina eliminacije elemenata diferencije slika nastalih pod utjecajem šuma je filter veličine [1].



(a) (b)

Slika 3.3: Primjena filtera veličine na diferenciju slika.
Diferencija slika sa slike 3.2 (a) i rezultat primjene filtera veličine na istu uz prag veličine od 15 elemenata (b).

Princip rada je slijedeći:

1. pronađu se 4-povezani ili 8-povezani elementi diferencije slika, oni zajedno tvore jedno područje,

2. sva područja koja imaju manje elemenata od određenog praga (praga veličine) se izbacuju iz diferencije slika, odnosno više se ne uzimaju u obzir.

Filter veličine je učinkovit u suzbijanju elemenata diferencije slika nastalih pod utjecajem šuma, ali ima jednu manu. Naime, filter veličine uzrokuje i zanemarivanje elemenata diferencije slika nastalih kretanjem malih i/ili sporo krećućih objekata.

Rezultat primjene filtera veličine na diferenciju slika sa slike 3.2 prikazan je na slici 3.3.

3.1.2 Robusna detekcija promjene [1]

Prilikom detekcije razlike među okvirima, umjesto da se promatraju odgovarajući slikovni elementi, mogu se promatrati odgovarajuća područja, okoline. Usporedba odgovarajućih područja okvira na temelju statistike svakako je pouzdanija od usporedbe temeljene samo na pojedinim slikovnim elementima.

Postavlja se pitanje kako formirati područja za usporedbu? Jedan pristup je podjela okvira na tzv. *superpiksele* (engl. *superpixel*), tj. jednaka područja veličine $n \times m$ slikovnih elemenata koja su međusobno disjunktna. Takav pristup prikazan je na slici 3.4a. Drugi pristup, prikazan na slici 3.4b, je upotreba lokalnih maski veličine $n \times m$.

(a) (b)

Slika 3.4: Kreiranje područja za usporedbu.
Originalni dio slike koji se sastoji od 9×9 slikovnih elemenata podijeljen je na disjunktna područja (superpiksele) veličine 3×3 slikovna elementa (a). Lokalna maska veličine 3×3 slikovna elementa postavljena na jedan slikovni element (b). (Izvor [1])

U literaturi [1][3][4] predložen je jedan od mogućih načina usporedbe područja korištenjem takozvane *mjere vjerodostojnosti* (engl. *likelihood ratio*). Mjera vjerodostojnosti definirana je slijedećim izrazom:

$$\lambda = \frac{\left[\frac{\sigma_1+\sigma_2}{2}+\left(\frac{\mu_2-\mu_1}{2}\right)^2\right]^2}{\sigma_1 \cdot \sigma_2} \tag{3.2}$$

gdje su μ_1 i σ_1 aritmetička sredina i varijanca sivih vrijednosti područja uspoređivanja u prvom okviru, a μ_2 i σ_2 aritmetička sredina i varijanca sivih vrijednosti područja uspoređivanja u drugom okviru. Korištenjem mjere vjerodostojnosti definira se diferencija slika:

$$DP_{jk}(x,y) = \begin{cases} 1 & \text{ako } \lambda > \tau \\ 0 & \text{inače} \end{cases} \tag{3.3}$$

gdje je τ prag.

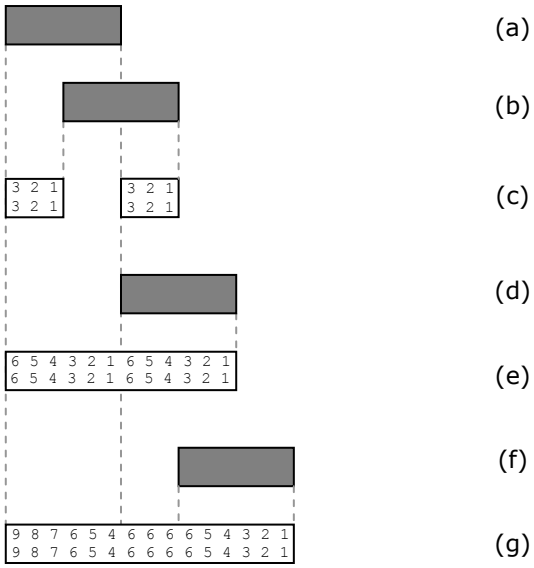
3.2 Akumulirana diferencija slika

Akumulirana diferencija slika (engl. *accumulated difference pictures*) [1][4] dobiva se uspoređivanjem svakog okvira sekvence s referentnim okvirom i povećavanjem vrijednosti elementa akumulirane diferencije slika za 1 svaki put kada su odgovarajući slikovni elementi referentnog i promatranog okvira dovoljno različiti. Akumulirana diferencija slika može se definirati izrazima:

$$ADP_0(x,y) = 0 \tag{3.4}$$

$$ADP_k(x,y) = ADP_{k-1}(x,y) + DP_{1k}(x,y) \tag{3.5}$$

Ovi izrazi vrijede za slučaj kada je prvi okvir sekvence referentni okvir. Princip akumulirane diferencije slika prikazan je na slici 3.5.



Slika 3.5: Shematski prikaz akumulirane diferencije slika.
Princip akumulirane diferencije slika prikazan je na primjeru objekta koji se pomiče za jednu poziciju udesno po okviru. (a) Položaj objekta u prvom, referentnom okviru, (b) položaj objekta nakon tri pomaka, (c) akumulirana diferencija slika nakon tri pomaka objekta, (d) položaj objekta nakon što se pomaknuo za svoju duljinu, (e) akumulirana diferencija slika nakon što se objekt pomaknuo za svoju duljinu, (f) položaj objekta nakon što se pomaknuo za više od svoje duljine, (g) akumulirana diferencija slika nakon što se objekt pomaknuo za više od svoje duljine.

Akumulirana diferencija slika uzima u obzir više od dva okvira sekvence, pa sadrži i veću količinu informacija od obične diferencije slika. Za razliku od obične diferencije slika, akumulirana diferencija slika u stanju je detektirati i male i/ili sporo krećuće objekte.

Pored ovako zadane apsolutne akumulirane diferencije slika u literaturi [1] se spominju i pozitivna *PADP* i negativna akumulirana diferencija slika *NADP* definirane preko pozitivne *PDP* i negativne *NDP* diferencije slika:

$$PDP_{jk}(x,y) = \begin{cases} 1 & \text{ako } F(x,y,j) - F(x,y,k) > \tau \\ 0 & \text{inače} \end{cases} \tag{3.6}$$

$$NDP_{jk}(x,y) = \begin{cases} 1 & \text{ako } F(x,y,j) - F(x,y,k) < \tau \\ 0 & \text{inače} \end{cases} \tag{3.7}$$

$$PADP_k(x,y) = PADP_{k-1}(x,y) + PDP_{1k}(x,y) \tag{3.8}$$

$$NADP_k(x,y) = NADP_{k-1}(x,y) + NDP_{1k}(x,y) \tag{3.9}$$

4

Segmentacija upotrebom pokreta

Nakon detekcije pokreta i identificiranja područja slike u kojima se događaju promjene, slijedeći je zadatak segmentacija slike. Metode obrađene u ovom poglavlju nazivaju se tzv. procesima nižeg nivoa (engl. *low-level processes*) jer se temelje na slikovnim elementima, za razliku od metoda koje se baziraju na konturama, područjima, značajkama i sličnom.

4.1 Korištenje akumulirane diferencije slika

U slučaju *SCMO* (stacionarna kamera, pokretni objekti) moguće je segmentirati scenu na jednostavan način, korištenjem diferencije slika i akumulirane diferencije slika.

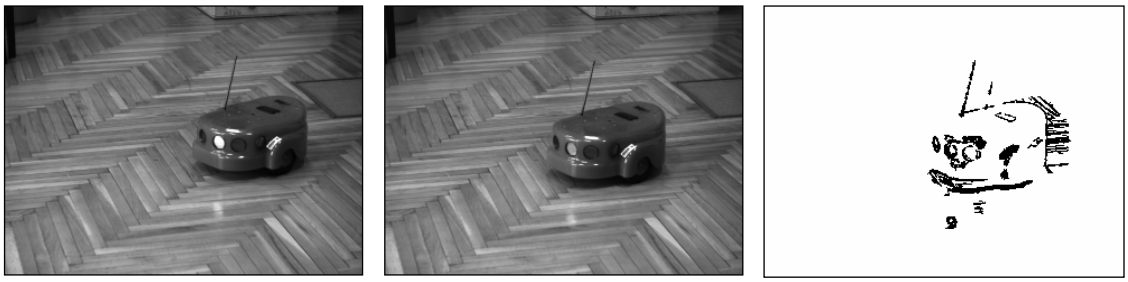
U jednostavnijim slučajevima, kada ne dolazi do zaklanjanja objekata drugim objektima (engl. *occlusion*), odnosno pomaka nekog objekta na mjesto koje je prethodno zauzimao drugi objekt (engl. *running occlusion*), moguće je dobiti masku objekta u referentnom okviru korištenjem pozitivne, odnosno negativne akumulirane diferencije slika [1]. Naime, nakon što se objekt potpuno pomakne sa svoje pozicije u referentnom okviru, veličina područja u *PADP* ili u *NADP* (ovisno o tome da li je tamnija pozadina ili objekt) koje odgovara kretanju objekta prestaje rasti. To područje odgovara slici objekta u referentnom okviru.

Jain i Nagel [4] opisuju metodu koja koristi određena svojstva akumulirane diferencije slika za određivanje nestacionarnih komponenti scene. Ta metoda uspješno je primijenjena na scene iz stvarnog svijeta.

4.2 Detekcija vremenski promjenjivih rubova

Detekcija rubova jedan je od temeljnih postupaka segmentacije slike. Kada se neki objekt u sceni kreće, kreću se i rubovi koji mu pripadaju - detekcijom vremenski promjenjivih rubova [1] (engl. *time-varying edge*) detektiramo objekte koji se

kreću. Ova metoda ima očitu prednost nad običnom detekcijom rubova - detektira samo rubove objekata koji se kreću, dok ostale (statične) rubove ne detektira.



(a) (b) (c)

Slika 4.1: Detekcija vremenski promjenjivih rubova.
Prikazana su dva okvira sekvence uzete iz stvarnog svijeta (a) i (b), te rezultat detekcije vremenski promjenjivih rubova (c). Operator detekcije vremenski promjenjivih rubova primijenjuje prag na umnožak prostornog i vremenskog gradijenta. Također je primijenjen i filter veličine s pragom 10 za eliminaciju šuma.

Operator koji daje vrijednost vremenski promjenjive rubnosti neke točke definiran je kao [1]:

$$E_t(x,y,t) = \frac{dF(x,y,t)}{ds} \cdot \frac{dF(x,y,t)}{dt} = E(x,y,t) \cdot D(x,y,t) \qquad (4.1)$$

gdje je *dF(x,y,t)*/*ds* vrijednost prostornog, a *dF(x,y,t)*/*dt* vrijednost vremenskog gradijenta u točki (*x,y,t*) Za prostorni gradijent može se koristiti bilo koji detektor rubova (engl. *edge detector*), dok se za vremenski gradijent može koristiti neka od metoda detekcije razlike između slikovnih elemenata (kao prilikom računanja diferencije slika). Valja napomenuti da se za operator množenja u izrazu (4.1) može koristiti operator logičko **I**, tj. točka pripada vremenski promjenjivom rubu ako njezina rubnost prelazi neki prag **I** ako vrijednost razlike u između promatranih okvira u toj točki prelazi neki prag. Taj operator također može biti implementiran i kao operator množenja na način da se vrijednosti vremenskog i prostornog gradijenta pomnože, a da se zatim primijeni prag na taj produkt.

Na slici 4.1 prikazan je rezultat detekcije vremenski promjenjivih rubova na dva okvira sekvence uzete iz stvarnog svijeta. Iz slike se vidi nedostatak postupka detekcije vremenski promjenjivih rubova: rubovi koji se nalaze u pozadini se detektiraju kada ih objekt prestane zaklanjati.

4.3 Označena diferencija slika

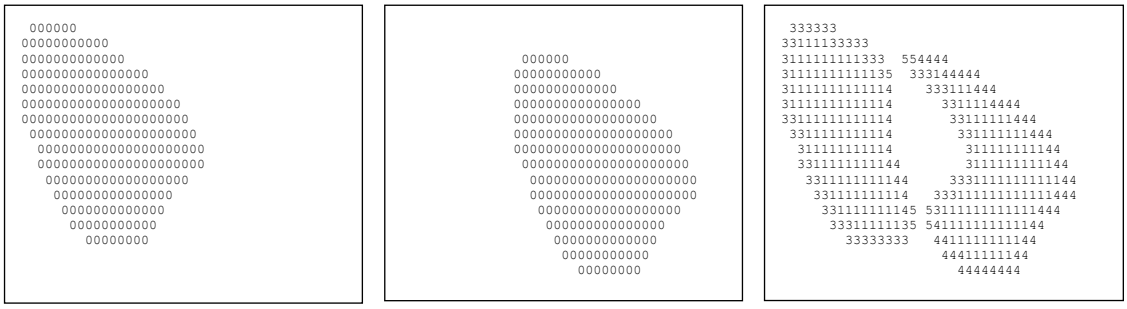
Označena diferencija slika (engl. *labeled difference pictures*) [3] je kombinacija diferencije slika i detekcije vremenski promjenjivih rubova.

Označena diferencija slika formira se iz obične diferencije slika na slijedeći način:

- element diferencije slika je rubni element, ako je označen s 1 i barem jedan od njegovih neposrednih susjeda je označen s 0;
- rubni element pripada prethodnom rubu (rubu prethodnog okvira), ako vrijednost nekog detektora ruba na tom mjestu u prethodnom okviru prelazi neki prag;

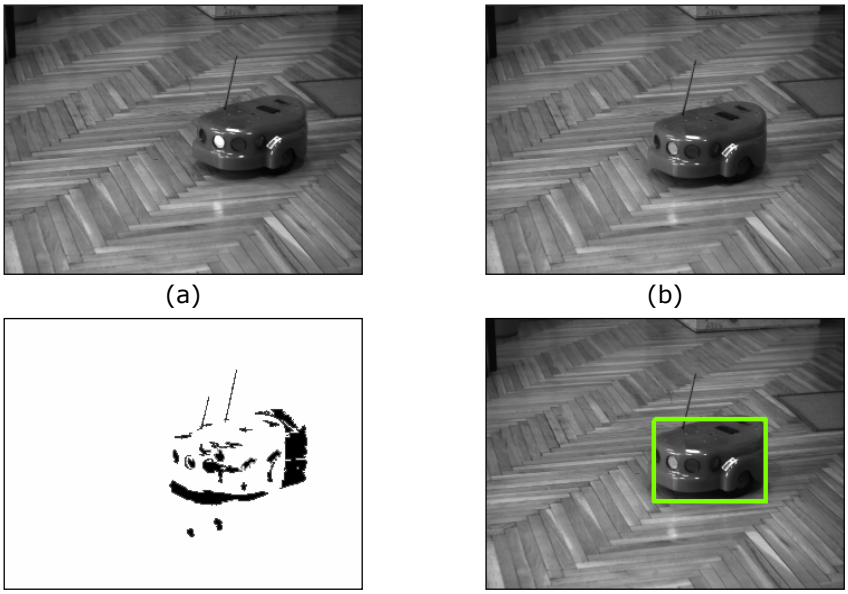
- rubni element pripada trenutnom rubu (rubu trenutnog okvira), ako vrijednost nekog detektora ruba na tom mjestu u trenutnom okviru prelazi neki prag.

Na slici 4.2 simbolički je prikazan princip označene diferencije slika.



(a) (b) (c)

Slika 4.2: Princip označene diferencije slika.
Prikazana su dva umjetno generirana okvira (a) i (b), te njihova označena diferencija slika (c). Elementi označene diferencije slika označeni su na slijedeći način: 1 - ne rubni elementi, 3 - rubni elementi koji pripadaju prethodnom rubu, 4 - rubni elementi koji pripadaju trenutnom rubu, 5 - rubni elementi koji pripadaju i prethodnom i trenutnom rubu.



(a) (b) (c) (d)

Slika 4.3: Segmentacija korištenjem označene diferencije slika.
Prikazani su prethodni okvir (a), trenutni okvir (b), označena (sivim vrijednostima) diferencija slika (c), trenutni okvir s (opisanim pravokutnikom) označenom pozicijom objekta (d). Zbog velike količine rubova u pozadini (parket) postavljen je vrlo visok prag za rubnost.

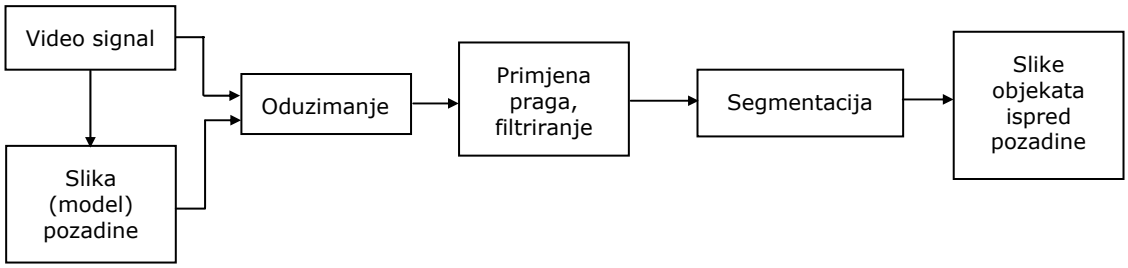
Označena diferencija slika sadrži više informacija od obične diferencije slika. Pomoću nje moguće je, u nekim slučajevima iz stvarnog svijeta, dosta točno odrediti položaj objekta u prethodnom i trenutnom okviru (pomoću prethodnih i trenutnih rubova). Koliko točno, ovisi o obliku objekta, njegovom gibanju, odnosu njegovih sivih vrijednosti prema sivim vrijednostima pozadine, te o količini rubova u pozadini. Na slici 4.3 prikazano je određivanje pozicije objekta pomoću označene diferencije slika na primjeru dvaju okvira sekvence uzete iz stvarnog svijeta.

Pravokutnik koji okružuje objekt u prethodnom ili u trenutnom okviru je u stvari pravokutnik koji okružuje sve prethodne ili trenutne rubove u označenoj diferenciji slika.

U radu [3] opisan je klasifikator koji na temelju svojstava označene diferencije slika svrstava njena područja u jednu od osam kategorija koje odgovaraju različitim tipovima pokreta. Ipak, mišljenje je autora ovog rada da je takav klasifikator neprimjenjiv na scene stvarnog svijeta jer da bi takav klasifikator funkcionirao, objekti moraju biti homogeni (u smislu sivih razina), a to je u stvarnom svijetu rijetkost.

4.4 Oduzimanje pozadine

Oduzimanje pozadine (engl. *background subtraction*) često je korištena metoda segmentacije dinamičkih scena. Ideja je vrlo jednostavna: ukoliko posjedujemo sliku pozadine, lako dobivamo sliku objekata ispred te pozadine uspoređivanjem trenutne slike scene i slike pozadine (slika 4.4). Postupak oduzimanja pozadine koristi diferenciju slika za detekciju promjene i dobivanje maske objekta ispred pozadine (slika 4.5).



Slika 4.4: Principjelna shema postupaka oduzimanja pozadine.

(a) (b) (c)

Slika 4.5: Oduzimanje pozadine pomoću diferencije slika.
Prikazana je slika pozadine (a), trenutni okvir (b), te maska objekta ispred pozadine dobivena korištenjem diferencije slika (c).

Ovako prikazan, postupak oduzimanja pozadine izgleda kao jako dobar način segmentacije jer daje kompletnu masku objekta u sceni bez gotovo ikakvog računanja. Ipak, postupak oduzimanja pozadine pokazuje brojne probleme [7][8][18]:

- **Pomicanje pozadinskih objekata.** Objekti koji su smatrani pozadinom u jednom trenutku se mogu pomaknuti, odnosno ukloniti iz pozadine. Na mjestima na kojima su se prije nalazili tada se pogrešno detektira postojanje objekta.
- **Inicijalizacija pozadine s pokretnim objektima.** U mnogim slučajevima slika pozadine bez krećućih objekata nije poznata te se javljaju slični problemi kao i sa pomicanjem pozadinskih objekata.
- **Titranja u pozadini.** U pozadini se mogu nalaziti objekti poput drveća, monitora i sl. koji pokazuju stalno titranje i uzrokuju pogreške u detekciji objekata.
- **Kamuflaža.** Objekti iznad pozadine mogu imati jednake vrijednosti intenziteta ili boje kao i pozadina, te se zbog toga ne mogu detektirati.
- **Promjene osvjetljenja.** U otvorenom prostoru javljaju se promjene u osvjetljenju, bilo zbog doba dana ili promjene vremenskih prilika. Zbog toga se javljaju razlike u odnosu na sliku pozadine koje ne odgovaraju objektima ispred pozadine. Ovakvi problemi javljaju se i u zatvorenim prostorima, npr. ako se upali ili ugasi svjetlo u prostoriji.
- **Sjene i međusobne refleksije.** Objekti u sceni općenito bacaju sjene oko sebe. Te sjene se pogrešno detektiraju kao pokretni objekti. Slični problemi se dešavaju zbog refleksije na glatkim i sjajnim površinama.

Da bi se riješili ti problemi potrebno je na određeni način modelirati pozadinu tako da se prilagođava promjenama u sceni. Razvijeni su brojni načini prilagođavanja, odnosno modeliranja pozadine. U poglavlju 2.1 navedeni su neki od načina te su navedeni radovi koji ih koriste.

5

Aktivni vid

U uvodnom poglavlju je izneseno kako dinamički vid predstavlja korak bliže biološkim sustavima za vid. Ipak, dinamički vid nije u potpunosti analogan biološkim sustavima vida. Razlog tomu leži u činjenici da životinje (i ljudi) nisu samo pasivni promatrači svoje okoline već aktivno promatraju okolinu: usmjeravaju pažnju na nešto, traže nešto pogledom, okreću glavu kako bi nešto vidjeli, šire zjenice oka, fokusiraju pogled na određenu udaljenost te na temelju vida vrše interakciju s okolinom.

Aktivni (engl. *active vision*) vid predstavlja još jedan korak bliže biološkim sustavima za vid jer podrazumijeva povratnu vezu prema senzoru (osjetilu) na temelju ulaznih podataka dobivenih vizualnim putem. Aktivni vid definira se kao aktivno ili adaptivno upravljanje svim parametrima kamere [19][1] u cilju što učinkovitije obrade. Parametri kamere uključuju zoom, orijentaciju, fokus i otvor blende.

U nastavku poglavlja dan je kratak uvod u važnija područja aktivnog vida: pozornost, fovealni vid, upravljanje pogledom te koordinacija oko-ruka.

5.1 Pozornost

Slika scene obično sadrži veći broj podataka nego što je potrebno za obavljanje zadataka sustava računalnog vida. Također, zbog računske složenosti, velika količina podataka onemogućuje obradu u stvarnom vremenu. Zbog toga je važno ograničiti vid na posebna područja interesa. Proces usmjeravanja na područja interesa naziva se pozornost (engl. *attention*).

Pozornost u aktivnom vidu analogna je pozornosti u biološkim sustavima za vid: pozornost se usmjerava na ono što je u nekom trenutku važno za zadatak koji se obavlja dok se ostale informacije zanemaruju. Načini ne koje se određuje na što će se usmjeriti pozornost mogu biti različiti: nagle promjene u sceni zaokupit će

[1] Ovaj rad predstavlja osnovni izvor ovog poglavlja i stoga nadalje neće biti posebno navođen.

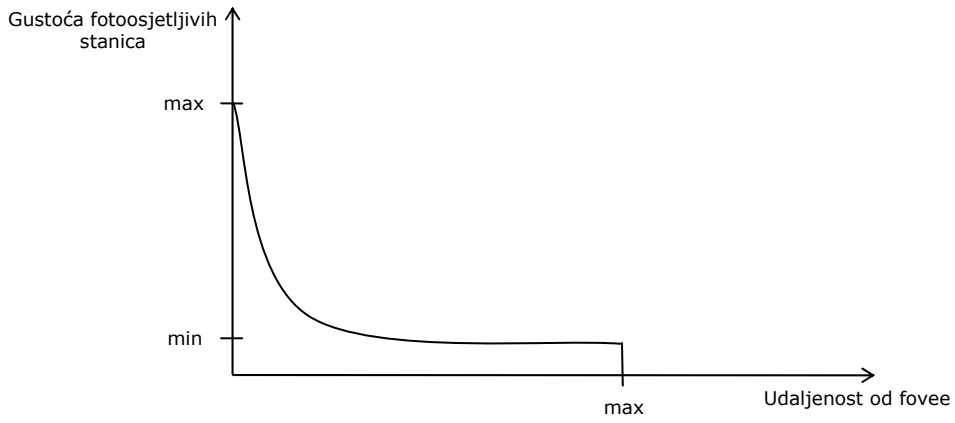
pozornost svakoga; kada nešto tražimo, usmjeravamo pozornost na dio po dio scene.

Osnovno obilježje pozornosti jest selekcija. Od velike količine ulaznih podataka uzima se samo bitan dio čime se pojednostavljuje obrada te povećava njena učinkovitost, a time i brzina.

5.2 Fovealni vid

Fovealni vid (engl. *foveal sensing*, *foveal vision*), kao dio aktivnog vida, odnosi se na korištenje senzora s nejednakim prostornim rasporedom (engl. *spatially variant sensor*), tj. senzora čija razlučivost nije jednaka na svakom mjestu.

Sam naziv fovealni vid dolazi od analogije s ljudskim okom, odnosno vidnim sustavom. U središnjem dijelu mrežnice (*retinae*) nalazi se područje najjasnijeg vida s najgušćim rasporedom fotoosjetljivih stanica (štapića i čunjića) tzv. žuta mrlja (*macula lutea*). U središtu žute mrlje nalazi se plitka središnja udubina *fovea centralis* [20]. Kako gustoća fotoosjetljivih stanica opada s udaljenošću od fovee (slika 5.1), većina vizualne moći koncentrirana je na područje interesa i ne troši se na nevažne objekte.



Slika 5.1: Ovisnost gustoće fotoosjetljivih stanica o udaljenosti od fovee u ljudskom oku.
(Izvor [19])

Slično kao što vrijedi za biološke sustave vida, sustavi za računalni vid mogu koristiti prednosti prostorno promjenjive razlučivosti kako bi smanjili količinu ulaznih podataka, a opet imali mogućnost dobivanja detaljne i oštre slike područja od interesa i širok kut gledanja. Kako bi postavio foveu (tj. točku najviše rezolucije) na područje interesa sustav fovealnog vida mora imati mogućnost upravljanja pogledom.

Wodnicki, Roberts i Levine [21] implementirali su fovealni senzor (engl. *foveated sensor*) u CMOS tehnologiji. Raspored receptivnih polja (engl. *receptive fields*) u fovealnom senzoru naziva se log-polarnim (engl. *log-polar mapping*).

5.3 Upravljanje pogledom

Upravljanje pogledom (engl. *gaze control*) odnosi se na aktivno upravljanje parametrima kamere kako bi se pomoglo u ostvarivanju nekog zadatka računalnog vida. Upravljanje pogledom dalje se dijeli na stabilizaciju pogleda (engl. *gaze stabilisation*) i promjenu pogleda (engl. *gaze change*).

Stabilizacija pogleda odnosi se na fiksaciju, smanjenje razlika između okvira kako bi se dobila čista slika promatranog objekta. U slučaju da se promatrani objekt kreće stabilizacija pokreta uključuje i aktivno praćenje.

Promjena pogleda usko je vezana uz pozornost. Pozornost određuje što (i gdje) će se gledati - promjena pogleda tamo usmjerava pogled. Dakle, promjena pogleda obuhvaća mehanizme pomicanja stabiliziranog pogleda na novu lokaciju.

5.4 Koordinacija oko-ruka

Koordinacija oko-ruka (engl. *eye-hand coordination*) obuhvaća upravljanje dvama sustavima, sustavom vida (jedna ili više kamera) i sustavom za manipuliranje (robotska ruka).

Uključivanje sustava za manipuliranje u sustav aktivnog vida omogućuje proširivanje znanja o objektima u sceni i sceni uopće. Na temelju vizualne percepcije scene može se upotrijebiti robotska ruka za dobivanje dodatnih informacija o čvrstoći, strukturi, težini objekta i slično. Nadalje, robotska ruka može izvoditi neke operacije s objektima koje vizualni sustav percipira, kako bi izvršila određeni zadatak.

Neki od problema kojima se bavi koordinacija oko-ruka su [22]:

- načini predstavljanja okoline, objekata, senzora i manipulatora, tj. njihovo modeliranje;
- učenje o svojstvima objekata i njihovim funkcijama, učenje načina obavljanja pojedinih akcija i rješavanja problema;
- samokalibracija sustava putem izdavanja naredbi za pomak manipulatora i promatranjem učinka.

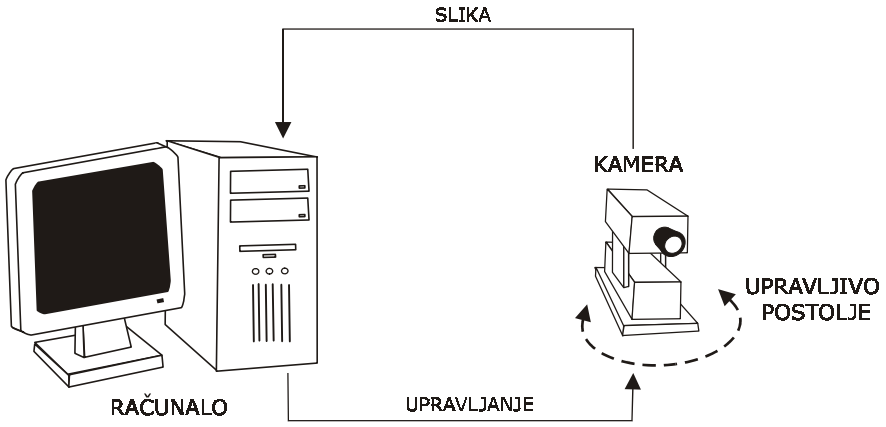
6

Sustav za aktivno praćenje pokretnih objekata - SAPPO

U sklopu ovog rada razvijen je sustav za aktivno praćenje pokretnih objekata korištenjem diferencija slika. Zadaća sustava je, da pomoću senzora - kamere i upravljivog postolja na kojem se ta kamera nalazi, pronalazi pokretne objekte u sceni te ih prati usmjeravajući kameru prema njima (u kontekstu aktivnog vida - stabilizacija pogleda, poglavlje 5.3). Ovo poglavlje sadrži detaljan opis tog sustava.

6.1 Opis sustava

Pojednostavljeno, sustav se sastoji od kamere, upravljivog postolja i računala. Slika dobivena iz kamere se obrađuje u računalu koje na temelju obrade upravlja postoljem na kojem se nalazi kamera (slika 6.1).

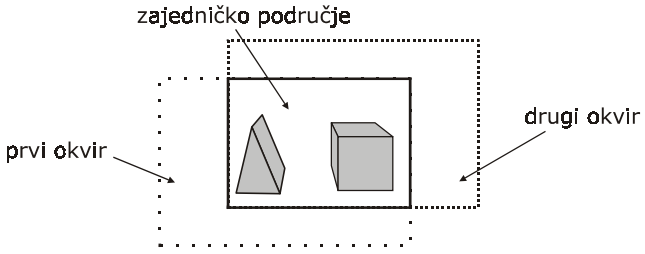


Slika 6.1: Komponente sustava za praćenje pokretnih objekata.

Obrada se odvija u tri glavne faze. U prvoj fazi određuje se pomak slike uslijed gibanja kamere, kako bi se pomak kamere mogao kompenzirati. Korištenjem podatka o pomaku slike, u drugoj fazi se računa označena diferencija slika zajedničkih dijelova trenutnog i prethodnog okvira, te se dobivene regije grupiraju kako bi se dobila pozicija objekta za praćenje. U trećoj fazi se zadaje brzina i smjer gibanja upravljivog postolja kako bi se kamera usmjerila prema objektu koji se prati.

6.2 Prva faza: određivanje pomaka kamere

Prilikom praćenja objekta pomiču se i objekt i kamera, što nas vodi na slučaj dinamičke scene *MCMO*. Pošto se pozicija objekta dobiva pomoću diferencije slika dvaju slijednih okvira, potrebno je na neki način odrediti pomak slike, odnosno pozadine, između tih okvira kako bi se diferencija slika provodila samo nad zajedničkim dijelom scene (slika 6.2).



Slika 6.2: Područje preklapanja dvaju okvira.
Prikazana su dva međusobno pomaknuta okvira i njihovo zajedničko područje.

Najjednostavniji način određivanja pomaka kamere bio bi pomoću upravljivog postolja. Korišteno postolje omogućuje određivanje apsolutne pozicije u svakom trenutku, no taj pristup nije dao zadovoljavajuće rezultate. Jedan od razloga je taj što je potrebno jako točno kalibrirati kameru i postolje u smislu određivanja broja slikovnih elemenata koji odgovaraju zakretu postolja (kamere) za određeni kut. No taj problem je moguće riješiti. Nepremostiv problem takvog pristupa je brzina rada. Naime, komunikacija s korištenim postoljem vrlo je spora (RS-232, 9600 bps), te je za rad u stvarnom vremenu potrebno svesti ju na minimum.

Jedini preostali način određivanja pomaka kamere je pomoću slike koju kamera daje. Ako objekt koji pratimo ne zauzima veći dio slike scene tada je moguće odrediti pomak kamere preko dominantnog gibanja u sceni - gibanja pozadine. Gibanje pozadine može se odrediti tako da se uparuju točke zajedničke za oba okvira. Ako smo u stanju odrediti pomak jedne točke pozadine u dva slijedna okvira, tada je problem određivanja pomaka kamere, odnosno pomaka slike uslijed gibanja kamere, riješen.

Dakle, kako bi se odredio pomak slike između dva okvira uslijed gibanja kamere potrebno je odabrati određene točke, odnosno značajke, na svakom od okvira, zatim korištenjem određenog kriterija odrediti koja točka jednog okvira odgovara kojoj točki drugog okvira, za svaki par točaka izračunati vektor razlike (engl. *disparity vector*) i na kraju odrediti dominantnu vrijednost (onu koja se najčešće pojavljuje, mod) vektora razlike (pri čemu je važan uvjet da stacionarni dio scene zauzima najveći dio slike). Slijedi detaljan opis postupka korištenog u SAPPO-u.

6.2.1 Detekcija točaka (značajki) za uparivanje

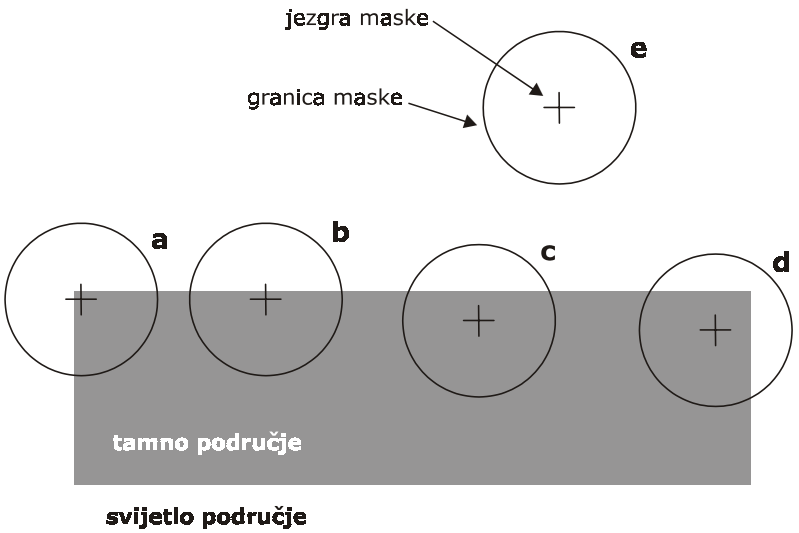
Uparivanje točaka u dvije različite slike koristi se u mnogim područjima računalnog vida kao što su stereo vid (engl. *stereo imaging*) i računanje slikovnog toka (engl. *image flow*). Razvijeni su mnogi operatori koji koji izdvajaju pojedine *interesantne točke*. Takve točke moraju posjedovati određena svojstva, a najvažnije je da su stabilne, tj. ako je jedna točka detektirana na jednoj slici onda mora biti detektirana i na svakoj drugoj slici koja ju sadrži. Za takve točke se obično uzimaju uglovi (engl. *corners*) odnosno dvodimenzionalne značajke (engl. *two dimensional features*).

U SAPPO-u se za detekciju dvodimenzionalnih značajki koristi detektor uglova SUSAN (engl. *SUSAN corner detector*). Ovaj detektor odabran je zbog svoje brzine i pouzdanosti.

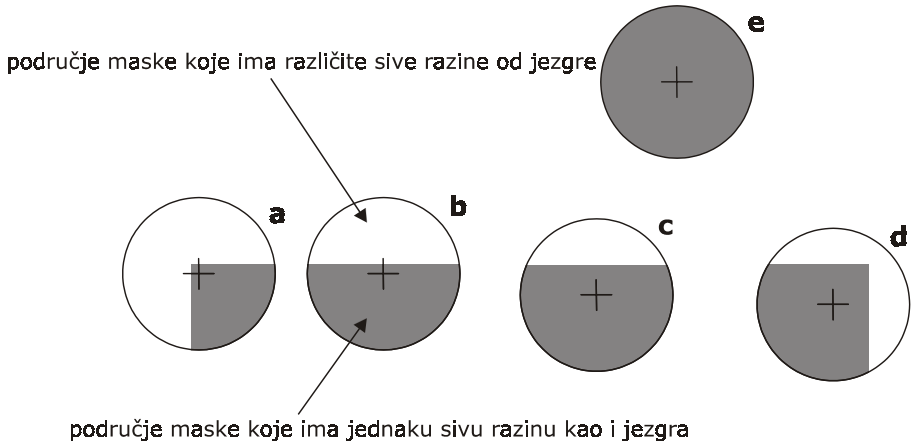
SUSAN detektor uglova temelji se na SUSAN (*Smallest Univalue Segment Assimilating Nucleus*) principu [23] koji se koristi još i za detekciju rubova i eliminaciju šuma. Ovdje će biti opisan SUSAN princip i njegovo korištenje za detekciju uglova.

Prevedeno na hrvatski SUSAN je *najmanji segment jednake vrijednosti koji asimilira jezgru*. Značenje tog naziva biti će jasno iz SUSAN principa. Promotrimo sliku 6.3, koja prikazuje kružnu masku (čije se središte naziva jezgra) na pet različitih mjesta u slici koja sadrži tamni pravokutnik na svijetloj pozadini. Segment jednake vrijednosti koji asimilira jezgru, USAN, definira se kao područje kružne maske koje ima jednaku (ili sličnu) vrijednost sivih razina kao i jezgra maske. Na slici 6.4 prikazane su maske sa slike 6.3 s označenim USAN područjima. Iz slika 6.3 i 6.4 se vidi da je veličina USAN područja maksimalna kada se maska nalazi cijelom površinom unutar jednolike površine, da pada na polovicu maksimuma kada se maska nalazi na rubu površine, a da je manje od polovice maksimuma kada se maska nalazi na kutu.

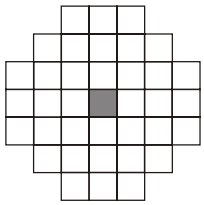
Maska je implementirana diskretnom aproksimacijom kruga radijusa 3.4 slikovna elementa što daje masku od ukupno 37 slikovnih elemenata (slika 6.5).



Slika 6.3: SUSAN princip.
Pet kružnih maski postavljene su na različita mjesta jednostavne slike. (Izvor [23])



Slika 6.4: USAN područja.
Pet kružnih maski sa slike 6.3 s označenim USAN područjima (sivo). (Izvor [23])



Slika 6.5: Diskretna aproksimacija kružne maske korištene u implementaciji SUSAN detektora uglova.
Prikazana je diskretna aproksimacija korištene kružne maske. Jezgra maske označena je sivo. (Izvor [23])

Detekcija uglova odvija se na slijedeći način za svaki pojedini slikovni element. Na promatrani slikovni element postavi se kružna maska (s jezgrom na slikovnom elementu). Pomoću izraza

$$c(\vec{r},\vec{r}_0) = \begin{cases} 1 & \text{ako } \left|I(\vec{r}) - I(\vec{r}_0)\right| \le t \\ 0 & \text{inače} \end{cases} \tag{6.1}$$

ili izraza

$$c(\vec{r},\vec{r}_0) = e^{-\left(\frac{I(\vec{r})-I(\vec{r}_0)}{t}\right)6} \tag{6.2}$$

i izraza

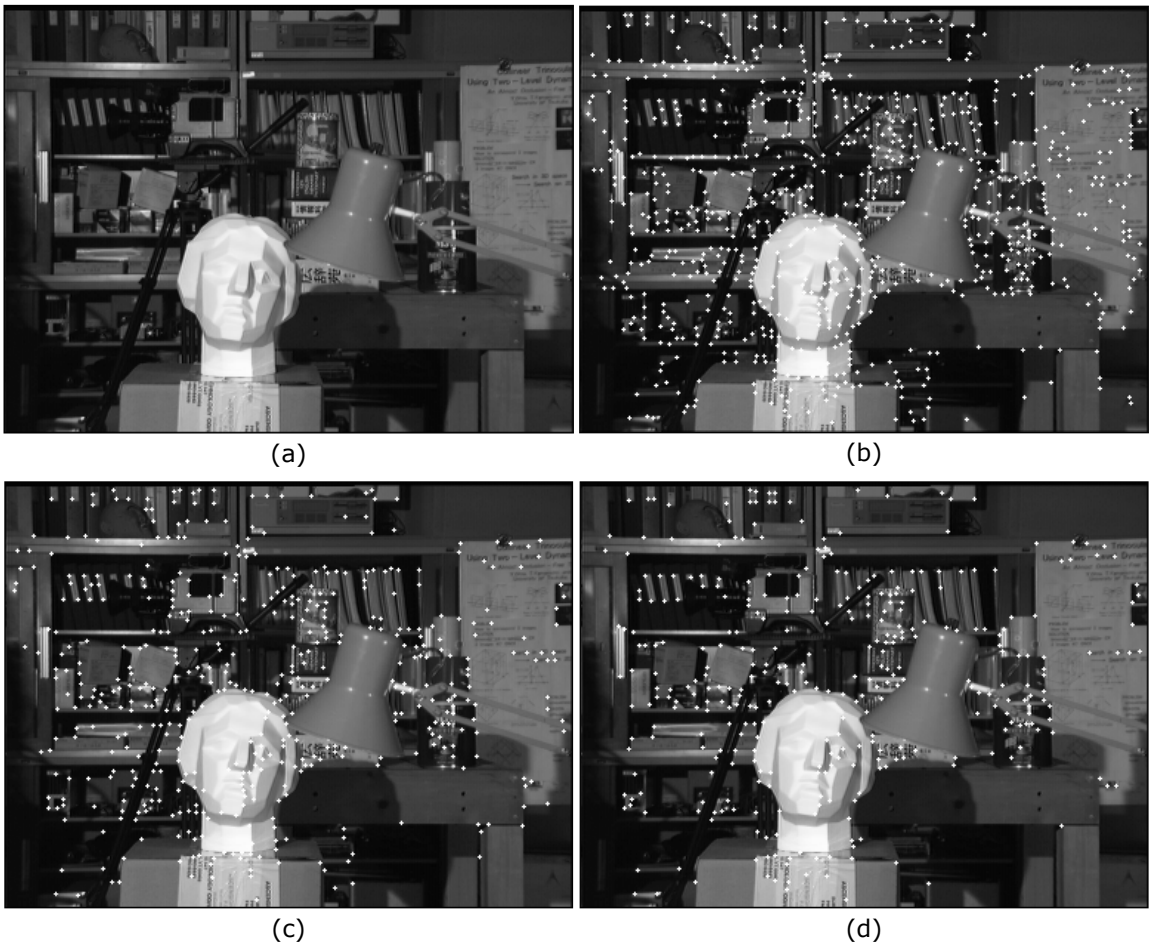
$$n(\vec{r}_0) = \sum_{\vec{r}} c(\vec{r},\vec{r}_0) \tag{6.3}$$

(gdje je $\vec{r}$ pozicija svake točke unautar maske, $\vec{r}_0$ pozicija jezgre maske, $I(\vec{r})$ intenzitet točke na poziciji $\vec{r}$, t prag sivih vrijednosti) izračuna se n - broj elemenata USAN područja. n se zatim uspoređuje s geometrijskim pragom g da bi se dobio odziv $R(\vec{r}_0)$:

$$R(\vec{r}_0) = \begin{cases} g - n(\vec{r}_0) & \text{ako } n(\vec{r}_0) < g \\ 0 & \text{inače} \end{cases} \tag{6.4}$$

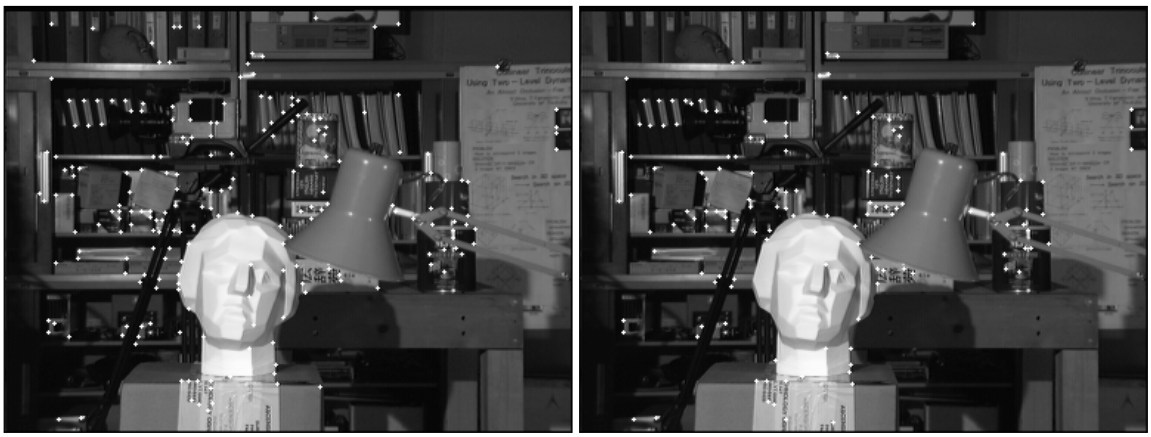
Geometrijski prag *g* postavlja se na manje od polovice maksimalne vrijednosti za *n*, n_{max} (SUSAN princip). Geometrijski prag *g* određuje kvalitetu detektiranih zančajki (uglova), dok prag sivih vrijednosti *t* određuje osjetljivost - broj detektiranih značajki. Kada je izračunat odziv za sve slikovne elemente, provodi se potiskivanje ne-maksimalnih elemenata (engl. *non maxima suppression*) korištenjem maske veličine 5×5.

U korištenoj implementaciji SUSAN detektora uglova [24] moguće je zadavanje praga sivih vrijednosti *t*. Slika 6.6 prikazuje ovisnost broja detektiranih uglova o vrijednosti parametra *t*. Vidi se kako broj detektiranih uglova pada s porastom vrijednosti praga *t*.



(a) (b)

(c) (d)



(e) (f)

Slika 6.6: Ovisnost broja značajki detektiranih SUSAN detektorom uglova o pragu *t*. Prikana je ulazna slika (a), te ulazna slika s označenim uglovima detektiranim SUSAN detektorom uglova za vrijednosti parametra *t*=10 (b), *t*=20 (c), *t*=30 (d), *t*=40 (e), *t*=50 (f).

6.2.2 Uparivanje značajki

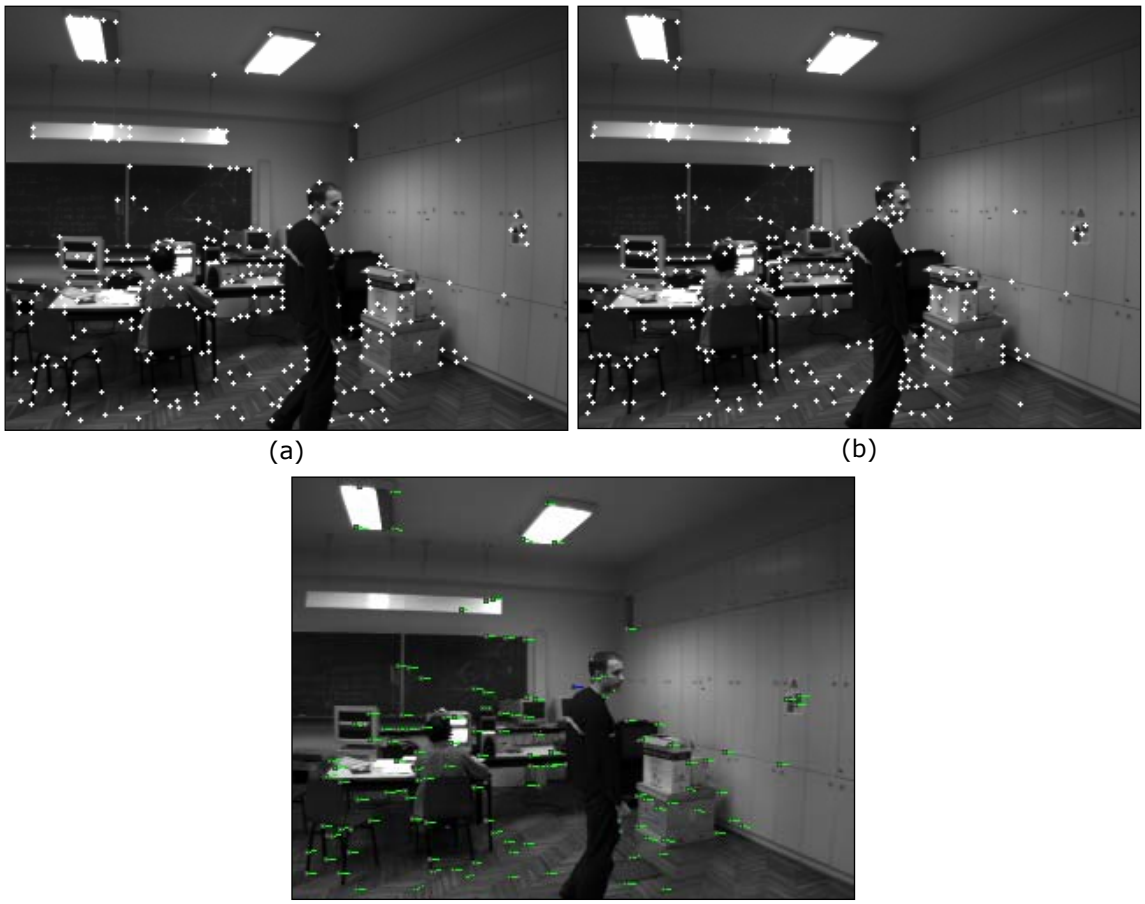
Kada su pronađene značajke (uglovi) u oba okvira, potrebno je odrediti koja značajka prvog okvira odgovara kojoj značajki drugog okvira. Jedan pristup, često korišten u stereo vidu i za percepciju pokreta, je računanje sličnosti okolina pronađenih značajki. Drugi pristup [11] koristi sive vrijednosti i prostorne derivacije u promatranim točkama.

U SAPPO-u korišten je drugi pristup. Jedan razlog je što se za detekciju uglova koristio SUSAN detektor uglova [24] koji svakoj točki koja je detektirana kao ugao pridružuje i njenu sivu razinu i prostorne derivacije u toj točki. Drugi, presudni, razlog je veća brzina uparivanja.

Postupak uparivanja je slijedeći:

1. Za svaku značajku prvog okvira promatraju se sve značajke drugog okvira čije su prostorne koordinate unutar određene okoline (ovisi o maksimalnoj brzini kretanja koja se očekuje).
2. Odabire se značajka drugog okvira koja ima najsličniju sivu vrijednost i najsličnije prostorne derivacije značajki iz prvog okvira i koja uparena s značajkom iz prvog okvira daje vektor pomaka sličan pripadnom vektoru pomaka (ako postoji) značajke iz prvog okvira (koji je preuzet iz prethodnog uparivanja).

Ovakvo jednostavno uparivanje daje veći broj točno uparenih značajki od broja netočno uparenih značajki. Točnost uparivanja ocijenjena je ljudskim okom i to na više primjera gibanja kamere. Primjer rezultata uparivanja dan je na slici 6.7.



(a) (b)

(c)

Slika 6.7: Primjer pronađenih vektora pomaka značajki.
Prikazani su prethodni (a) i trenutni (b) okvir sekvence s označenim uglovima, te trenutni okvir s označenim izračunatim vektorima pomaka značajki (c).

6.2.3 Pronalaženje dominantnog gibanja

Uparivanjem značajki na dva slijedna okvira dobivaju se odgovarajući vektori pomaka. Neki od vektora mogu odgovarati gibanju nekog objekta u sceni, drugi su rezultat pogrešnog uparivanja, a neki su poslijedica gibanja kamere. Ako je ispunjen uvjet da pozadina zauzima najveći dio slike, onda je pomak slike uslijed gibanja kamere jednak dominantnom gibanju - gibanju pozadine.

Iz svih vektora pomaka pronalazi se pomak koji se najčešće pojavljuje. Način određivanja vektora pomaka u SAPPO-u je slijedeći:

1. Vektori se grupiraju po sličnosti, tako da su vektori unutar jedne grupe vrlo slični (gotovo jednaki).
2. Pronalazi se grupa koja sadrži najveći broj elemenata - grupa koj sadrži dominanti pomak.
3. Metodom LMedS (engl. *least median of squares*) [25] se pronađe "sredina" takve grupe - vektor pomaka slike uslijed kretanja kamere.

6.3 Druga faza: detekcija pokreta i lociranje pokretnog objekta

U poglavlju 4.3 je pokazano kako se pomoću označene diferencije slika može odrediti pozicija objekta. Kako je pokazano na slici 4.3 iz označene diferencije slika može se jednostavno dobiti pravokutnik koji okružuje pokretni objekt. Takav postupak segmentacije koristi se u SAPPO-u.

Označena diferencija slika računa se na slijedeći način:

1. Izračuna se diferencija slika korištenjem izraza (3.1).
2. Na dobivenu diferenciju slika primijeni se filter veličine. Korišten je postupak sekvencijalnog pronalaženje povezanih komponenti [1] (engl. *sequential connected components algorithm*) i to u varijanti 8-povezanosti.
3. Korištenjem Sobelovog operatora detekcije rubova [1] se odredi koje rubne točke diferencije slika pripadaju prethodnom, odnosno trenutnom rubu.

Postupak detekcije i lociranja pokretnog objekta radi jako dobro u slučaju kada kamera miruje (*SCMO*). Kada se kamera kreće dolazi do problema. Ranije je spomenuto da se određuje pomak slike uslijed pomaka kamere kako bi se pronašlo zajedničko područje scene dvaju slijednih okvira (slika 6.2), a zatim izračunala označena diferencija slika za to područje okvira. Jedina razlika u tom području okvira trebala bi biti poslijedica kretanja objekta, te bi se pomoću te razlike (označene diferencije slika) odredio trenutni položaj objekta. Na žalost, ove pretpostavke ne vrijede uvijek pa dolazi do pojave razlika u zajedničkom području okvira koje nisu uzrokovane gibanjem pokretnog objekta (ili pokretnih objekata). Postoji više razloga iz kojih ove pretpostavke ne vrijede.

Jedan od razloga je što se kamera ne giba translacijski, već se rotira oko neke osi. Zajedničko područje okvira stoga nije jednako. Za male kuteve pomaka kamere između dva slijedna okvira može se dobiti gotovo jednako zajedničko područje i zato je nužno postići što manji pomak između dva slijedna okvira, što vodi na mali vremenski pomak između dva slijedna okvira, odnosno veliku brzinu obrade.

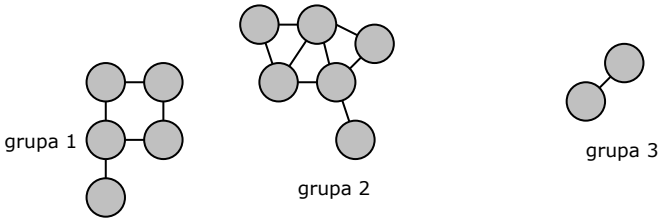
Još jedan razlog su i pogreške leća kamere koje iskrivljuju sliku. Pogreške leća najviše dolaze do izražaja uz rub slike pa će se slika jednog objekta koji se u jednom okviru nalazi uz rub slike razlikovati od slike istog tog objekta na drugom mjestu (bliže središtu slike) u drugom okviru. Razlika će biti manja što je manji pomak slike objekta između promatranih okvira što opet vodi na zahtjev za velikom brzinom obrade.

Pošto zahtjev za malim pomakom između slijednih okvira nije uvijek u potpunosti ispunjen, trebalo je pronaći određenu metodu za eliminaciju područja u označenoj diferenciji slika koje su posljedica spomenutih pogrešaka. Promatranjem izgleda označene diferencije slika prilikom horizontalnog gibanja kamere utvrđena su slijedeća svojstva pogrešnih aktivnih regija:

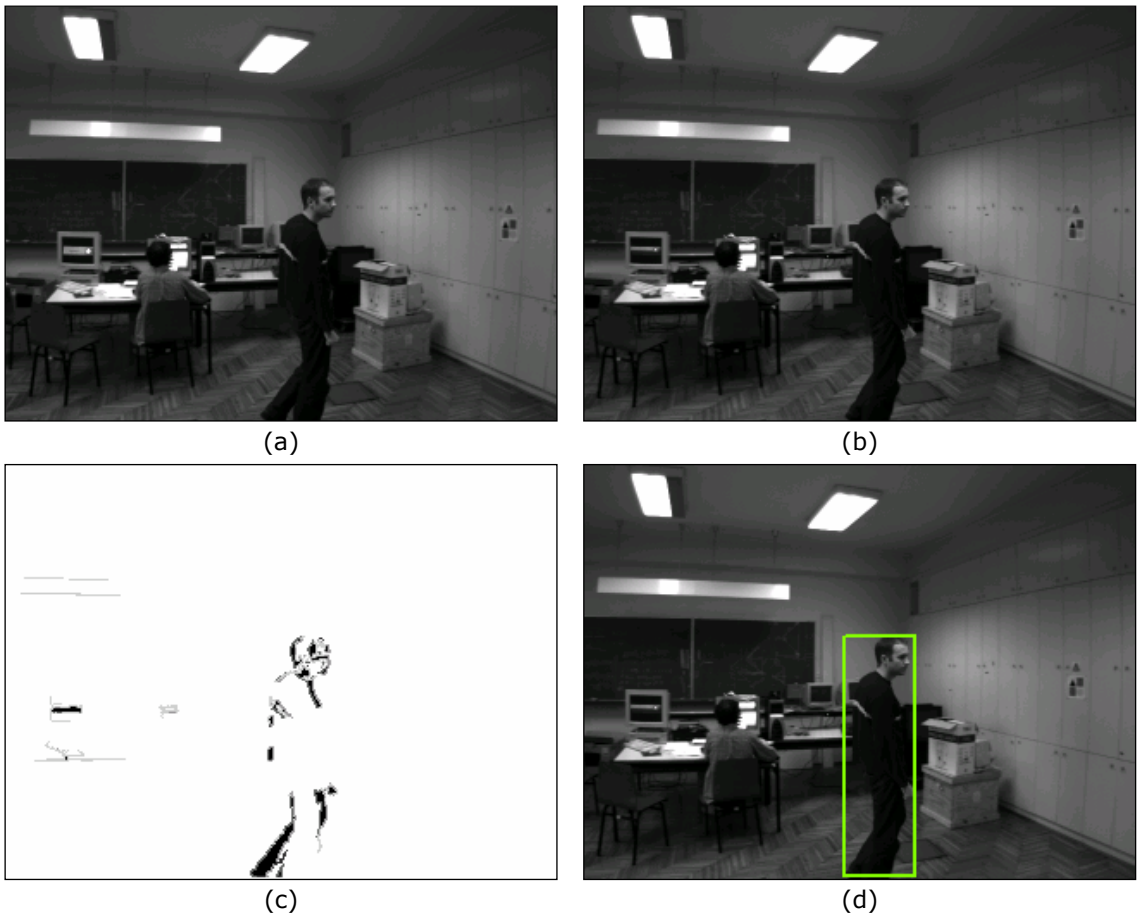
1. odgovaraju izrazitim rubovima koji se nalaze u pozadini,
2. sadrže veliki broj elemenata koji su označeni kao rubovi u odnosu na ukupni broj elemenata,
3. uglavnom se pojavljuju uz rub slike.

Za razliku od pogrešnih aktivnih područja, glavno svojstvo aktivnih područja koje odgovaraju kretanju objekata je da imaju manji broj elemenata koji su označeni kao rubovi u odnosu na ukupni broj elemenata (pod uvjetom da se objekti dovoljno brzo kreću). Uvidom u te podatke, javlja se jednostavan način eliminacije pogrešnih

aktivnih regija - svako aktivno područje čiji omjer rubnih elemenata u odnosu na ne rubne elemente prelazi određeni prag se zanemaruje. Korištenje takve metode dalo je dobre rezultate, pa je stoga i ugrađena u sustav.



Slika 6.8: Principjelna shema grupiranja na temelju teorije grafova.
Prikazani krugovi predstavljaju čvorove grafa. Povezanost čvorova znači da je njihova međusobna udaljenost manja od praga.



Slika 6.9: Primjer detekcije i određivanja pozicije pokretnog objekta.
Prikazani su prethodni (a), trenutni (b) okvir sekvence uzete iz stvarnog svijeta, pripadna označena diferencija slika (c) i trenutni okvir s označenom pozicijom objekta (d).

Pored područja koje su nastale zbog pogrešaka a nisu eliminirana, javljaju se i područja koja su posljedica određenih promjena koje ne želimo pratiti (npr. titranje monitora). U sceni se također može pojaviti i više pokretnih objekata. Sustav se na neki način mora odlučiti koji će objekt pratiti. Taj problem rješava se tako da se

aktivna područja grupiraju po kriteriju udaljenosti. Upotrijebljen je algoritam grupiranja na temelju teorije grafova (slika 6.8): aktivna područja predstavljaju čvorove grafa, dva se čvora grafa povezuju ako je udaljenost među njima manja od nekog praga, svaki odvojeni dio grafa (komponenta povezanosti grafa) predstavlja jednu grupu.

Sustav odabire grupu koja sadrži najveći broj aktivnih regija. Razlog tomu je što se sustav koristio za praćenje ljudi koji pri kretanju uzrokuju pojavu većeg broja aktivnih regija. Da bi se izbjeglo usmjerivanje kamere prema malim aktivnim regijama koje mogu biti posljedica pogreške ili titranja u pozadini, ne prate se grupe čiji je opisani pravokutnik manji od nekog zadanog pravokutnika (u biti praga veličine objekta). Na žalost, sustav zbog toga ne reagira na manje objekte, no taj kompromis je bio potreban kako bi se smanjila osjetljivost na pogreške.

Na slici 6.9 prikazana su dva okvira sekvence uzete iz stvarnog svijeta koja odgovara slučaju *MCMO*, pripadna označena diferencija slika i trenutni okvir sekvence s označenom pozicijom pokretnog objekta koji se prati.

6.4 Treća faza: usmjeravanje kamere prema pokretnom objektu

Kada je objekt lociran potrebno je usmjeriti kameru prema njemu. Korišten je jednostavan, ali dovoljno dobar, algoritam praćenja upravljanjem brzinom zakreta upravljivog postolja:

1. Odredi se vektor razlike trenutne pozicije objekta i središta slike koju daje kamera.
2. Ako je razlika trenutne pozicije objekta i središta slike dovoljno malena onda se brzina ne mijenja tj. preskaču se slijedeći koraci.
3. Pojedine komponente tog vektora se pretvore u kuteve (za to je potrebno znati širinu vidnog polja kamere).
4. Brzina svake osi upravljivog postolja se računa kao $\omega = C \cdot \varphi$, gdje je φ kut dobiven u koraku 2, a C konstanta. Vrijednost te konstante određuje koliko će vremena proteći dok se kamera pomakne za kut φ. To vrijeme iznosi $1/C$ sekundi.
5. Ako objekt nije detektiran (pretpostavlja se da se prestao kretati), onda se kamera (postolje) zaustavlja u trenutku za koji se očekuje da je kamera usmjerena na poziciju objekta. To vrijeme se određuje pomoću konstante C.

Na prvi pogled se čini da bi se moglo pratiti pokretni objekt na neki drugi, bolji način. Pošto se iz označene diferencije slika može odrediti trenutna i prethodna pozicija objekta, može se odrediti gdje će se taj objekt nalaziti u slijedećem trenutku, pa se može kameru usmjeriti na to mjesto. No, taj pristup se pokazao pogrešnim iz razloga što je nije moguće točno odrediti pozicije objekta iz označene diferencije slika. Označena diferencija slika samo otprilike određuje poziciju objekta u prethodnom i trenutnom okviru, te pozicije uvelike ovise o načinu i brzini kretanja objekta, o odnosu sivih vrijednosti objekta i pozadine, te o količini rubova u sceni. Najbolje što se može učiniti jest usmjeriti kameru prema mjestu na kojem se objekt trenutno nalazi - smanjiti razliku između središta slike i položaja objekta.

Na žalost, u trenutnoj verziji sustava objekti se prate samo u horizontalnom smjeru. Jedan razlog tome je što nije moguće, na prije opisan način, eliminirati pogrešne aktivne regije kada se kamera giba i horizontalno i vertikalno, a drugi

razlog je smanjenje brzine obrade uslijed izdavanja naredbi za oba smjera što dovodi do prevelikog pomaka između slijednih okvira.

6.5 Opis implementacije sustava

SAPPO je implementiran korištenjem *ljuske za računalni vid* (*CVSH*) razvijene na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva u programskom jeziku *C++*. Svi opisani postupci implementirani su u programskom jeziku *C++*. Ljuska *CVSH* omogućava jednostavno dodavanje novih postupaka obrade, te sadrži programsku podršku koja omogućava korištenje različitih kamera i upravljivih postolja.

Podaci o korištenoj sklopovskoj i programskoj opremi za realizaciju sustava prikazani su u tablici 6.1.

Tablica 6.1: Sklopovska i programska oprema korištena u implementaciji SAPPO-a.

tip komponente	komponenta	opis
kamera	*Basler a301f*	digitalna kamera na sabirnici *IEEE 1394*
upravljivo postolje	*Directed Perception PTU-46-17.5*	RS-232, 9600 bps, binarno sučelje
pribavljanje slike	*Procomp 1394r*	PCI kartica, sučelje za *IEEE 1394* sabirnicu, standardno programsko sučelje
računalo	2×*AMD Athlon Palomino MP* @ 1.4 GHz	
operacijski sustav	*RedHat Linux 7.2*, custom kernel 2.4.18	
C++ prevodioc	*g++* v3.0.2	

Prvotno je, prilikom razvoja sustava, korištena digitalna kamera *Videre DCAM*, no ona je kasnije zamijenjena digitalnom kamerom *Basler a301f* zbog njenih boljih karakteristika. Razlog zamjene kamere leži u pojačanim zahtjevima na oštrinu slike zbog detekcije uglova. Obje kamere za komunakaciju koriste sabirnicu *IEEE 1394*.

Uvođenjem kamere *Basler a301f* javila se potreba za smanjivanjem ulazne slike. Ulazna slika veličine 640×480 slikovnih elemenata nije pogodna za obradu u stvarnom vremenu te se smanjuje na format 320×240. Brzina obrade slike veličine 320×240 omogućuje rad u stvarnom vremenu. Velika brzina obrade nužna je za ispravan rad sustava jer je mali vremenski pomak između slijednih okvira od presudne važnosti.

U sustavu se koristi upravljivo postolje tvrtke *Directed Perception* model *PTU-46-17.5*. Postolje je odabrano zbog velike brzine kretanja (najveća brzina je 300° u sekundi) i naprednih mogućnosti upravljanja (upravljanje zadavanjem apsolutne i relativne pozicije, upravljanje zadavanjem brzine i smjera kretanja - također apsolutno i relativno, mogućnosti zadavanja brzine i akceleracije). Postolje se može zakretati horizontalno (engl. *pan*) i vertikalno (engl. *tilt*), ali sustav (iz već spomenutih razloga) koristi samo horizontalno zakretanje. Mana ovog upravljivog

postolja je spora serijska komunikacija (*RS-232*) koja jako umanjuje mogućnosti njegovog korištenja.

Obrada se izvodi na dvoprocesorskom računalu. Obrada koristi oba procesora, jedan procesor se koristi za dobavljanje okvira i za ostale poslove vezane uz rad ljuske, dok drugi procesor odrađuje opisane faze koje se koriste u praćenju.

6.6 Eksperimenti

Kako bi se odredio utjecaj pojedinih parametara korištenih postupaka na rad sustava i pronašla kombinacija parametara što bliža optimalnoj, bilo je potrebno provesti veliki broj eksperimenata. Ovdje će biti prikazano nekoliko eksperimenata vezanih uz ugađanje važnijih parametara sustava. Također će biti prikazani i eksperimenti kojima se testira uspješnost rada sustava, tj. praćenja.

Promatra se ponašanje sustava s obzirom na točnost lociranja pokretnog objekta u ovisnosti o slijedećim parametrima:

1. prag sivih razina u postupku SUSAN detekcije uglova, *t*;
2. prag odnosa broja rubnih elemenata i ukupnog broja elemenata aktivnih regija označene diferencije slika u postupku eliminacije pogrešnih regija, *r*;
3. prag udaljenosti u postupku grupiranja aktivnih regija, *d*;
4. prag diferencije slika u postupku označene diferencije slika, τ;
5. prag veličine za filter veličine primijenjen u postupku označene diferencije slika, *N*.

Osim ovih parametara, postoji još i parametar rubnosti korišten u postupku označene diferencije slika, no utjecaj tog parametra neće biti razmatran. Razlog tomu je neovisnost postupka o tom parametru, jer se gotovo svi rubni elementi diferencije slika označuju kao točke prethodnog i trenutnog ruba zbog prisustva rubova u pozadini.

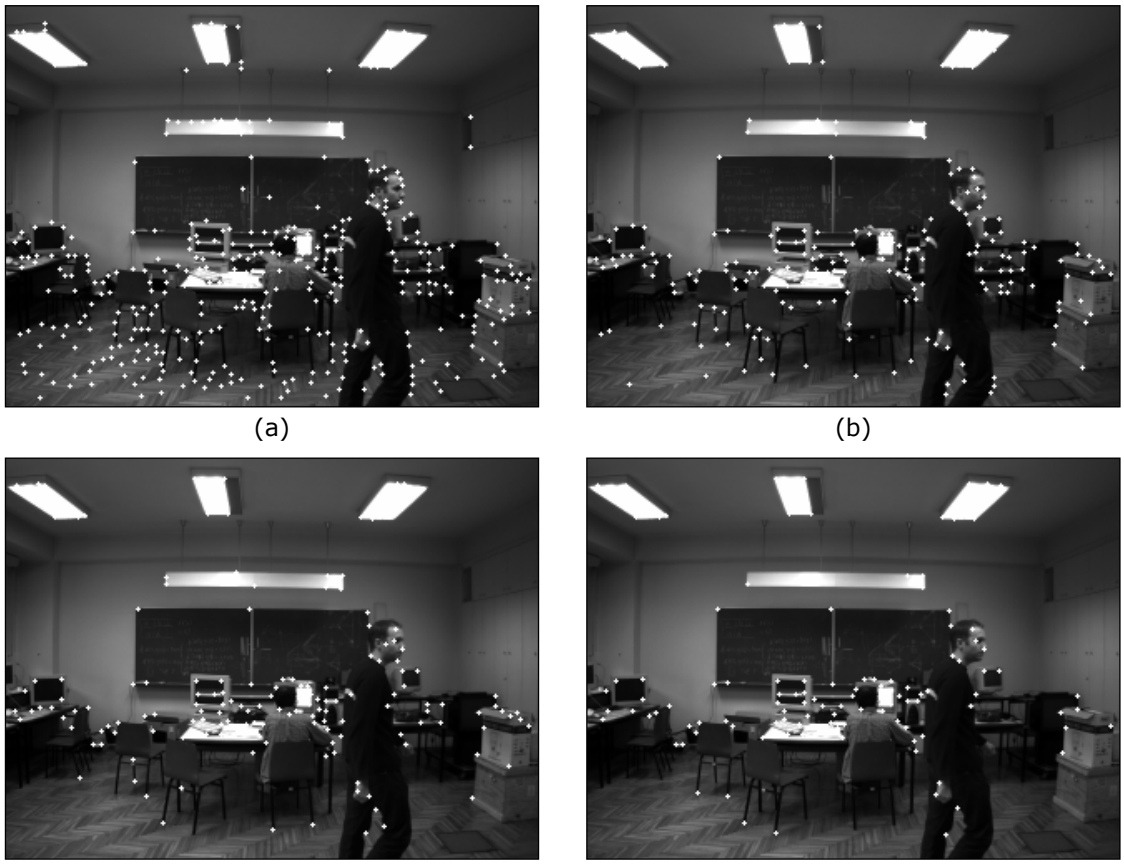
Eksperimenti su provedeni na sekvenci koja prikazuje praćenje čovjeka u hodu, snimljenoj tokom rada sustava. Razlog provođenja eksperimenata na jednoj sekvenci jest u prvom redu ponovljivost eksperimenata i zadržavanje jednakih uvjeta za provjeru svakog parametra. Drugi razlozi za to su jednostavnost i brzina provođenja eksperimenata.

Određivanje kombinacije parametara koja je što bliže optimalnoj ne može se provoditi za sve moguće kombinacije različitih vrijednosti parametara, jer to vodi na jako veliki broj eksperimenata (npr. ako bi se za svaki parametar provjeravalo samo 5 različitih vrijednosti, dobio bi se potreban broj eksperimenata 5^5=3125). Iz tog razloga koristi se drugačija strategija: poredaju se parametri po svojoj važnosti za rad sustava (važnost je ocijenjena iskustvom) i za svaki parametar se pronalazi vrijednost za koju se sustav najbolje ponaša, držeći pritom ostale parametre nepromijenjenima (na "ugođenoj" vrijednosti, ako je pronađena, ili na nekoj vrijednosti za koju se sustav dobro ponaša).

6.6.1 Utjecaj parametra *t*

Kao što je već navedeno (poglavlje 6.2.1) parametar *t* određuje količinu uglova detektiranih u slici korištenjem SUSAN detektora uglova. Slika 6.10 prikazuje jedan okvir korištene sekvence s detektiranim uglovima za različite vrijednosti parametra *t*. Iz slike se jasno vidi kako veća vrijednost parametra *t* dovodi do manjeg broja detektiranih uglova.

Pošto se prvo eksperimentira s ovim parametrom to znači da je njegova važnost za rad sustava najveća. Na prvi pogled se čini da tomu nije tako. No, eksperimenti će pokazati da ispravnost rada faze u kojoj se određuje pomak slike uslijed kretanja kamere jako ovisi o broju detektiranih značajki, tj. o parametru *t*, a ta je faza upravo presudna za ispravan rad sustava jer predstavlja preduvjet za ispravan rad ostalih postupaka.



(a) (b)

(c) (d)

Slika 6.10: Primjer ovisnosti broja uglova detektiranih SUSAN detektorom uglova o pragu sivih vrijednosti *t*.
Prikazani su rezultati detekcije uglova za t=10 (a), t=20 (b), t=30 (c) i t=40 (d).

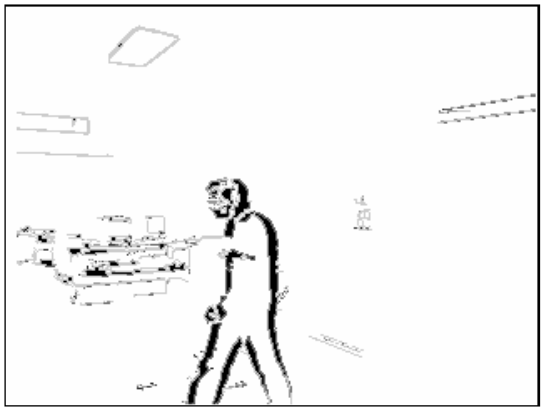
Eksperimentom je određivan broj okvira u kojima postupak određivanja pomaka slike uslijed pomaka kamere nije dao zadovoljavajuće rezultate, što je ocijenjeno promatranjem označene diferencije slika. Ocjenjuje se da postupak nije uspio ukoliko u označenoj diferenciji slika postoji preveliki broj pogrešnih aktivnih regija.

U ovom eksperimentu uzete su slijedeće vrijednosti ostalih parametara: τ=15, N=20.

Tablica 6.2: Broj neuspjeha za pojedine vrijednosti parametra *t*.

t	10	20	30	40	50
broj neuspjeha/ ukupan broj	15/135	9/135	6/135	6/135	8/135

U tablici 6.2 prikazan je broj okvira za koje postupak nije uspio za pet različitih vrijednosti parametra *t*. Slika 6.11 prikazuje označene diferencije slika dva različita okvira za dobivene uz različite vrijednosti parametra *t*.

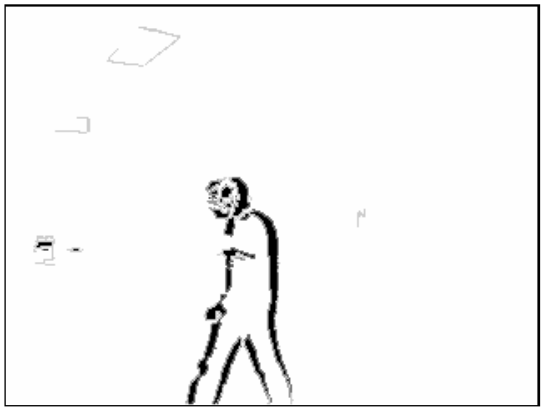


(a)

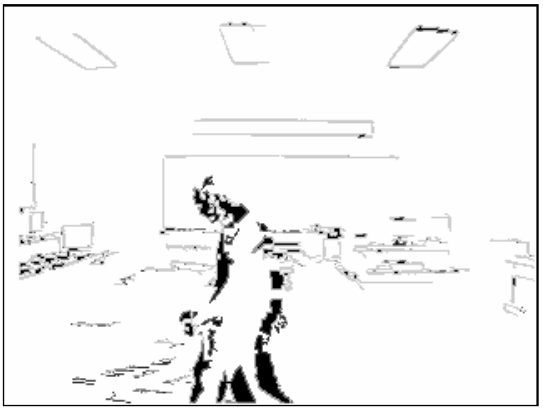
(b)

(c)

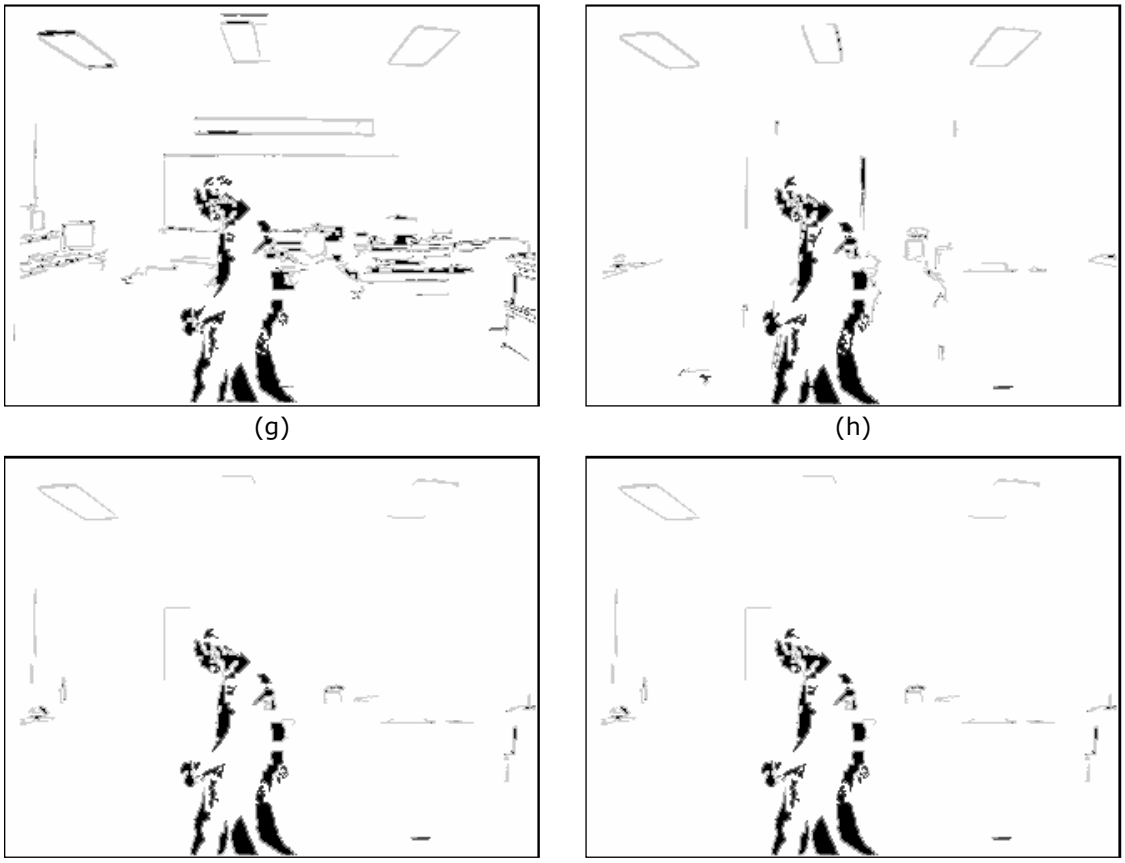
(d)



(e)



(f)



(g) (h)

(i) (j)

Slika 6.11: Primjer označenih diferencija slika dvaju različitih okvira za različite vrijednosti parametra *t*.
(a) do (e) diferencije slika za okvir sekvence #95 za vrijednosti parametra *t* redom 10, 20, 30, 40, 50. (f) do (j) diferencije slika za okvir sekvence #121 za vrijednosti parametra *t* redom 10, 20, 30, 40, 50.

Rezultati eksperimenta pokazuju da postupak pronalaženja pomaka slike uslijed kretanja kamere najbolje radi (za ovu sekvencu) kada parametar *t* poprima vrijednosti između 30 i 40.

6.6.2 Utjecaj parametra *r*

Vrijednost ovog parametra određuje za koje aktivne regije označene diferencije slika će pretpostavljati da su posljedica kretanja pokretnog objekta, a za koje će se pretpostavljati da su rezultat pogreške. U obzir se uzimaju samo aktivne regije za koje vrijedi da im je omjer broja elemenata koje su označeni kao rubovi u prethodnom ili trenutnom okviru i ukupnog broja elemenata manji od praga *r*. Aktivne regije kod kojih je taj omjer manji od praga se zanemaruju, tj. nemaju nikakav utjecaj na rad sustava. Iz tog razloga važno je pronaći dobru vrijednost za ovaj prag.

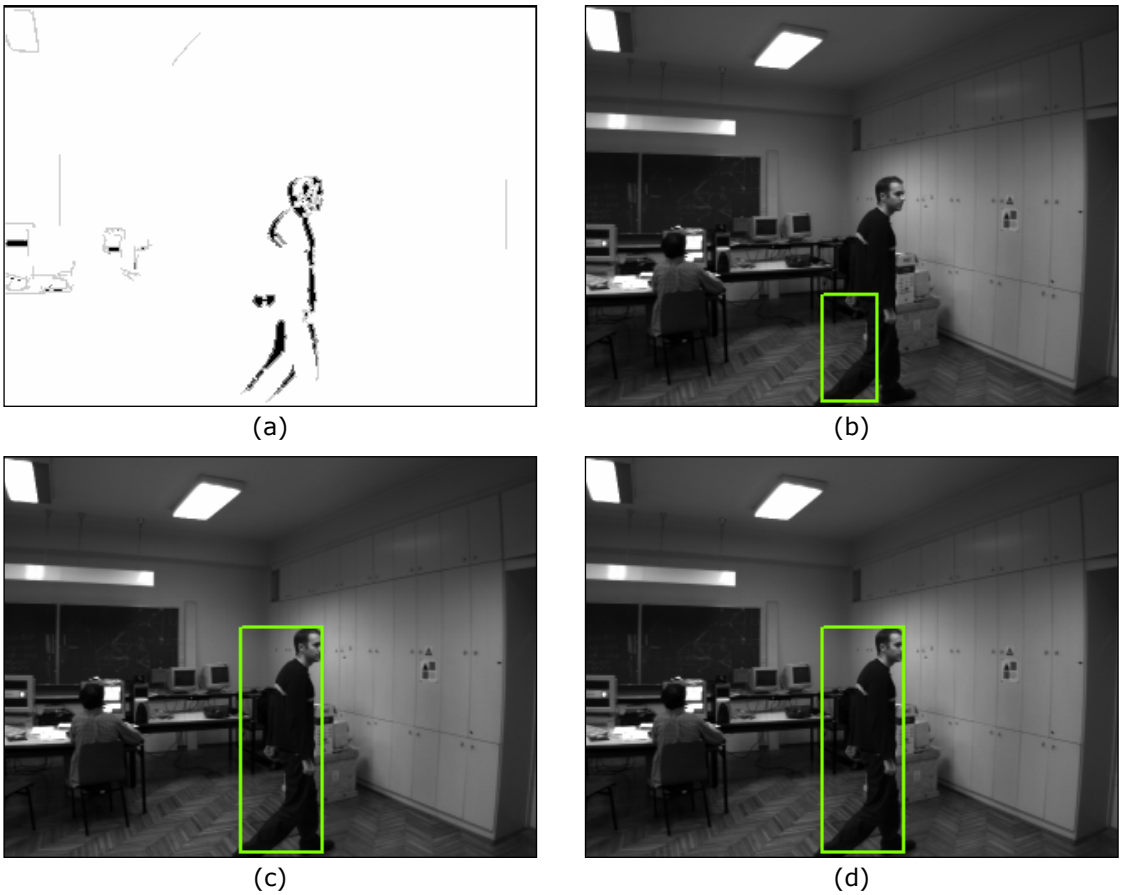
Postavljanjem ovog parametra na nisku vrijednost (blizu 0,0) sustav neće reagirati na gotovo nikakve promjene - čak i one na koje treba, dok postavljanjem na visoku vrijednost (blizu 1,0) sustav reagira na gotovo sve promjene - čak i one pogrešne. Podešavanjem ovog parametra želi se postići što veća osjetljivost na promjene uzrokovane pokretnim objektom i što manja osjetljivost na promjene koje su rezultat pogreške.

Uspjeh pojedinog eksperimenta se ocijenjuje brojem okvira sekvence u kojima je uspješno određena pozicija objekta. Tijekom eksperimenta ostali parametri imaju slijedeće vrijednosti: t=40, τ=15, N=20, d=40.

Tablica 6.3: Broj okvira s uspješno lociranim objektom u ovisnosti o parametru *r*.

r	0,5	0,6	0,7	0,8
broj uspješih/ ukupan broj	73/135	98/135	106/135	109/135

Tablica 6.3 pokazuje ovisnost broja okvira u kojima je postupak određivanja pozicije objekta bio uspješan o parametru *r*. Primjeri lociranja objekta u jednom okviru za različite vrijednosti parametra *r* prikazani su slikom 6.12.



(a) (b) (c) (d)

Slika 6.12: Primjer ovisnosti lociranja objekta o parametru *r*.
Prikazana je označena diferencija slika za okvir #25 sekvence (a) i pronađeni opisani pravokutnik objekta za vrijednosti parametara r=0,5 (b), r=0,6 (c), r=0,7 (d).

Sudeći po rezultatima eksperimenata prikazanim tablicom 6.3, sustav bolje radi kada je prag *r* visok, odnosno kada je osjetljiviji na aktivne regije iz razloga što se manje ispravnih aktivnih regija zanemaruje. Ipak ovaj podatak treba uzeti s rezervom pošto je u ovoj sekvenci, za vrijednost praga t = 40 (tablica 6.2), jako

malen broj okvira u za koje označena diferencija slika sadrži velik broj pogrešnih aktivnih regija.

6.6.3 Utjecaj parametra *d*

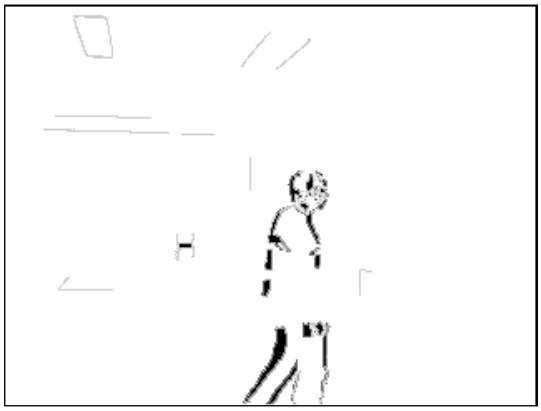
Parametar d, prag udaljenosti kod grupiranja aktivnih regija uvelike ovisi o dimenzijama praćenih objekata i njihovom načinu kretanja (kruto ili ne). Ovdje će se ispitivati ovisnost uspješnosti lociranja objekta o vrijednosti parametra *d* za ispitnu sekvencu. Za drugačije pokretne objekte u pravilu bi trebalo odabrati druge vrijednosti praga.

U eksperimentima se ocjenjuje utjecaj pojedinih vrijednosti praga *d* na uspješnost pronalaženja opisanog pravokutnika praćenog objekta. Pri tome se ostali parametri drže na vrijednostima: t=40, r=0,7, τ=15, N=20.

Tablica 6.4 pokazuje ovisnost broja okvira u kojima je postupak određivanja pozicije objekta bio uspješan o parametru *d*. Utjecaj praga *d* na uspješnost grupiranja, tj. određivanja pozicije pokretnog objekta, pokazan je na primjeru dva okvira (slika 6.13).

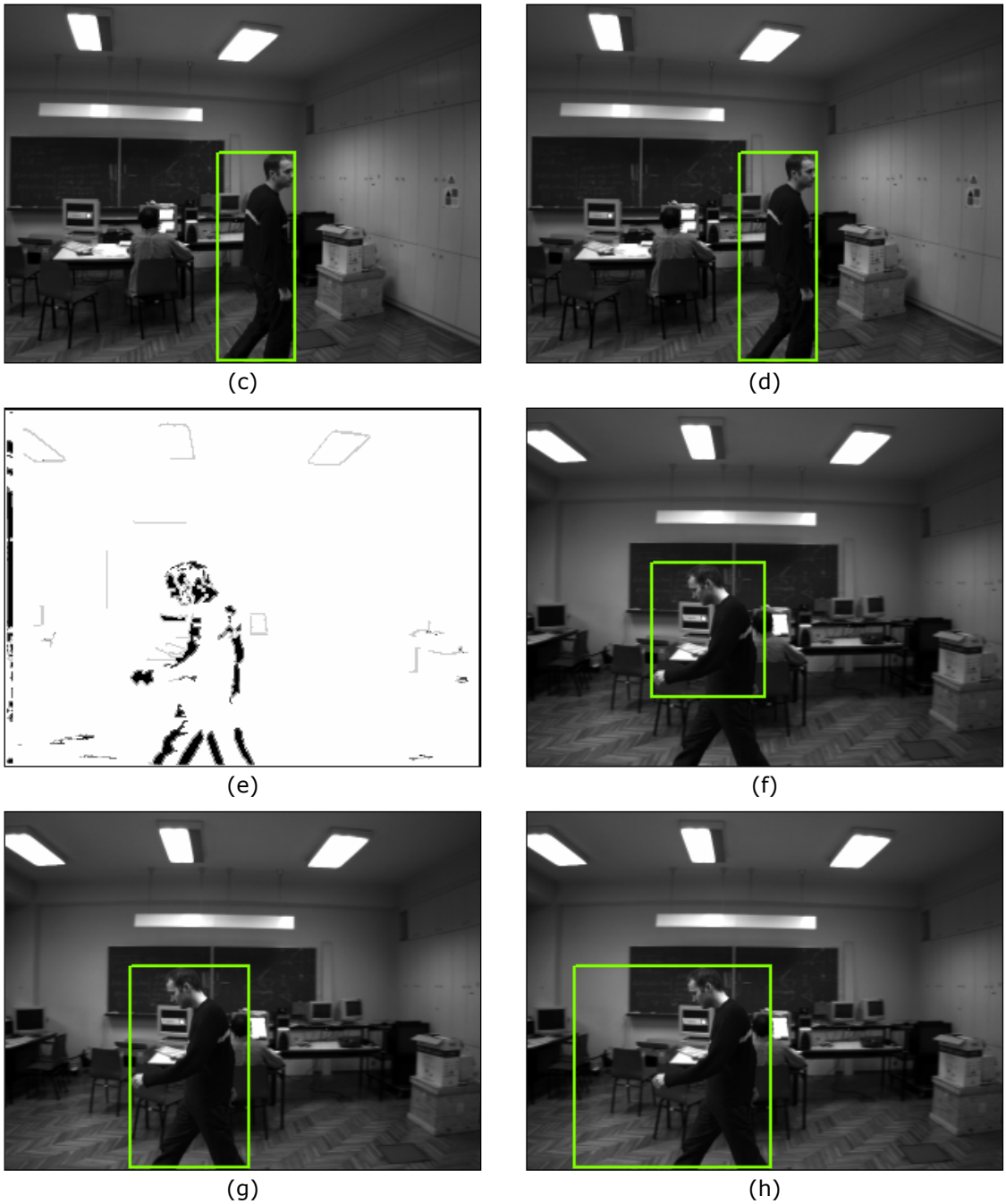
Tablica 6.4: Broj okvira s uspješno lociranim objektom u ovisnosti o parametru *d*.

d	20	30	40	50
broj uspješih/ ukupan broj	85/135	105/135	106/135	96/135



(a)

(b)



(c) (d) (e) (f) (g) (h)

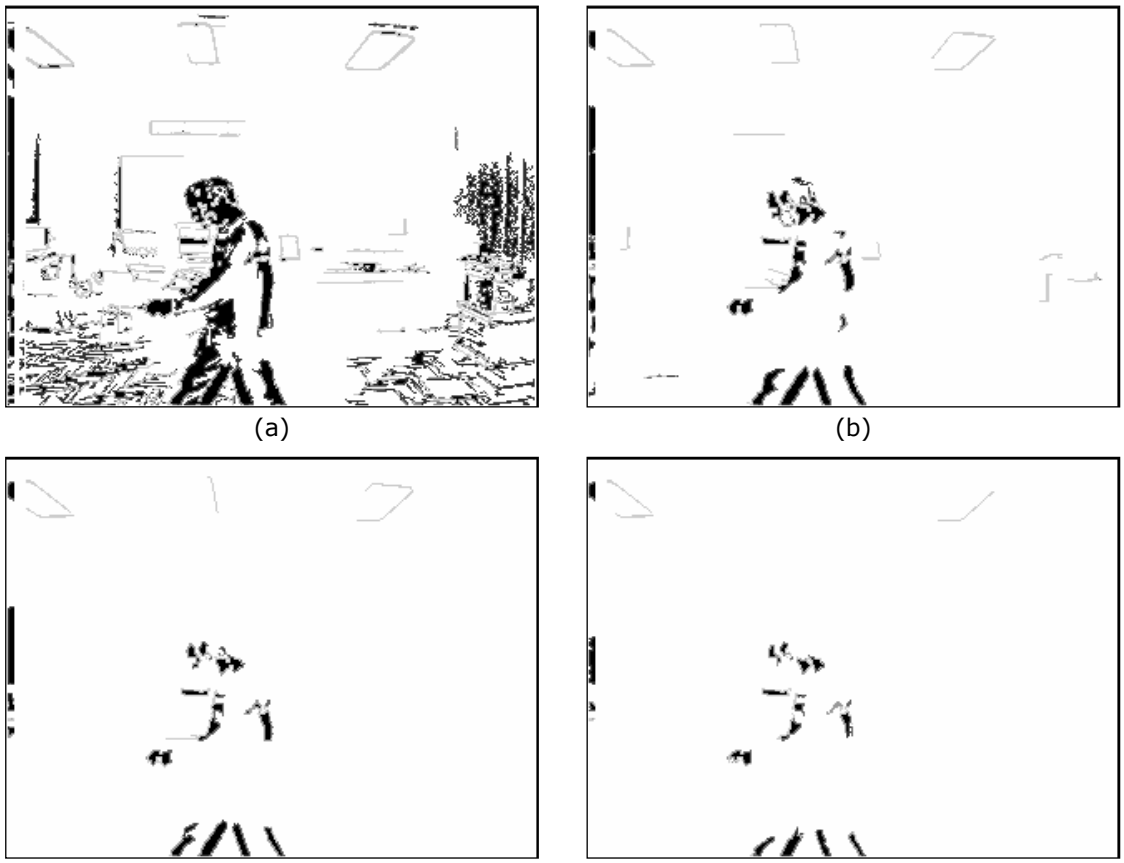
Slika 6.13: Uspješnost grupiranja u ovisnosti o parametru *d* na primjeru dva okvira. Prikazana je označena diferencija slika za okvir #14 (a) i okvir #14 s označenim objektu opisanim pravokutnikom za *d*=20 (b), *d*=30 (c) i *d*=50 (d). U ovom slučaju *d*=20 nije dovoljno velik, dok *d*=30 već je. Također je prikazana označena diferencija slika za okvir #118 (e) i okvir #118 s označenim objektu opisanim pravokutnikom za *d*=20 (b), *d*=30 (c) i *d*=40 (d). U ovom slučaju *d*=20 nije dovoljno velik, *d*=30 daje dobre rezultate, dok je već *d*=40 prevelik.

Iz dobivenih rezultata se vidi da je (za ovu sekvencu!) najbolja vrijednost praga *d*=40.

6.6.4 Utjecaj parametra τ

Parametar τ, prag diferencije slika, određuje što će se sve detektirati kao promjena, tj. određuje osjetljivost sustava na promjene u sceni.

Slika 6.14 prikazuje izgled označene diferencije slika za različite vrijednosti praga τ, uz vrijednost parametra N=20.



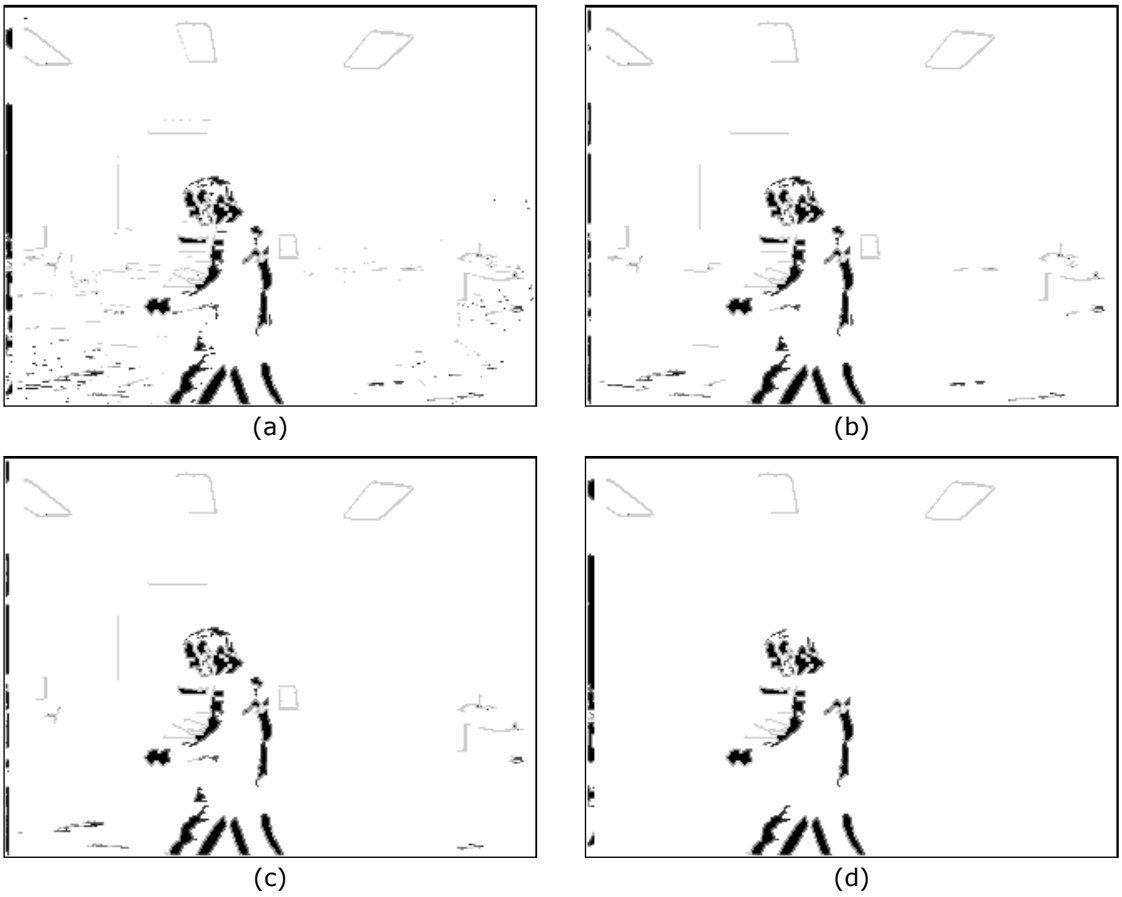
Slika 6.14: Izgled označene diferencije slika za različite vrijednosti praga τ.
Prikazane su označene diferencije slika okvira #118 ispitne sekvence za vrijednosti praga τ=5 (a), τ=20 (b), τ=35 (c), τ=50 (d).

Iz slike 6.14 se vidi da se za veće vrijednosti parametra τ detektira manji broj pogrešnih aktivnih regija, ali se zato i smanjuje osjetljivost na praćeni objekt. Zbog toga bi trebalo ovaj prag odabrati tako da se ne detektira previše promjena (kao na slici 6.14a), ali ni premalo (kao na slici 6.14c i 6.14d). Naravno, odabir vrijednosti praga trebao bi ovisiti o uvjetima u sceni, osvjetljenju i kontrastu objekta u odnosu na pozadinu. Za scenu prikazanu na ispitnoj sekvenci odabrana je vrijednost praga τ=15.

6.6.5 Utjecaj parametra *N*

Parametar *N*, prag korišten za filter veličine, određuje osjetljivost sustava na veličinu promjena u sceni.

Slika 6.15 prikazuje izgled označene diferencije slika za različite vrijednosti praga *N*, uz vrijednost parametra τ=15.



Slika 6.15: Izgled označene diferencije slika za različite vrijednosti praga *N*.
Prikazane su označene diferencije slika okvira #118 ispitne sekvence za vrijednosti praga N=1 (nije primijenjen filter veličine) (a), N=10 (b), N=20 (c), N=70 (d).

Iz slike 6.15a se vidi da je potrebno primijeniti filter veličine na diferenciju slika kako bi se eliminirao šum. Ipak, prag filtera veličine ne bi smio biti prevelik kako se ne bi smanjila osjetljivost sustava na pokretne objekte. Za scenu iz ispitne sekvence odabrana je vrijednost N=20.

6.6.6 Praćenje

Na temelju prethodnih eksperimenata određeni su parametri za koje se sustav ponaša najbolje na ispitnoj sekvenci: t=40, r=0,7, d=40, τ=15, N=20.

Slike 6.16 i 6.17 prikazuju rezultate praćenja za ispitnu sekvencu, u kojoj se prati čovjek u hodu, i za drugu sekvencu, u kojoj se prati čovjek u trku. Obje sekvence su snimljene tijekom rada sustava.


#15


#25


#35


#45


#55


#65

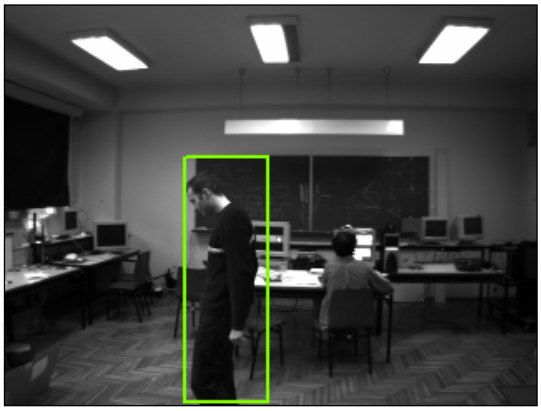

#75


#85

#95

#105

#115



#125

Slika 6.16: Rezultati praćenja čovjeka u hodu.
Prikazano je dvanaest okvira sekvence snimljene tokom praćenja čovjeka u hodu s označenim opisanim pravokutnikom praćenog objekta. Iz njih se vidi kako sustav uspješno drži objekt u blizini središta vidnog polja. Premda pozicija objekta nije uvijek točno određena (okviri #65 i #105) sustav ipak uspješno prati.

#25

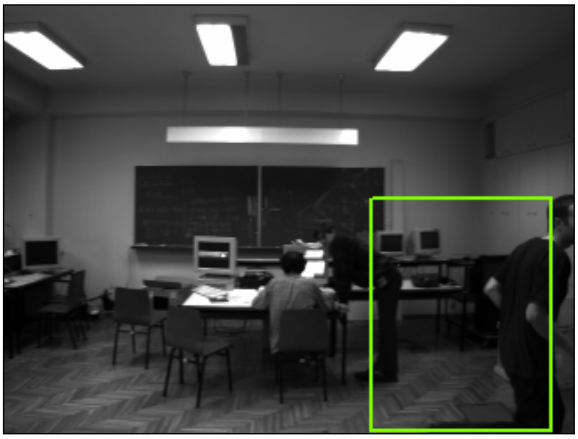
#35

#45

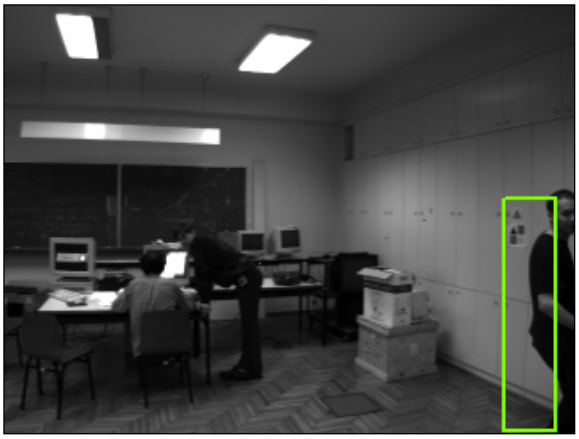
#55

#65

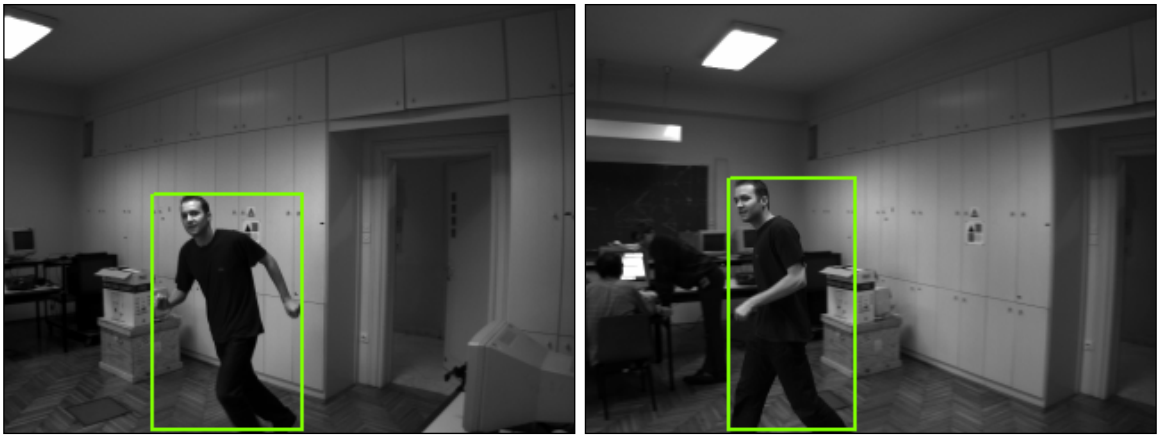
#75



#85



#95



Slika 6.17: Rezultati praćenja čovjeka u trku.
Prikazano je dvanaest okvira sekvence snimljene tokom praćenja čovjeka u trku s označenim opisanim pravokutnikom praćenog objekta. Iz njih se vidi kako sustav uspješno drži objekt u blizini središta vidnog polja. Vidi se i da pozicija objekta nije uvijek točno određena (okviri #45, #55 i #105) te da je objekt, zbog velike brzine kretanja, skoro pobjegao iz vidnog polja kamere (okviri #85 #95). Ipak, može se reći da sustav uspješno prati objekt.

6.7 Rezultati

SAPPO je testiran praćenjem ljudi u pokretu.

Opisana implementacija sustava omogućuje propusnost od oko 20 okvira u sekundi, ovisno o količini komunikacije s upravljivim postoljem. Valja napomenti da bi se, uz trenutnu komunikaciju s upravljivim postoljem, dobila propusnost od 25 okvira po sekundi. Eksperimentalno je utvrđeno da maksimalna kutna brzina upravljivog postolja, uz koju je moguće postići dovoljno malen pomak između okvira da bi sustav uspješno radio, iznosi 36° u sekundi.

Sustav je u većini slučajeva uspješno pratio ljude u hodu i trku na udaljenosti od 1,5 do 3 metra od kamere koja je bila postavljena na visini od oko 1,6 metara. Nagle promjene smjera gibanja također su uspješno praćene.

7

Sustav za izlučivanje slike vozila iz scene - SIVS

Ovo poglavlje opisuje još jedan sustav razvijen u sklopu ovog rada. Izrada ovog sustava je vezana uz studentski projekt Ministarstva znanosti i tehnologije "Automatski unos i raspoznavanje registarskih tablica vozila".

Za razliku od sustava SAPPO, opisanog u poglavlju 6, SIVS (Sustav za Izlučivanje slike Vozila iz Scene) ima točno definirano područje primjene, vezano uz navedeni projekt. Zadatak ovog sustava je nadzor mjesta na kojem se očekuje prolazak i zaustavljanje motornih vozila (granični prijelaz, naplatne kućice i sl.) i pribavljanje slike zaustavljenog vozila. Poželjno je da uzeta slika obuhvaća samo vozilo i da je vozilo snimljeno što bolje (što bolja razlučivost i oštrina) kako bi se smanjila složenost daljnih faza obrade (izlučivanje slike registarske tablice) i povećala točnost prepoznavanja (prepoznavanje pojedinih znakova na tablici).

7.1 Opis sustava

U cilju izvršavanja definiranog zadatka, sustav mora izvršavati slijedeće podzadatke:

1. Detekcija vozila.
2. Određivanje pozicije vozila.
3. Snimanje vozila.

Sustav koji obavlja ove zadatke mogao bi biti izveden na različite načine. Jedan od načina je korištenje senzora (fotoosjetljivog, nagaznog ili nekog drugog) za detekciju vozila i kamere upravljane tim senzorom za snimanje vozila. Umjesto senzora može se koristiti kamera (kamera je također senzor) i računalo, te metodama računalnog vida detektirati vozilo, odrediti njegovu poziciju i snimiti ga. U izvedbi SIVS-a koristi se poslijednji pristup iz razloga što taj pristup omogućuje veću fleksibilnost i točnost snimanja. Još jedna prednost ovog pristupa jest smanjivanje složenosti daljnjih faza obrade, jer se korištenjem metoda analize

dinamičkih scena i aktivnog vida može na vrlo jednostavan način izdvojiti samo slika vozila što se pomoću senzorom upravljane kamere ne može izvesti.

Dakle, sustav koristi kameru i za detekciju i za snimanje. U sustavu razlikujemo dvije glavne faze rada:

1. Segmentacija i praćenje vozila.
2. Izlučivanje slike vozila.

Prva faza određuje poziciju zaustavljenog vozila u sceni, dok druga faza uzima sliku tog vozila.

Prilikom razvoja sustava, s obzirom na korištenje kamere za izlučivanje slike vozila, implementirane su tri različite konfiguracije sustava:

1. **Korištenje stacionarne kamere.** Iz slike dobivene kamerom "izrezuje" se samo slika zaustavljenog vozila. Ta slika vozila je općenito manje razlučivosti od ulazne slike.
2. **Korištenje upravljive kamere.** Kada se detektira zaustavljeno vozilo u sceni, parametri upravljanja kamerom (zakret - engl. *pan*, nagib - engl. *tilt* i zoom) se podešavaju tako da vozilo zauzima cijelu sliku.
3. **Korištenje dviju kamera.** Ova konfiguracija koristi jednu, statičnu, kameru za detekciju i lociranje zaustavljenog vozila, a drugu, upravljivu, samo za snimanje. Parametri upravljive kamere podešavaju se na isti način kao i u konfiguraciji koja koristi samo jednu upravljivu kameru.

Pošto sve tri konfiguracije koriste isti postupak segmentacije i praćenja vozila, opis pojedinih konfiguracija dan je u poglavlju koje opisuje fazu izlučivanja slike vozila.

7.2 Prva faza: segmentacija i praćenje vozila

Zadatak ove faze je segmentacija scene i praćenje vozila te detekcija zaustavljanja vozila kako bi se moglo prijeći na fazu izlučivanja slike vozila.

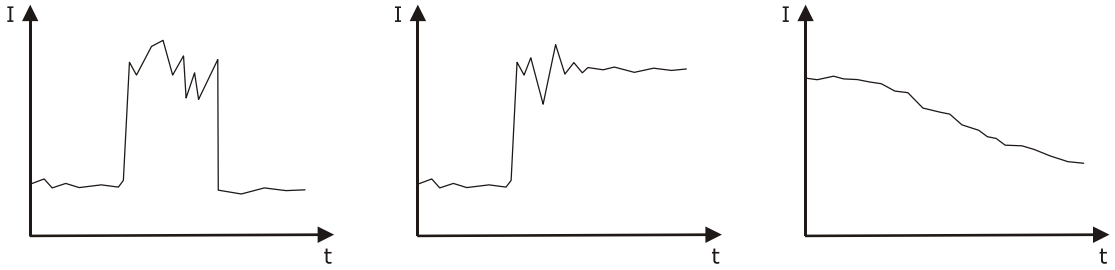
U nastavku slijedi podrobni opis korištenih postupaka detekcije promjene i segmentacije te praćenja.

7.2.1 Detekcija promjene i segmentacija

Temelj implementiranog postupka detekcije promjene i segmentacije scene predstavlja postupak opisan u radu [9]. Postupak nazvan *vremenski slojevi za adaptivno oduzimanje pozadine* (engl. *Temporal Layers for Adaptive Background Subtraction*) korišten je kao jedan od postupaka segmentacije pokretnih objekata u sustavu za videonadzor VSAM (od engl. *Video Surveilance And Monitoring*). Postupak se može svrstati u postupke temeljene na regijama, odnosno pojedinim slikovnim elementima (poglavlje 2). Postupak je u stanju prepoznati i zaustavljene objekte te je u stanju nositi se s ograničenim slučajem zaklanjanja objekata.

Ideja postupka je slijedeća. Promatramo slikovni element koji prolazi kroz različite promjene. U početku slikovni element odgovara pozadini. Ukoliko kroz taj slikovni element samo prođe neki objekt (ne zaustavi se na njemu), njegova vrijednost će se naglo promijeniti (naravno, ukoliko objekt ima različite sive vrijednosti od pozadine), a zatim malo varirati, nakon čega će se vratiti na početnu vrijednost, vrijednost koja odgovara pozadini (slika 7.1a). Ako objekt zakloni pozadinu na mjestu tog slikovnog elementa i zaustavi se, vrijednost tog slikovnog elementa naglo će se promijeniti, neko vrijeme varirati i stabilizirati na promijenjenoj vrijednosti koja odgovara objektu (slika 7.1b). Postepene promjene osvjetljenja u

sceni (npr. zakrivanje sunca za oblak) izazvat će postepene, glatke promjene vrijednosti slikovnog elementa (slika 7.1c).



(a) (b) (c)

Slika 7.1: Promjene intenziteta slikovnog elementa za uobičajene događaje. Objekt koji prolazi kroz slikovni element uzrokuje skokovitu promjenu u intezitetu slikovnog elementa, da bi se nakon prolaska objekta vratio na početnu vrijednost (a). Ukoliko se objekt zaustavi na mjestu promatranog slikovnog elementa, intenzitet će se stabilizirati na vrijednost koja odgovara objektu (b). Promjene osvjetljenja u sceni uzrokuju postepene, glatke promjene intenziteta slikovnog elementa (c).

Na temelju tih promatranja vidimo da slikovni element možemo svrstati u tri kategorije:

1. Pozadinski (engl. *background*) – slikovni element pripada pozadini.
2. Promjenjivi (engl. *transient*) – slikovni element prolazi kroz promjene uzrokovane prolaskom objekta.
3. Stacionarni (engl. *stationary*) – slikovni element odgovara objektu zaustavljenom iznad pozadine.

Za klasifikaciju slikovnog elementa u jednu od ove tri kategorije dovoljno je odrediti dva faktora: postojanje značajnog skoka u vrijednosti i stabilnost vrijednosti slikovnog elementa. Za određivanje ovih faktora potrebno je pratiti promjene vrijednosti slikovnog elementa kroz vrijeme. U tu svrhu čuvaju se vrijednosti svakog pojedinog slikovnog elementa *k* okvira u prošlost.

Za mjeru značajnosti skoka vrijednosti definira se tzv. *okidač pokreta* (engl. *motion trigger*) *T*:

$$T = \max\left\{\left|I_t - I_{(t-j)}\right|, \forall j \in \{1,\ldots,5\}\right\} \tag{7.1}$$

gdje I_t predstavlja vrijednost intenziteta nekog slikovnog elementa u trenutku *t*, *k* okvira u prošlosti. Okidač pokreta je u stvari maksimalna apsolutna razlika između vrijednosti intenziteta u trenutku *t* i vrijednosti intenziteta u 5 okvira koji prethode trenutku *t*.

Mjera stabilnosti (engl. *stability measure*) *S*, definira se za pojedini slikovni element, kao varijanca vrijednosti intenziteta od trenutka *t* do sadašnjosti:

$$S = \frac{k\sum_{j=0}^{k} I_{(t+j)}^2 - (\sum_{j=0}^{k} I_{(t+j)})^2}{k(k-1)} \tag{7.2}$$

U sustavu SIVS okidač pokreta i mjera stabilnosti definirani su nešto drugačije.

Okidač pokreta računa se kao maksimalna apsolutna razlika vrijednosti intenziteta slikovnog elementa trenutnog okvira I_c i vrijednosti intenziteta u prethodnih k okvira:

$$T = \max\left\{\left|I_c - I_{(c-j)}\right|, \forall j \in \{1,\ldots,k\}\right\} \tag{7.3}$$

Korištenjem ovako definiranog okidača pokreta dobivaju se jednaki rezultati kao i okidačem pokreta definiranim izrazom (7.1), uz korištenje manje memorije (pamti se 5 okvira manje).

Mjera stabilnosti računa se kao broj elemenata neprekinutog niza sličnosti krenuvši od vrijednosti intenziteta slikovnog elementa u trenutnom okviru prema vrijednosti k okvira u prošlost. Računanje mjere stabilnosti može se zornije prikazati slijedećom funkcijom u pseudokodu:

```
funkcija Izra•unajMjeruStabilnosti(red)
{
    ms = 0;
    za i = 1 do k
    {
       ako |red.zadnji() - red.predzadnji()| > prag
          prekini;
       ina•e
       {
          ms++;
          red.skiniZadnjeg();
       }
    }
    vrati ms;
}
```

Ovako definirana mjera stabilnosti dala je jednake rezultate kao i mjera stabilnosti definirana izrazom (7.2), uz puno brže računanje.

Korištenjem okidača pokreta i mjere stabilnosti i slike pozadine oblikuje se *mapa promjenjivosti* (engl. *transience map*). Mapa promjenjivosti je polje, jednakih dimenzija kao i ulazna slika, koje sadrži informacije o klasifikaciji svakog pojedinog slikovnog elementa u kategorije: pozadinski, promjenjivi, stacionarni. Formiranje mape promjenjivosti opisuje slijedeći algoritam (koji se izvodi za svaki pojedini slikovni element):

```
ako (M == stacionarni ili pozadinski) I T > prag_T
    M = promjenjivi;
ina•e
{
    ako M == promjenjivi I S > prag_S
    {
       ako vrijednost intenziteta == intenzitet pozadine
          M = pozadinski;
       ina•e
          M = stacionarni;
    }
}
```

Mapa promjenjivosti oblikuje se za svaki okvir, nakon čega se prelazi na izdvajanje regija koje odgovaraju krećućim objektima i objektima zaustavljenim iznad pozadine (slojevima - L). Regije se formiraju korištenjem sekvencijalnog pronalaženja povezanih komponenti [1] u varijanti 8-povezanosti. Prilikom formiranja u obzir se ne uzimaju regije s manje elemenata od nekog praga čime se

uklanjaju regije za koje se pretpostavlja da ne odgovaraju slici vozila. Nakon formiranja, svaka regija *R* se analizira u skladu sa slijedećim algoritmom:

```
ako R == promjenjiva
{// svi sl. elementi su ozna•eni kao promjenjivi
    R = kre•u•i objekt;
}
ina•e ako R == stacionarna
{// svi sl. elementi su ozna•eni kao stacionarni
    // ukloni sve sl. elemente dodijeljene nekom sloju
    R = R - (L[0] + L[1] + ... + L[j]);
    // ako je nešto ostalo stvori novi sloj
    ako R != 0
    {
      dodaj novi sloj L(j+1)=R;
      R = zaustavljeni objekt;
    }
}
ina•e
{// R sadr•i mješavinu promjenjivih i stacionarnih sl. elemenata
   formiraj regije na R - ((L[0] + L[1] + ... + L[j]);
   za svaku regiju SR dobivenu procesom formiranja
      ako SR == promjenjiva
         SR = kre•u•i objekt;
      ina•e ako SR == stacionarna
      {
         dodaj novi sloj L(j+1) = SR;
         SR = zaustavljeni objekt;
      }
      ina•e
         SR = kre•u•i objekt;
}
```

Za potrebe algoritma pojedini slojevi predstavljeni su kao i regije. Za svaku regiju se vode slijedeći podaci:

1. Opisani pravokutnik.
2. Položaj gornjeg lijevog ugla opisanog pravokutnika u slici
3. Polje koje svaki pojedini slikovni element unutar opisanog pravokutnika opisuje kao pozadinski (odnosno ne pripada regiji) ili kao stacionarni odnosno promjenjivi (pripada regiji).
4. Polje koje sadrži vrijednosti intenziteta pripadnih slikovnih elemenata.
5. Broj stacionarnih i promjenjivih elemenata.

Pomoću tako definiranih podataka jednostavno se vrše operacije sa pojedinim regijama koje su potrebne za opisani postupak analize regija.

Ovaj postupak svaki zaustavljeni objekt dodaje kao sloj iznad pozadine. Time se omogućuje nošenje s zaklanjanjem u ograničenom obliku - kada su zaklonjeni objekti zaustavljeni. Ilustracija rada postupka dana je u poglavlju 0 koje opisuje eksperimente provedene na sustavu.

7.2.2 Praćenje

Proces segmentacije daje regije koje odgovaraju pojedinim objektima (vozilima) u pojedinim okvirima. Da bi se moglo zaključivati o ulasku u scenu i izlasku iz scene svakog pojedinog vozila, potrebno ih je pratiti - određivati njihov položaj u svakom

okviru. Praćenje vozila u SIVS-u je nužno iz razloga što je potrebno uzeti sliku svakog vozila samo jednom, što bez praćenja nije izvedivo.

Prilikom praćenja vode se podaci o svakom objektu koji se pojavio u sceni. Za opisivanje objekata vode se isti podaci kao i za regije odnosno slojeve (opisani u prošlom poglavlju) prošireni podacima potrebnim za praćenje:

1. Identifikacijski broj objekta.
2. Broj okvira u kojima se objekt pojavio.
3. Broj okvira proteklih od prve pojave objekta.
4. Stanje objekta: ulazi, potpuno u sceni, izlazi.
5. Da li objekt zaklanja ili je zaklonjen.

Praćenje objekata provodi se uspoređivanjem regija koje se pojavljuju u trenutnom okviru s regijama koje odgovaraju objektima.

Da bi odredili koji objekt odgovara kojoj regiji trenutnog okvira potrebno je uvesti određenu mjeru sličnosti. U implementaciji SIVS-a korištena je jednostavna mjera sličnosti koja se temelji na sličnosti regijama opisanih pravokutnika i broja elemenata regija. Mjera sličnosti definirana je izrazom:

$$s = \frac{\min(n_1, n_2)}{\max(n_1, n_2)} \cdot \frac{\min(h_1, h_2)}{\max(h_1, h_2)} \cdot \frac{\min(w_1, w_2)}{\max(w_1, w_2)} \tag{7.4}$$

gdje su n_1, h_1 i w_1 broj elemenata, visina i širna prve regije, a n_2, h_2 i w_2 broj elemenata, visina i širna druge regije. Mjera sličnosti je broj između 0 i 1. Za regije koje nisu nimalo slične mjera sličnosti je 0, dok će za potpuno jednake regije biti 1. Ovako definirana mjera sličnosti brzo se računa i daje dobre rezultate, pa je stoga i zadržana.

Proces podudaranja regija za svaku regiju pronađenu u trenutnom okviru traži njenu sličnost sa regijom svakog objekta čije se središte (središte opisanog pravokutnika) nalazi unutar opisanog pravokutnika promatrane regije. Taj uvjet proizlazi iz činjenice da pri velikoj brzini pribavljanja okvira (od oko 25 okvira u sekundi) pomak vozila nije velik, pogotovo stoga što se očekuje sporo kretanje vozila i njihovo zaustavljanje. Kada se odrede sve međusobne sličnosti, odabire se objekt čija je sličnost s promatranom regijom najveća.

Kod podudaranja regija objekata koji ulaze u scenu ili izlaze iz scene (regije koje se nalaze tik uz rub slike) ne koristi se mjera sličnosti, već se samo promatra preklapanje regija, iz razloga što se prilikom ulaska ili izlaska veličine regija jako mijenjaju iz okvira u okvir.

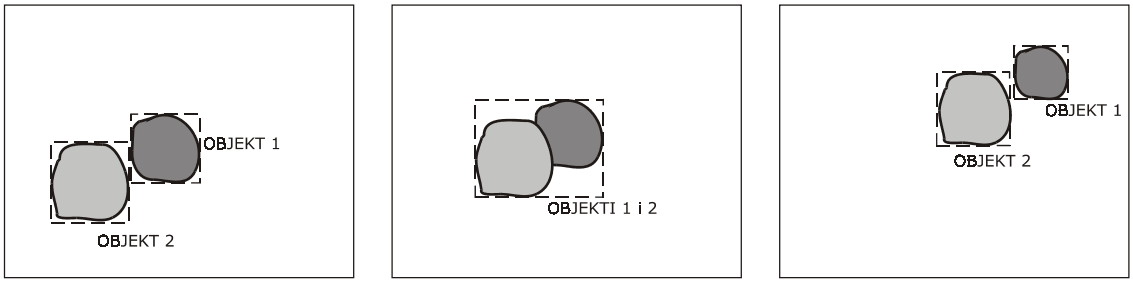
Postoje slučajevi kod kojih se ne uspijeva pronaći odgovarajući objekt za neku regiju trenutnog okvira. Do toga može doći iz više razloga:

1. Pojavio se novi objekt u sceni.
2. Došlo je do spajanja regija dvaju ili više objekata u jednu uslijed zaklanjanja (slučajevi zaklanjanja koji se ne mogu riješiti na prethodno opisani način).
3. Došlo je do razdvajanja regije na njih više uslijed prestanka zaklanjanja.

Ukoliko se pojavio novi objekt u sceni, on se jednostavno dodaje u listu praćenih objekata. Rješavanje spajanja i razdvajanja regija nešto je složenije.

Nakon inicijalnog procesa podudaranja, promatraju se objekti koji nisu upareni s niti jednom regijom u trenutnom okviru. Za svaki takav objekt (slika 7.2a) promatra se postoji li (neuparena) regija u trenutnom okviru koja je dovoljno velika da bi mogla sadržavati taj objekt i da li se objekt (u poslijednjoj detektiranoj

poziciji) preklapa s tom regijom. Ukoliko se nađe regija koja ispunjava te uvjete stvara se novi objekt, za taj objekt se označava da predstavlja grupu objekata koji se kreću zajedno, te se označava koji mu objekti pripadaju (slika 7.2b). Kada se takva grupa objekata razdvoji, u trenutnom okviru pojavit će se veći broj regija nego u prethodnom okviru. Za te regije se pretpostavlja da odgovaraju objektima prije zaklanjanja te se podudaranje provodi na temelju njihove udaljenosti od objekata prije zaklanjanja (slika 7.2c). Takvo podudaranje na temelju udaljenosti nije u potpunosti ispravno, no ako se promatraju vozila koja se kreću određenim redoslijedom, u redu, podudaranje će biti ispravno.



(a) (b) (c)

Slika 7.2: Shematski prikaz praćenja spajanja i razdvajanja regija objekata.
Prikazani su objekti prije zaklanjanja (a), objekti u toku zaklanjanja (b) i objekti nakon zaklanjanja (c).

Za određivanje pouzdanosti praćenja određenog objekta, odnosno za određivanje da li neka regija koja se pojavljuje stvarno odgovara objektu ili je samo posljedica šuma, koriste se dva prethodno navedena podatka - kroz koliko se okvira objekt pojavljivao te koliko je okvira prošlo od njegovog prvog pojavljivanja. Ukoliko je neka regija posljedica šuma ona će se pojaviti u jednom ili dva okvira nakon čega se više neće pojavljivati. Kada broj okvira koji je prošao od prvog pojavljivanja objekta postane puno veći od broja pojavljivanja objekta objekt se izbacuje iz liste.

Podatak o stanju objekta, tj. o tome da li objekt ulazi u scenu, da li je u potpunosti u sceni ili izlazi iz nje, koristi se kao dodatno osiguranje od detekcije nevaljanih objekata. Naime, za objekt se neće pretpostaviti de je vozilo ukoliko nije najprije ulazio u scenu te u potpunosti ušao u scenu. Podatak o izlasku iz scene koristi se za brisanje objekta iz liste praćenih objekata.

Kada je praćenjem utvrđeno da je objekt ušao u scenu, da se potpuno nalazi u sceni, te da se zaustavio, prelazi se na slijedeću fazu koja uzima sliku tog objekta, za kojeg se pretpostavlja da je vozilo. Zaustavljanje vozila detektira se na temelju tipa regije koja predstavlja taj objekt - ako se objekt zaustavio, odgovarajuća regija biti će stacionarna, odnosno predstavljat će sloj iznad pozadine.

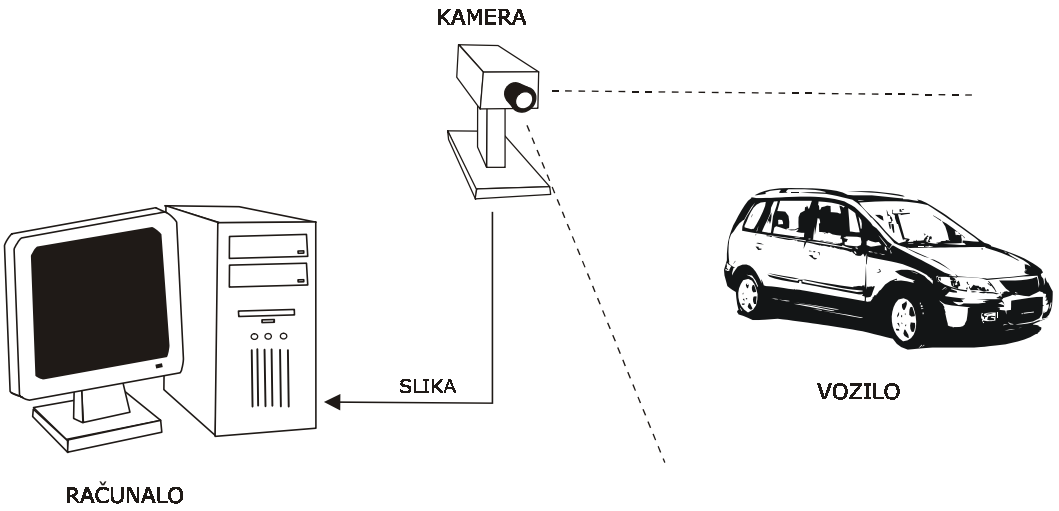
7.3 Druga faza: izlučivanje slike vozila

Ulaz u ovu fazu rada sustava su pozicija i veličina objekta dobivene u fazi segmentacije i praćenja. Na temelju tih podataka moguće je iz scene jednostavno izlučiti sliku vozila. U SIVS-u ova je faza napravljena na tri različita načina: korištenjem stacionarne kamere, korištenjem upravljive kamere i korištenjem dviju kamera.

7.3.1 Korištenje stacionarne kamere

Prikaz konfiguracije dan je slikom 7.3. Sustav se sastoji od fiksirane kamere i računala koje obrađuje sliku dobivenu iz kamere. Kamera je postavljena tako da promatra scenu u kojoj se očekuje nailazak i zaustavljanje vozila. Na temelju opisanog pravokutnika objekta dobivenog iz faze segmentacije i praćenja, jednostavno se izdvaja ("izrezuje") taj pravokutnik iz slike scene. Ta "izrezana" slika sadrži samo sliku vozila koja se tada može predati nekom modulu za prepoznavanje.

Ovakav postupak izlučivanja slike najjednostavniji je od svih triju korištenih postupaka. Bez obzira na jednostavnost može se uspješno koristiti, ukoliko je razlučivost kamere dovoljno velika (dovoljna bi bila oko 640×480), za dobivanje slike objekta s čitkom registarskom tablicom.



Slika 7.3: Konfiguracija SIVS-a sa stacionarnom kamerom.

7.3.2 Korištenje upravljive kamere

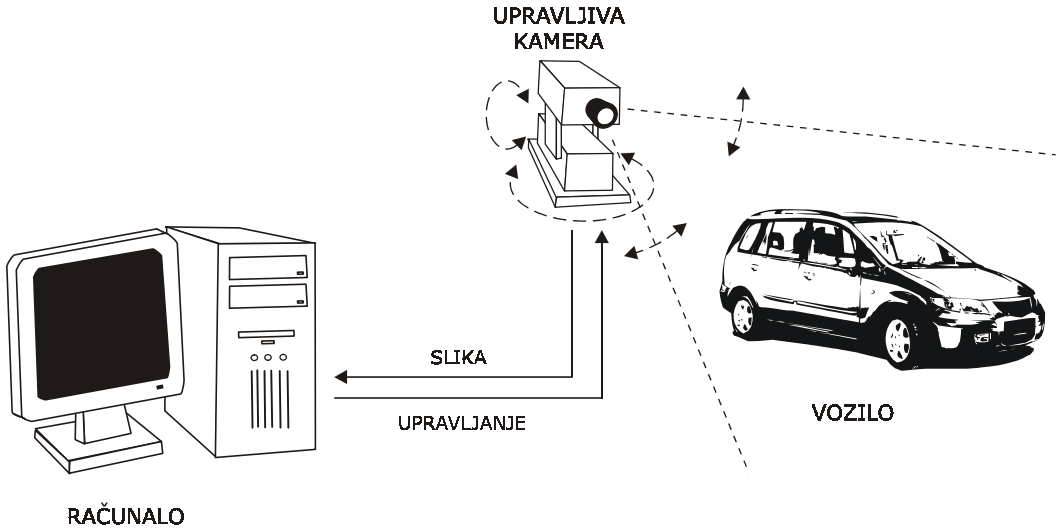
U cilju dobivanja što bolje slike objekta - vozila može se koristiti upravljiva kamera. Pri tome se misli na kameru kojoj se mogu podešavati kut zakreta, kut nagiba i kut gledanja (engl. *pan-tilt-zoom camera*). Konfiguracija sustava koja koristi upravljivu kameru dana je slikom 7.4.

Način izlučivanja slike vozila je slijedeći:

1. Računalo na temelju slike dobivene iz statične kamere (u početnom položaju) segmentira scenu i prati objekte.
2. Kada se detektira zaustavljeno vozilo proces segmentacije se zaustavlja.
3. Računalo izdaje naredbe upravljivoj kameri kako bi objekt postavila u središte svog vidnog polja i smanjila širinu vidnog polja (zumirala) tako da se u slici nalazi samo vozilo.
4. Uzima se slika vozila, kamera se vraća natrag u početni položaj te se nastavlja s postupkom segmentacije i praćenja.

Tako izlučena slika sadrži samo sliku vozila u punoj razlučivosti kamere te je stoga kvalitetnija od slike dobivene postupkom koji koristi stacionarnu kameru.

Upravljanje zakretom, nagibom i kutom gledanja kamere spada u područje aktivnog vida. Upravljanje zakretom i nagibom svrstava se u postupke upravljanja pogledom, tj. promjene pogleda. Premda se ne koristi fovealni senzor, upravljanje širinom vidnog polja kamere može se svrstati u fovealni vid (poglavlje 5.2) jer se određeni dijelovi scene, područja interesa, promatraju s većom razlučivosti od drugih (kada se smanji širina vidnog polja).



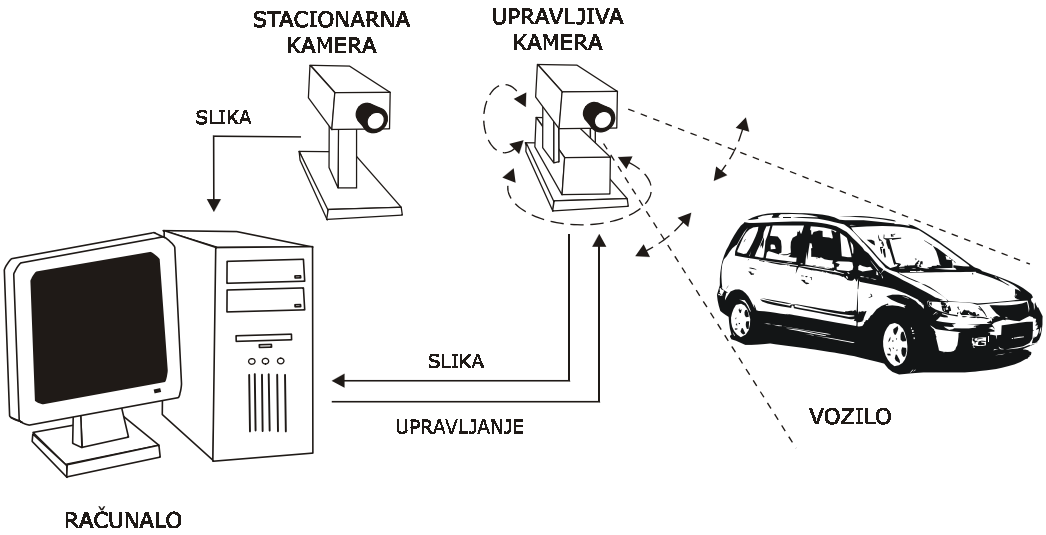
Slika 7.4: Konfiguracija SIVS-a sa upravljivom kamerom.

7.3.3 Korištenje dviju kamera

Postupak izlučivanja koristi dvije kamere, jednu statičnu, sa širokim kutom gledanja, i jednu upravljivu (slika 7.5). Ovaj postupak je u principu kombinacija postupaka koji koriste stacionarnu i upravljivu kameru. Na temelju slike dobivene iz stacionarne kamere kontinuirano se provodi segmentacija i praćenje, a kada se detektira zaustavljanje vozila, drugoj kameri se izdaju naredbe za postavljanje vozila u središte vidnog polja i podešavanje kuta gledanja tako da se samo vozilo nalazi u vidnom polju kamere.

Razlog uvođenja ovog postupka leži u sporosti smanjivanja kuta gledanja upravljive kamere. Kako se kod postupka koji koristi upravljivu kameru proces praćenja odvija samo kada se kamera nalazi u početnom položaju, potrebno je što prije uzeti sliku vozila i vratiti se u početni položaj kako bi se nastavilo praćenje. U implementaciji sustava (poglavlje 7.4) korištena je kamera kojoj za postavljanje objekta u središte vidnog polja, smanjivanje kuta gledanja te vraćanje u početni položaj treba nešto više od 5 sekundi. Za to vrijeme u sceni se mogu događati svakojake promjene koje bi mogle uzrokovati neispravan rad sustava.

U općem slučaju, korištenje dviju kamera zahtijeva određenu kalibraciju, odnosno određivanje transformacije iz koordinatnog sustava jedne kamere u koordinatni sustav druge kamere. Ipak, implementirani postupak nije zahtijevao gotovo nikakav postupak kalibracije iz razloga što su kamere postavljene jedna uz drugu čime se dobiva jako malena pogreška. Kalibriran je samo početni položaj kamera.



Slika 7.5: Konfiguracija SIVS-a sa dvije kamere.

7.4 Opis implementacije sustava

SIVS je implementiran korištenjem ljuske za računalni vid CVSH. Svi postupci isprogramirani su u programskom jeziku *C++*. Tablica 7.1 daje pregled korištene sklopovske i programske opreme.

U sve tri konfiguracije sustava korištena je ista oprema (tablica 7.1), s tim da su u implementaciji sustava koji koristi dvije kamere korištene dvije jednake kamere *Sony EVI-D31*.

Tablica 7.1: Sklopovska i programska oprema korištena u implementaciji SIVS-a.

tip komponente	komponenta	opis
kamera	*Sony EVI-D31*	analogna kamera sa video izlazom po PAL standardu
upravljivo postolje	integrirano s kamerom	RS-232, 9600 bps, protokol VISCA
pribavljanje slike	*Imagenation PXC 200 AL*	PCI kartica, digitalizator PAL signala, posebno sučelje
računalo	*Pentium 4* @ 2 GHz	
operacijski sustav	Windows 2000	
C++ prevodioc	MSVC v6.0	

Kao implementacijski detalj može se spomenuti i to da je sva obrada (segmentacija i praćenje) rađena na programski smanjenoj ulaznoj slici kako bi se povećala brzina obrade i smanjilo korištenje memorije. Ulazna slika veličine 320×240 smanjivala se,

po svakoj dimenziji za 2, na veličinu 160×120. Osim ulazne slike veličine 320×240 mogla se uzimati i slika veličine 640×480.

7.5 Eksperimenti

Vezano uz rad sustava, točnije, uz proces segmentacije, postoji nekoliko važnijih parametara:

1. prag okidača pokreta, *m*
2. prag mjere stabilnosti, s
3. prag razlike od pozadine, τ
4. prag veličine korišten za filter veličine, *N*
5. broj prošlih okvira koji se pamte, *q*.

Ovdje će biti opisani eksperimenti vezani uz ugađanje tih parametara. Također će biti provedeni i eksperiment vezan uz uspješnost rješavanja problema zaklanjanja objekata, te uspješnosti cjelokupnog procesa segmentacije, praćenja i izlučivanja slike vozila.

Eksperimenti vezani uz ugađanje parametara provode se na ispitnoj sekvenci koja prikazuje ulazak u scenu, zaustavljanje i izlazak iz scene jednog vozila. Eksperiment vezan uz rješavanje zaklanjanja proveden je korištenjem ispitne sekvence u kojoj se pojavljuju dva vozila koja se međusobno zaklanjaju. Obje sekvence snimljene su kamkorderom. Jedan eksperiment koji prikazuje uspješnost cjelokupnog procesa segmentacije, praćenja i izlučivanje slike vozila proveden je na ispitnoj sekvenci u kojoj se pojavljuju dva vozila koja se međusobno zaklanjaju, a drugi je proveden u laboratoriju korištenjem robota upravljanog putem radio veze.

7.5.1 Utjecaj parametra *m*

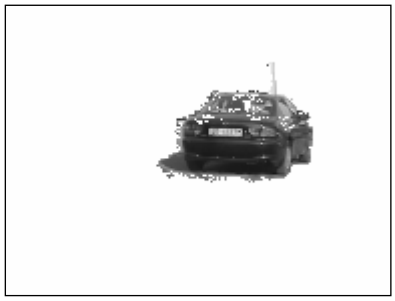
Utjecaj praga okidača pokreta *m* ispitan je promatranjem regija koje je sustav izdvojio kao objekte iznad pozadine. Ocjenjuje se uspješnost izdvajanja regija i osjetljivost na šum.

Eksperimenti su provedeni uz vrijednosti ostalih parametara: s=2, τ=20, N=120, q=5.

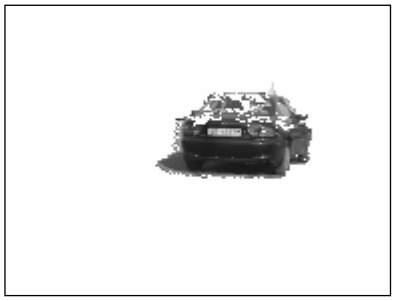
Slika 7.6 prikazuje regije izdvojene iz jednog okvira ispitne sekvence za različite vrijednosti parametra *m*. Iz nje se vidi da se za manje vrijednosti praga *m* regije koje odgovaraju vozilu bolje izdvajaju. Za prevelike (m=40) vrijednosti praga dobiva se više regija koje odgovaraju jednom objektu. Ipak, za preniske vrijednosti praga *m* dobiva se pretjerana osjetljivost na šum koja onemogućuje ispravan rad sustava. Primjer pretjerane osjetljivosti na šum (koji je u ovom slučaju posljedica automatske promjene fokusa kamkordera) dan je slikom 7.7.

Ispitivanjem rada sustava na cijeloj sekvenci određeno je da proces izdvajanja regija najbolje radi za vrijednosti parametra *m* oko 25. Ova vrijednost postavljena je dosta visoko zbog činjenice da je u ispitnoj sekvenci prisutno dosta šuma uzrokovanog automatskim fokusiranjem kamkordera kojim je snimana. Uz korištenje neke druge kamere vrijednost bi mogla biti i niža, čime bi se postiglo bolje izdvajanje regija koje odgovaraju objektima.

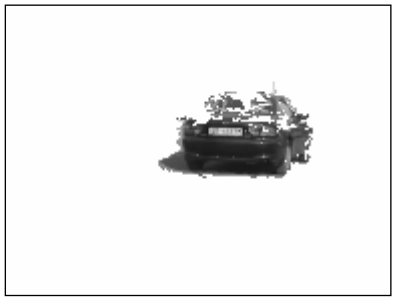
(a)



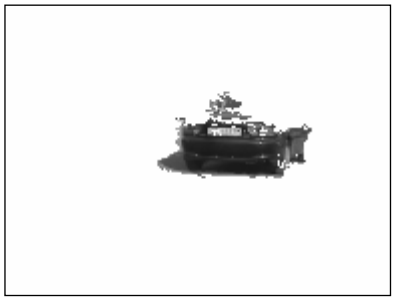
(b)



(c)



(d)



(e)

Slika 7.6: Izgled izdvojenih regija za različite vrijednosti parametra *m* na primjeru jednog okvira ispitne sekvence.
Prikazan je okvir #376 ispitne sekvence (a) i izdvojene regije za vrijednosti parametra *m*=5 (b), *m*=10 (c), *m*=25 (d) i *m*=40 (e).

(a)

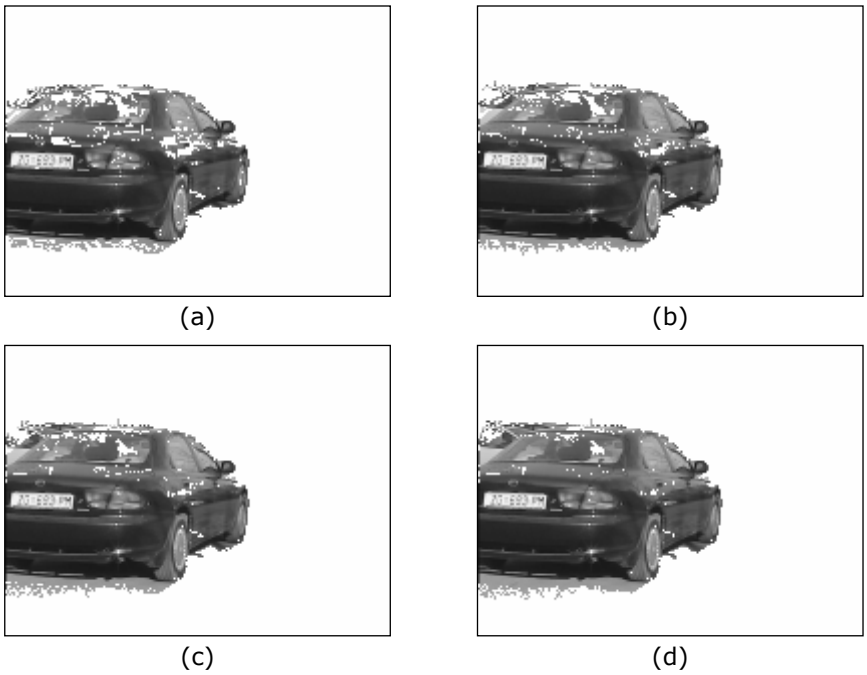
(b)

Slika 7.7: Primjer prevelike osjetljivosti sustava na šum za male vrijednosti parametra *m*.
Prikazane su izdvojene regije za okvir #169 ispitne sekvence za vrijednosti praga *m*=5 (a) i *m*=25 (b).

7.5.2 Utjecaj parametra *s*

Parametar *s* određuje koliko vremena (okvira) mora proći bez promjene vrijednosti nekog slikovnog elementa da se može reći da je ta vrijednost stabilna. Na taj način ovaj prag određuje vrijeme nakon kojeg se određeni slikovni element može svrstati u pozadinski ili stacionarni.

Eksperimenti su provođeni za različite vrijednosti parametra *s* uz vrijednosti ostalih parametara: m=25, τ=20, N=120, q=5.



Slika 7.8: Primjeri izdvojenih regija za različite vrijednosti parametra *s*.
Prikazane su regije izdvojene za okvir #208 ispitne sekvence za vrijednosti parametra s=1 (a), s=2 (b), s=3 (c), s=4 (d).

Slika 7.8 prikazuje utjecaj parametra *s* na izdvajanje regija. Za veće vrijednosti parametra *s* objekti ostavljaju za sobom dulji "trag" koji se uvrštava u njemu pripadnu regiju. Dulji "trag" za veće vrijednosti *s* poslijedica je činjenice da slikovni element mora biti stabilan kroz dulji period prije no što može postati pozadinski ili stacionarni.

Tablica 7.2: Ovisnost brzine detekcije zaustavljanja o parametru *s*.

s	1	2	3	4
broj okvira	7	18	42	64
Vrijeme (s)	0,23	0,60	1,40	2,13

Pojava "traga" nije od presudne važnosti za rad sustava, pošto je točnost određivanja pozicije vozila potrebna samo u trenutku zaustavljanja vozila. Značaj parametra *s* je u brzini i kvaliteti određivanja zaustavljanja objekta. Tablica 7.2 pokazuje utjecaj parametra *s* na brzinu detekcije zaustavljanja vozila. Iz tablice se

vidi da vrijeme detekcije progresivno raste s porastom vrijednosti parametra *s*. S obzirom na te podatke odabrana je vrijednost *s*=2 iz razloga što se postiže veća pouzdanost detekcije u relativno malom vremenu.

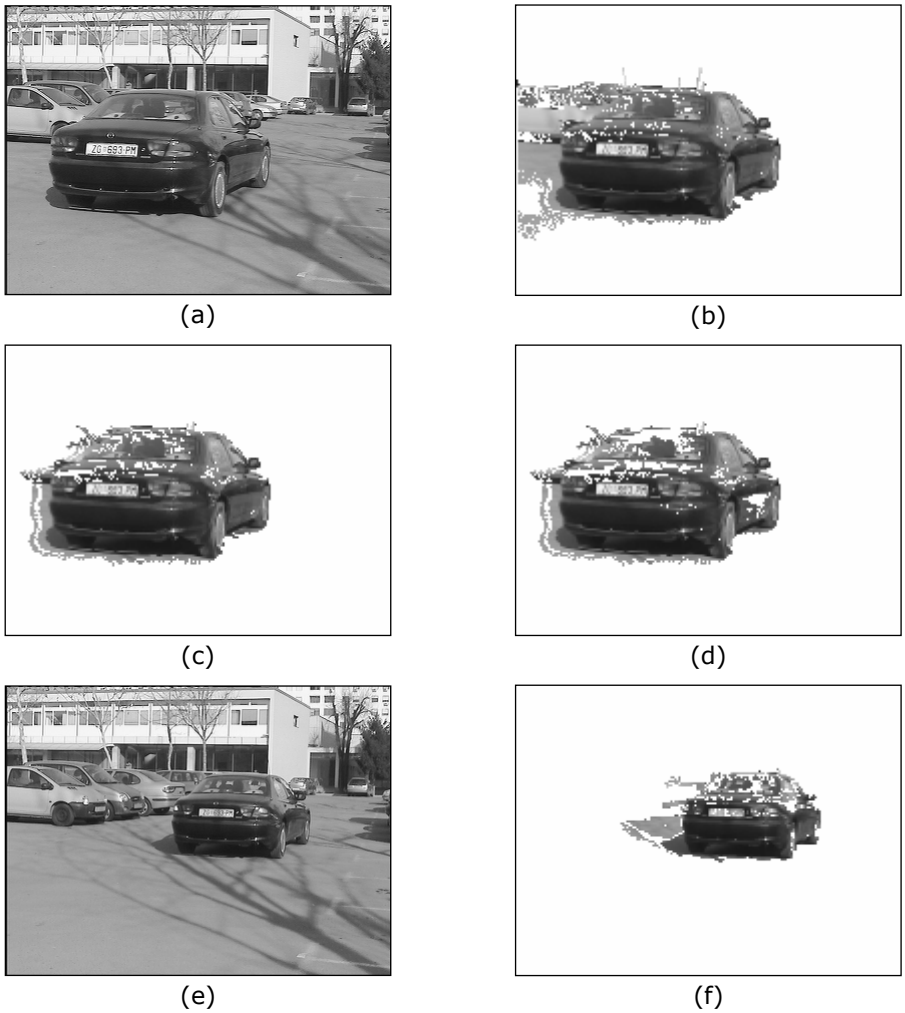
7.5.3 Utjecaj parametra τ

Parametar τ određuje koji će slikovni elementi, nakon promjene i stabilizacije, biti promatrani kao pozadinski, a koji kao stacionarni. Slično kao i parametar *m* ovaj parametar utječe na osjetljivost sustava na promjene, ali u odnosu na pozadinu.

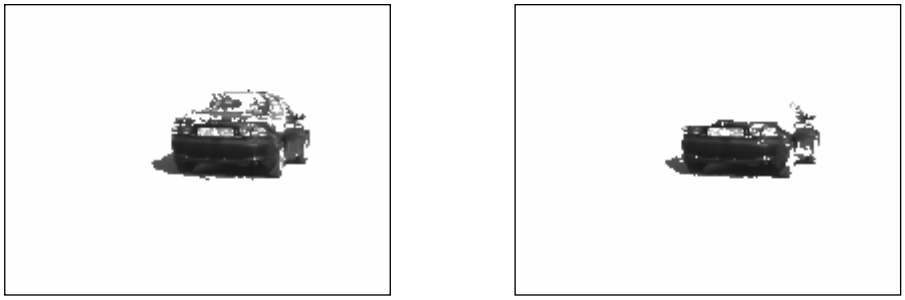
Eksperimenti su provođeni uz vrijednosti ostalih parametara: *m*=25, *s*=2, *N*=120, *q*=5. Promatrana je razina izdvajanja slikovnih elemenata koji ne pripadaju objektu iznad pozadine i razina neizdvajanja slikovnih elemenata koji pripadaju objektu.

Na slici 7.9 prikazani su rezultati eksperimenata. Vidi se da za manje vrijednosti praga τ dolazi do izdvajanja velikog broja slikovnih elemenata koji ne pripadaju objektu iznad pozadine, već su poslijedica šuma (uzrokovanog promjenjivim fokusom). Za veće vrijednosti praga τ javlja se veći broj neizdvojenih slikovnih elemenata koji pripadaju objektu iznad pozadine.

S obzirom na dobivene rezultate eksperimenta uzima se vrijednost praga τ=20.



(a) (b) (c) (d) (e) (f)



(g) (h)

Slika 7.9: Primjeri utjecaja parametra τ na izdvajanje regija.
Prikazan je okvir #223 ispitne sekvence (a) i rezultati izdvajanja regija za vrijednosti parametra τ=10 (b), τ=20 (c), τ=40 (d). Također je prikazan je okvir #293 (e) i rezultati izdvajanja regija za vrijednosti parametra τ=10 (f), τ=20 (g), τ=40 (h).

7.5.4 Utjecaj parametra *N*

Filter veličine se u postupku segmentacije koristi iz dva razloga. Jedan razlog je eliminacija regija nastalih pod utjecajem šuma, a drugi eliminacija regija koje su premale da bi odgovarale vozilima u sceni.

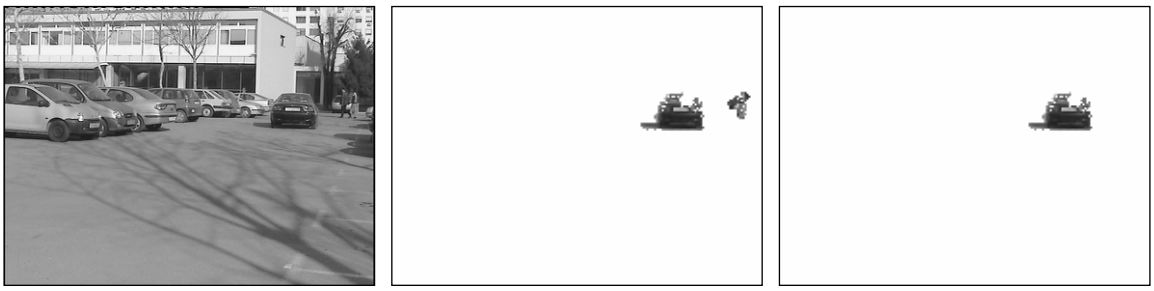
U ovom eksperimentu pokazuje se korištenje parametra *N*, praga veličine, u obje svrhe: eliminaciju šuma i eliminaciju regija koje ne odgovaraju vozilima, odnosno regija koje odgovaraju udaljenim vozilima koja ionako nisu od interesa. Vrijednosti ostalih parametara prilikom eksperimenta su: m=25, s=2, τ=20, q=5.

Slike 7.10 i 7.11 pokazuju rezultate eksperimenata. Pokazano je kako se primjenom filtera veličine može ukloniti šum i objekti koji nisu od interesa.

Ukoliko se želi samo ukloniti šum dovoljna je vrijednost parametra m oko 10. Za eliminaciju premalenih objekata parametar se postavlja ovisno o veličini objekata koji se prate.

(a) (b) (c)

Slika 7.10: Utjecaj parametra *N* na eliminaciju šuma.
Prikazan je okvir #213 ispitne sekvence (a) i izdvojene regije za vrijednosti parametra N=1 (b), N=20 (c).



(a) (b) (c)

Slika 7.11: Utjecaj parametra *N* na eliminaciju presitnih objekata.
Prikazan je okvir #472 ispitne sekvence (a) i izdvojene regije za vrijednosti parametra N=20 (b), N=120 (c).

7.5.5 Utjecaj parametra *q*

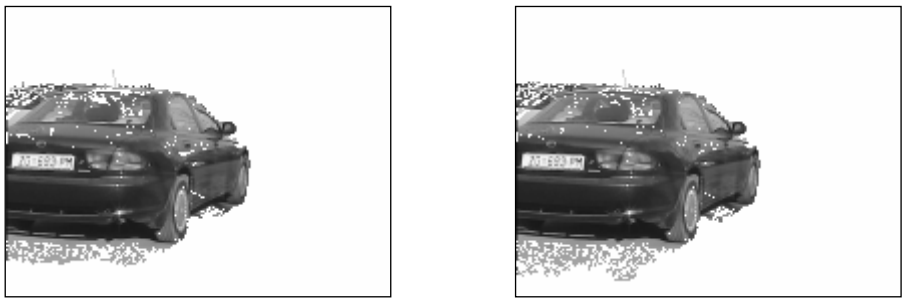
Parametar q određuje broj okvira is prošlosti koji se pamte. Ima sličan značaj kao i parametar s. Potrebno je da bude barem za 1 veći od s.

Ovdje će samo biti pokazan utjecaj parametra na duljinu "traga" koje vozilo ostavlja iza sebe. Vrijednosti ostalih parametara su: m=25, s=2, τ=20, N=120.

Slika 7.12 prikazuje izdvojene regije za dvije vrijednosti parametra q. Vidi se da "trag" vozila raste s porastom vrijednosti parametra q.

Pored utjecaja na proces segmentacije, parametar q jako utječe na povećanje korišene memorije i na brzinu obrade.

Pošto je ranije odabrana vrijednost s=2, q treba biti najmenje 3. Pošto nema potrebe da bude veća, odabrana je vrijednost 3.



(a) (b)

Slika 7.12: Primjer utjecaja parametra *q* na izdvajanje regija.
Prikazane su izdvojene regije za okvir #208 ispitne sekvence za vrijednosti parametra q=10 (a) i q=15 (b).

7.5.6 Rješavanje zaklanjanja

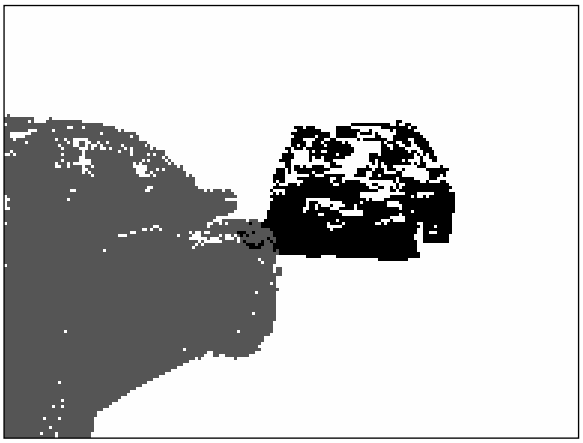
Ovdje će biti prikazano rješavanje zaklanjanja u ograničenom slučaju u kojem je ono moguće. Eksperiment je proveden uz vrijednosti parametara: m=25, s=2, τ=20, N=120, q=3.

Prikaz eksperimenta dan je slikom 7.13.

(a)

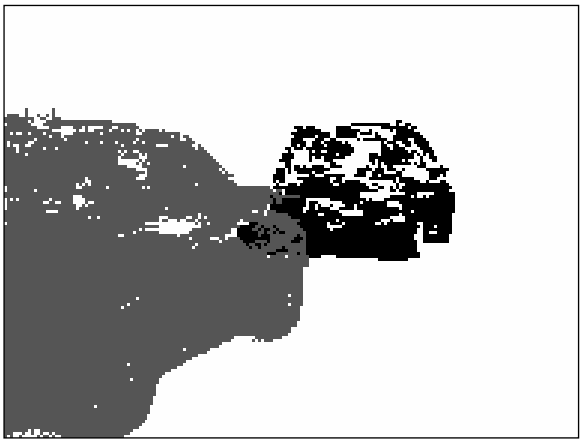
(b)

(c)



(d)

(e)



(f)

(g) (h)

Slika 7.13: Ilustracija rješavanja zaklanjanja.
Prikazana su četiri okvira sekvence u kojima dolazi do međusobnog zaklanjanja vozila i regije koje odgovaraju percepciji različitih objekata. Regije koje odgovaraju različitim objektima označene su različitim sivim razinama. Okvir #529 (a) i odgovarajuće regije (b): trenutak prije početka zaklanjanja. Okvir #541 (c) i odgovarajuće regije (d): drugi automobil zaklanja prvi automobil, ali se zaklanjanje uspješno rješava jer prvi automobil stoji. Okvir #550 (e) i odgovarajuće regije (f): zaklanjanje se i dalje uspješno rješava jer prvi automobil još uvijek stoji. Okvir #556 (g) i pripadajuće regije (h): pošto se i prvi automobil kreće, više nije moguće rješavati zaklanjanje na ovaj način.

7.5.7 Cjelokupni rad sustava

Ovdje će biti prikazani eksperimenti vezani uz cjelokupni rad sustava.

Eksperiment vezan uz konfiguraciju sustava koja koristi stacionarnu kameru proveden je na sekvenci koja sadrži dva automobila koji se međusobno zaklanjaju. Rezultati eksperimenta prikazani su slikom 7.14.

Eksperiment kojim je provjeravana ispravnost rada konfiguracije sustava s upravljivom kamerom proveden je u laboratoriju. Umjesto vozila korišteni su mobilni roboti upravljani radio vezom. Rezultati eksperimenta prikazani su slikom 7.15. Isti rezultati dobivaju se za konfiguraciju sustava koja koristi dvije kamere, jedina razlika je u brzini izlučivanja slike.

(a) (b)


[1]|complete

(c)


[1]|complete|stopped

(d)

(e)


[1]|complete|stopped

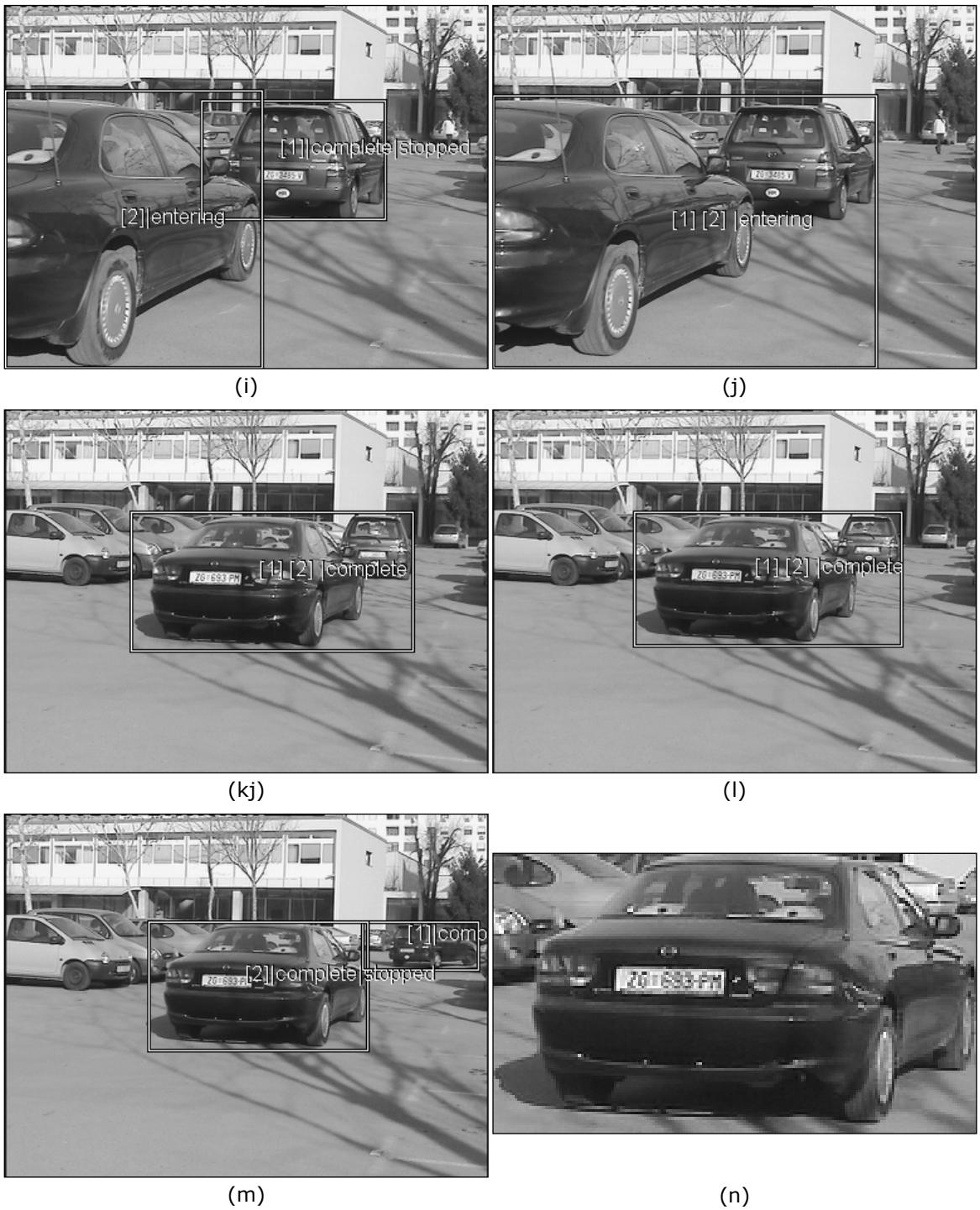
(f)


[1]|complete|stopped
[2]|entering

(g)


[1]|complete|stopped
[2]|entering

(h)



(i) (j)

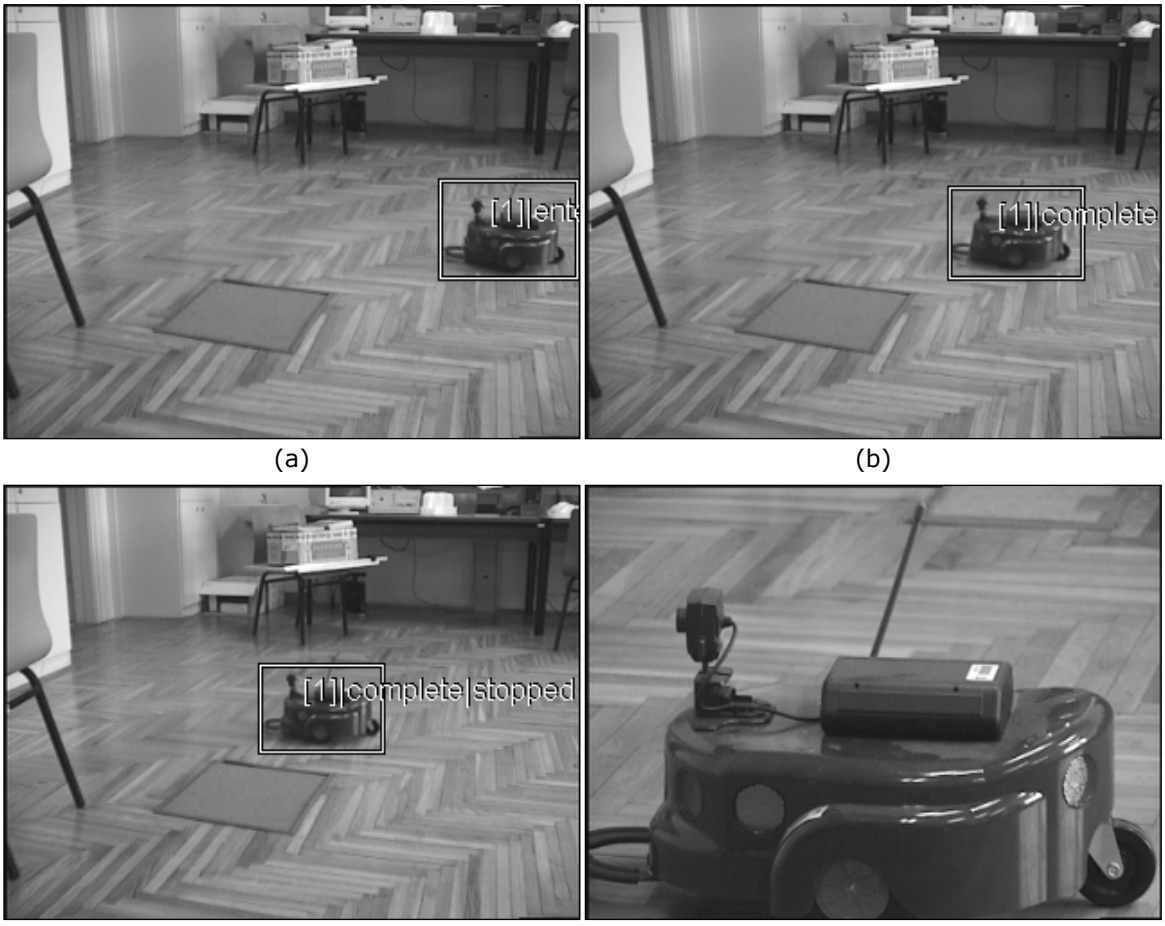
(kj) (l)

(m) (n)

Slika 7.14: Primjer rada SIVS-a za sekvencu iz stvarnog svijeta.

Prikazani su okviri sekvence koji odgovaraju značajnijim događanjima u sceni. Također su prikazane i izlučene slike vozila. Okvir #370 (a): prvi automobil ulazi u scenu. Okvir #380 (b): prvi automobil još uvijek ulazi u scenu, ali se počinje pratiti. Okvir #410 (c): prvi automobil je kompletan u sceni. Okvir #490 (d): prvi automobil se zaustavio te je izlučena njegova slika (e). Okvir #520 (f): prvi automobil još uvijek stoji, ulazi drugi automobil. Okvir #526 (g): prvi automobil stoji, drugi automobil ulazi i počinje se pratiti. Okvir #542 (h): dolazi do zaklanjanja prvog automobila od strane drugog, ali postupak se nosi sa

zaklanjanjem jer prvi automobil stoji. Okvir #550 (i): drugi automobil još više zaklanja prvi. Okvir #553 (j): pošto se prvi automobil počeo kretati, automobili se prate kao skupina jer se postupak više ne može nositi sa zaklanjanjem. Okvir 634 (k): automobili se još uvijek zaklanjaju i kreću zajedno. Okvir #652 (l): automobili se još uvijek zaklanjaju i kreću zajedno. Okvir #708 (m): automobili su se razdvojili, drugi automobil je zaustavljen pa se izlučuje njegova slika (n).



(a) (b)

(c) (d)

Slika 7.15: Primjer rada konfiguracije SIVS-a koja koristi upravljivu kameru.
Prikazana su tri okvira koji odgovaraju značajnijim događajima u sceni i izlučena slika vozila (robota). Okvir #93 (a): robot ulazi u scenu i prati ga se. Okvir #112 (b) robot se potpuno nalazi u sceni. Okvir #150 (c): robot se zaustavlja pa se izlučuje njegova slika (d).

7.6 Rezultati

Rad sustava ispitan je za sve tri konfiguracije. Prva konfiguracija, korištenje stacionarne kamere, ispitana je za scene s pravim vozilima na otvorenom, dok su druge dvije konfiguracije (korištenje upravljive kamere i korištenje dviju kamera) ispitane u laboratoriju.

Konfiguracija sustava koja koristi samo statičnu kameru ispitana je korištenjem sekvenci snimljenih kamkorderom na otvorenom. Snimani su automobili koji ulaze u scenu, zaustavljaju se, te izlaze iz scene. Na ovaj način ispitan je postupak

segmentacije i praćenja na primjerima iz stvarnog svijeta. Sustav se pokazao uspješnim. Premda je u tablici 7.1 kao korišteno računalo navedeno vrlo snažno računalo (*Pentium 4* @ 2 GHz), postupak obrade može raditi dovoljno brzo i na slabijem računalu. Postupak je isproban i na računalu *Pentium 3* @ 533 MHz, pri čemu se dobila propusnost od oko 15 okvira u sekundi što je dovoljno za stvarnu primjenu.

Konfiguracija koji koristi upravljivu kameru i konfiguracija koja koristi dvije kamere ispitane su u laboratoriju. Kao zamjena za vozila korišteni su mali roboti upravljani radio vezom (*amigobot*). Sustav se pokazao uspješnim i u ovoj konfiguraciji.

8

Zaključak

Ovaj rad se bavio ispitivanjem mogućnosti dobivanja informacija iz sekvence okvira uzetih iz dinamičke scene na temelju promjena u pojedinim slikovnim elementima i korištenja tih informacija u svrhu segmentacije i praćenja. Za promjene u slikovnim elementima se pretpostavlja da su rezultat kretanja objekata u sceni.

Na primjeru sustava za aktivno praćenje pokretnih objekata (SAPPO) pokazane su mogućnosti korištenja postupaka niske razine kao što su diferencija slika i označena diferencija slika. Premda ti postupci općenito nisu dovoljni za potpunu segmentaciju scene, u velikoj mjeri mogu pomoći u procesu segmentacije kao mehanizam određivanja područja interesa, područja na koja će se usmjeriti pozornost. Kombinacija ovih postupaka s postupkom određivanja pomaka pozadine na temelju računanja pomaka uglova u sceni uspješno je riješen problem segmentacije u slučaju pokretne kamere (MCMO). Na taj način dobiven je sustav koji aktivno prati, odnosno drži objekte u svom vidnom polju (stabilizacija pogleda u kontekstu aktivnog vida). Izgrađeni sustav uspješno prati objekte promjenjivog oblika kao što su ljudi i to u složenim scenama u kojima se nalazi mnoštvo predmeta, a time i mnoštvo rubova. U takvoj sceni, uobičajeni postupci segmentacije, temeljeni na gradijentu odnosno detekciji rubova, teško da bi dali zadovoljavajuće rezultate.

Poboljšanje SAPPO-a moguće je u procesu segmentacije. Korišteni postupci segmentacije nisu dovoljno precizni, odnosno ne određuju točnu poziciju praćenog objekta u svakom okviru, te bi ih trebalo koristiti u sprezi s nekim drugim postupcima. Vrijedilo bi istražiti primjenu postupaka temeljenih na aktivnim konturama. Područja promjene u sceni mogla bi poslužiti za inicijalizaciju aktivne konture koja bi tada mogla "fino" opisati konturu objekta. Kako bi se na taj način dobila precizna pozicija praćenog objekta, mogla bi se predviđati pozicija objekta u slijedećim okvirima te na takav način bolje pratiti objekt. Precizno određivanje pozicije objekta omogućilo bi i rješavanje problema koji su sprečavali praćenje u vertikalnom smjeru (upravljanje kutem nagiba kamere).

Primjena postupaka segmentacije na temelju pokreta, odnosno promjene u sceni, pokazana je u sustavu za izlučivanje slike vozila iz scene (SIVS). Kako je za zadatak ovog sustava važna što točnija segmentacija, korišten je postupak oduzimanja pozadine. Na taj način jednostavno se dobiva potpuna slika vozila (za

razliku od postupaka diferencije slika i detekcije vremenski promjenjivih rubova). Korišteni postupak omogućuje i rješavanje zaklanjanja objekata u ograničenom slučaju kada u zaklanjanju sudjeluju zaustavljeni objekti. Pošto korišteni postupak pokazuje probleme koji se javljaju kod postupaka oduzimanja pozadine, potrebno je definirati ograničenja na rad sustava, odnosno uvjete pod kojima će sustav ispravno raditi. Ti uvjeti su:

1. U trenutku pokretanja sustava u sceni se ne smiju nalaziti pokretni objekti.
2. Ne smije dolaziti do značajnijih promjena osvjetljenja scene.
3. Izvor osvjetljenja ne smije se nalaziti previše nisko, odnosno objekti ne smiju bacati jako duge sjene.
4. U prednjem i srednjem planu scene ne smiju se kretati ljudi i slični objekti.

Prva tri uvjeta rezultat su nedostataka postupka oduzimanja pozadine, dok treći uvjet proizlazi iz činjenice da sustav ne prepoznaje objekte kao automobile, već samo traži dovoljno velike regije, a ljudi u prednjem i srednjem planu su dovoljno veliki. Prvi uvjet nije teško ispuniti, pošto sustav uključuju ljudi koji mogu kontrolirati uvjete uključivanja. Drugi i treći uvjeti su u pravilu ispunjeni na mjestima koja su predviđena za rad sustava (npr. gotovo svi granični prijelazi su natkriveni). Četvrti uvjet može se ispuniti ograničavanjem kretanja ljudi ispred kamere, što u svakom slučaju mora biti ispunjeno kako ne bi zaklanjali vozila i onemogućavali dobivanje njihove potpune slike.

Naravno, poželjno je da postoje što manja ograničenja na rad sustava te bi se poboljšanja sustava trebala kretati u tom smjeru. Problemi postupka oduzimanja pozadine mogu se riješiti ispravnim modeliranjem pozadine. Uvođenjem procesa klasifikacije regija mogao bi se riješiti problem pojave ljudi u sceni. Postoje različiti načini za razlikovanje ljudi i vozila. Na primjer, postupak Selingera i Wixsona [26] omogućuje razlikovanje krutih objekata (kao što su vozila) i objekata promjenjivog oblika (kao što su ljudi) na temelju njihovog izgleda kroz više okvira, odnosno na temelju njihovog kretanja. Uvođenjem informacija o boji, odnosno korištenjem slike u boji umjesto sive slike, moguće je riješiti problem sjena.

U SIVS-u je korišten vrlo jednostavan postupak praćenja - na temelju veličine opisanog pravokutnika, broja elemenata i položaja regija koje odgovaraju objektima. Takav postupak praćenja može raditi dobro, ali samo u ograničenim uvjetima. Za imalo složenije scene potrebno je razviti složeniji postupak praćenja.

Upravljanjem, zakretom, nagibom i kutom gledanja kamere, postignuto je izlučivanje slike vozila u punoj razlučivosti, čime se povećava detaljnost slike vozila i omogućava pouzdano prepoznavanje. Premda nije korišten pravi fovealni senzor, takav način korištenja kamere može se svrstati u fovealni vid jer sustav u isto vrijeme (ili u gotovo isto vrijeme, kod konfiguracije koja koristi samo jednu kameru) ima široki uvid u scenu (u cijeloj razlučivosti široko vidnog polja) i detaljan uvid (u cijeloj razlučivosti usko vidno polje) u dio scene od interesa.

Literatura

[1] R. Jain, R. Kasturi, B.G. Shunck, *Machine Vision*, McGraw-Hill, Inc., 1995.

[2] R. Jain, "Dynamic Scene Analysis Using Pixel-Based Processes", *Computer* 12(1):12-18, 1981.

[3] R. Jain, "Extraction of Motion Information from Peripheral Processes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol PAMI-1, no. 2, April 1979, pp. 206-214.

[4] R. Jain, H.-H. Nagel, "On the Analysis of Accumulative Difference Pictures from Image Sequences of Real World Scenes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol PAMI-1, no. 2, April 1979, pp. 206-214

[5] C. Ridder, O. Munkelt, H. Kirchner, "Adaptive Background Estimation and Foreground Detection using Kalman-Filtering", *Proceedings of International Conference on recent Advances in Mechatronics, ICRAM'95*, UNESCO Chair on Mechatronics, 193-199, 1995.

[6] D. R. Magee, "Tracking Multiple Vehicles using Foreground, Background and Motion Models", *Report 2001.21*, University of Leeds, 2001.

[7] M. Harville, G. Gordon, J. Woodfill, "Foreground segmentation using adaptive mixture models in color and depth", *Workshop on Detection and Recognition of Events in Video*, 2001.

[8] O. Javed, K. Shafique, M. Shah, "A Hierarchical Approach to Robust Background Subtraction using Color and Gradient Information", *IEEE Workshop on Motion and Video Computing*, Orlando, Dec 5-6 2002.

[9] R. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomato, O. Hasegawa, "A System for Video Surveillance and Monitoring: VSAM Final Report", *Technical report CMU-RI-TR-00-12*, Robotics Institute, Carnegie Mellon University, May 2000.

[10] D. Beymer, P. McLauchlan, B. Coifman, J. Malik, "A Real-Time Computer Vision System for Measuring Traffic Parameters", *Proc. IEEE - Computer Vision and Pattern Recognition,* IEEE, 1997, pp 495-501.

[11] S. M. Smith, J. M. Brady, "ASSET-2: Real-time motion segmentation and shape tracking", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):814-820, 1995.

[12] M. Kass, A. Witkin, D. Terzopoulos, "Snakes: Active Contour Models", *International Journal of Computer Vision*, vol. 1, pp. 321 - 331, January 1988.

[13] F. Leymarie, M. D. Levine, "Tracking Deformable Objects in the Plane Using Active Contour Model", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 6, June 1993.

[14] Z. Kalafatić, "Praćenje objekata uporabom rijetkog optičkog toka", *Doktorska disertacija*, Sveučiliše u Zagrebu, Fakultet Elektrotehnike i Računarstva, Zagreb 1999.

[15] S. Lefèvre, J.G. Gérard, A. Piron, N. Vincent, "An Extended Snake Model For Real-time Multiple Object Tracking", *International Workshop on Advanced*

Concepts for Intelligent Vision Systems, pp. 268-275, Ghent (Belgium), September 2002.

[16] G. D. Sullivan, "Visual interpretation of known objects in constrained scenes", *Phil. Trans. Roy. Soc. (B)*, 337:361 - 370, 1992.

[17] D. Koller, K. Daniilidis, H.-H. Nagel, "Model-Based Object Tracking in Monocular Image Sequences of Road Traffic Scenes", *International Journal of Computer Vision*, 10(3):257 - 281, 1993.

[18] K.Toyama, J. Krumm, B. Brumitt, B. Meyers, "Wallflower: Principles and Practice of Background Maintenance", *International Conference on Computer Vision*, pp. 255 - 261, September 1999.

[19] J. Hynoski, H.R. Wu, "Active Vision – a survey of the field and research directions", *Technical Report 95-04*, Department of Robotics and Digital Technology, Faculty of Computing and Information Technology, Monash University, 1995.

[20] www.hrstud.hr/psihologija/bioloska/skripta/morfo/morfoTEXT.pdf, 2003.

[21] R. Wodnicki, G. W. Roberts and M. D. Levine, "A foveated image sensor in standard CMOS technology", *IEEE Custom Integrated Circuits Conference*, Santa Clara, Califorina, pp. 357-361, May 1995.

[22] Attendees of the NSF Active Vision Workshop, "Promising Directions of Active Vision", *University of Chicago Technical Report CS 91-27*, November, 1991.

[23] S.M. Smith, J.M. Brady, "SUSAN - a new approach to low level image processing", *Int. Journal of Computer Vision*, 23(1):45-78, May 1997.

[24] http://www.fmrib.ox.ac.uk/~steve/susan/susan2l.c, 2003.

[25] http://www-sop.inria.fr/robotvis/personnel/zzhang/Publis/Tutorial-Estim/node25.html, 2003.

[26] A. Selinger, L. Wixson, "Classifying moving objects as rigid or non-rigid without correspondences", *DARPA Image Understanding Workshop (IUW)*, Monterey, CA, November 1998, pp. 341-348.

Prilog A:
Opis programske implementacije

U ovom prilogu opisana je programska implementacija oba razvijena sustava.

Oba sustava implementirana su kao postupci za ljusku CVSH (engl. *computer vision shell*) razvijenu na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave (ZEMRIS) Fakulteta elektrotehnike i računarstva. Ljuska je u stvari jedan projekt, napisan u jeziku *C++*, koji implementira pribavljanje slike iz različitih izvora (kao što su datoteke i digitalizatori slike) i prikaz slike. Pored toga, ljuska sadrži i razne postupke osnovne obrade slike, klase za različite tipove podataka, klase koje omogućju višedretveno (engl. *multithreaded*) programiranje, klase koje omogućuju manipuliranje upravljivim postoljima i kamerama, klase koje implementiraju TCP/IP komunikaciju i sl. U ljusku se dodaju novi postupci obrade jednostavnim uključivanjem datoteka s obradom u projekt.

U nastavku priloga opisane su klase i funkcije koje implementiraju postupke obrade korištene u realizaciji sustava SAPPO i SIVS.

Sustav za aktivno praćenje pokretnih objekata - SAPPO

Ovdje je dan opis programske implementacije sustava SAPPO. Najprije je dan popis (važnijih) korištenih klasa i funkcija po datotekama, nakon čega se detaljno opisuje svaka pojedina klasa i funkcija. Na kraju je opisan način rada postupka.

Korištene klase i funkcije

Datoteke korištene za implementaciju SAPPO-a nalaze se unutar ljuske u direktoriju `adr`. Slijedi popis datoteka te klasa i funkcija sadržanih u njima.

Datoteke `alg_corner.h` i `alg_corner.cpp`:

```
void corner(const dib_base& src,  int thr,
            CORNER_LIST cl, dib_base& dst)

class CornerWorker
```

Datoteke `alg_labeled_dp.h` i `alg_labeled_dp.cpp`:

```
void labeled_dp(dib_base& fr1, const dib_base& fr2,
                int rect_top, int rect_left,
                int rect_bottom, int rect_right,
                int d_row, int d_col,
                int diff_thr, int size_thr,
                dib_base& dst, RegionVector  &rv)

class LabeledDpWorker
```

Datoteke `ccam_ptuBin.h` i `ccam_ptuBin.cpp`:

```
class ccam_ptuBin
class ccam_ptuBinImp
```

Datoteke `ip_track.h` i `ip_track.cpp`:

```
class ip_track
class ip_track_imp
class track_worker
Feature CalculateDisparity(FeatureVector& fv)
```

Datoteke `util_labeling.h` i `util_labeling.cpp`:

```
class Labeling
```

Datoteke `opcodes.h`, `ptu.h`, `ptu.c`, `linuxser.h` i `linuxser.cpp` stigle su s upravljivim postoljem. Sadrže primitive za komunikaciju s upravljivim postoljem.

Datoteke `susan.h` i `susan.c` skinute su s interneta (kao jedna datoteka koja je kasnije razbijena na dvije). Sadrže funkcije koje implementiraju SUSAN detektor rubova.

Opis korištenih klasa i funkcija

Funkcija: `void corner(const dib_base& src, int thr, CORNER_LIST cl, dib_base& dst)`

Opis:

Funkcija koja predstavlja sučelje prema postupku detekcije uglova korištenjem SUSAN detektora uglova. Kreira objekt klase `CornerWorker` koji provodi detekciju uglova.

Argumenti:

`src` - ulazna slika

`thr` - prag sivih vrijednosti

`cl` - lista dobivenih uglova

`dst` - izlazna slika (ulazna slika s označenim uglovima)

Klasa: `CornerWorker`

Definicija:

```
class CornerWorker{
private:
  int threshold;
  CORNER *corner_list;
public:
```

```
  typedef const dib_base& Param1;
  typedef dib_base& Param2;
public:
  CornerWorker(int t, CORNER *cl);

  template <class PixelSrc, class PixelDst>
  inline int operator() (
    const dib_base& dib_src, dib_base& dib_dst,
    const PixelSrc& dummy1, const PixelDst& dummy2);
};
```

Opis:

Korištenjem gotovih funkcija SUSAN detektora uglova detektira uglove u ulaznoj slici, pohranjuje uglove u listu i stvara izlaznu sliku s označenim uglovima.

Podatkovni članovi:

`threshold` - prag sivih vrijednosti korišten za detekciju uglova

`corner_list` - pokazivač na polje uglova

Funkcijski članovi:

`CornerWorker(int t, CORNER *cl)` - konstruktor, provodi inicijalizaciju podatkovnih članova

`inline int operator() (const dib_base& dib_src, dib_base& dib_dst,const PixelSrc& dummy1, const PixelDst& dummy2)` - izvršava detekciju rubova na ulaznoj slici `dib_src` i stvara izlaznu sliku `dib_dst`

Funkcija:
```
void labeled_dp(dib_base& fr1, const dib_base& fr2,
                int rect_top, int rect_left,
                int rect_bottom, int rect_right,
                int d_row, int d_col,
                int diff_thr, int edge_thr, int size_thr,
                dib_base& dst, RegionVector  &rv)
```

Opis:

Funkcija predstavlja sučelje prema postupku označene diferencije slika. Kreira objekt tipa `LabeledDpWorker` koji iz ulazne slike izračunava oznčenu diferenciju slika.

Argumenti:

`fr1` - prethodni okvir

`fr2` - trenutni okvir

`rect_top`, `rect_left` - koordinate gornjeg lijevog ugla pravokutnika u kojem se izračunava označena diferencija slika

`rect_bottom`, `rect_right` - koordinate donjeg desnog ugla pravokutnika u kojem se izračunava označena diferencija slika

`d_row` - pomak između prethodnog okvira po y koordinati (- gore, + dolje)

`d_col` - pomak između prethodnog okvira po x koordinati (- lijevo, + desno)

`diff_thr` - prag sivih vrijednosti za diferenciju slika

`edge_thr` - prag rubnosti da detekciju rubova

`size_thr` - prag veličine za filter veličine

`dst` - izlazna označena diferencija slika

`rv` - lista pronađenih aktivnih regija označene diferencije slika

Klasa: `LabeledDpWorker`

Definicija:

```
class LabeledDpWorker{
private:
  dib_base *pdib_ref;
  int rect_top;
  int rect_left;
  int rect_bottom;
  int rect_right;
  int d_row, d_col;
  int diff_threshold;
  int edge_threshold;
  int size_threshold;
  RegionVector *rv;
public:
  typedef const dib_base& Param1;
  typedef dib_base& Param2;
public:
  LabeledDpWorker(dib_base& r, int r_t,
                  int r_l, int r_b,
                  int r_r, int dr, int dc,
                  int dt, int et, int st,
                  RegionVector &RV);
  template <class PixelSrc, class PixelDst>
  inline int operator() (
    const dib_base& dib_src, dib_base& dib_dst,
    const PixelSrc& dummy1, const PixelDst& dummy2);
};
```

Opis:

Za zadani prethodni i trenutni okvir računa označenu diferenciju slika, pohranjuje pronađene aktivne regije u listu i stvara sliku označene diferencije slika.

Podatkovni članovi:

`pdib_ref` - pokazivač na prethodni okvir

`rect_top`, `rect_left` - koordinate gornjeg lijevog ugla pravokutnika u kojem se izračunava označena diferencija slika

`rect_bottom`, `rect_right` - koordinate donjeg desnog ugla pravokutnika u kojem se izračunava označena diferencija slika

`d_row` - pomak između prethodnog okvira po y koordinati (- gore, + dolje)

`d_col` - pomak između prethodnog okvira po x koordinati (- lijevo, + desno)

`diff_threshold` - prag sivih vrijednosti za diferenciju slika

`edge_threshold` - prag rubnosti da detekciju rubova

`size_threshold` - prag veličine za filter veličine

`rv` - pokazivač na listu pronađenih aktivnih regija označene diferencije slika

Funkcijski članovi:

`LabeledDpWorker(dib_base& r, int r_t, int r_l, int r_b, int r_r, int dr, int dc, int dt, int et, int st, RegionVector &RV)` - konsturktor, inicijalizira podatkovne članove

`inline int operator() ( const dib_base& dib_src, dib_base& dib_dst, const PixelSrc& dummy1, const PixelDst& dummy2)` - računa označenu diferenciju slika trenutnog okvira `dib_src` i prethodnog okvira `pdib_ref` i rezultat sprema u `dib_dst`, podatke o aktivnim regijama sprema u listu na koju pokazuje `rv`

Klasa: `ccam_ptuBin`

Definicija:

```
class ccam_ptuBin
{
  boost::scoped_ptr<ccam_ptuBinImp> pimp;
public:
  ccam_ptuBin(int port);
  virtual ~ccam_ptuBin();
public:
  virtual void setPan(double pan);
  virtual void setTilt(double tilt);
public:
  virtual void waitCompletion();
public:
  virtual double pan();
  virtual double tilt();
  virtual void getPanTilt(double& p, double& t);
public:
  void setIndependent();
  void setVelocity();
  void setSlaved();
  void setImmediate();
  void setPanOfset(double o);
  void setTiltOfset(double o);
  void setPanSpeed(double ps);
  void setTiltSpeed(double ts);
};
```

Opis:

Klasa predstavlja sučelje prema upravljiom postolju *PTU-46-17.5* trtke *Directed Perception*. Sama klasa ne implementira komunikaciju s postoljem, već koristi funkcije dobivene od proizvođača.

Podatkovni članovi:

`pimp` - pokazivač na objekt klase `ccam_ptuBinImp` koji služi za inicijalizaciju komunikacije

Funkcijski članovi:

`ccam_ptuBin(int port)` - konstruktor, inicijalizira podatkovni član i izdaje komande za postavljanje upravljivog postolja u nulti položaj

`virtual ~ccam_ptuBin()` - destruktor, ne radi ništa

`virtual void setPan(double pan)` - izdavanje naredbe za zauzimanje apsolutnog kuta zakreta `pan` (u stupnjevima)

`virtual void setTilt(double tilt)` - izdavanje naredbe za zauzimanje apsolutnog kuta nagiba `tilt` (u stupnjevima)

`virtual void waitCompletion()` - blokira pozivajući program sve dok upravljivo postolje ne izvrši zadane naredbe

`virtual double pan()` - upravljivom postolju izdaje naredbu za očitanje kuta zakreta, dobiveni kut zakreta (u stupnjevima) vraća kao povratnu vrijednost

`virtual double tilt()` - upravljivom postolju izdaje naredbu za očitanje kuta nagiba, dobiveni kut nagiba (u stupnjevima) vraća kao povratnu vrijednost

`virtual void getPanTilt(double& p, double& t)` - izdaje naredbu za očitanje kuteva zakreta i nagiba, kroz argumente `p` i `t` vraća te kuteve (u stupnjevima)

`void setIndependent()` - izdaje naredbe koje postavljaju upravljivo postolje u *independent* način rada

`void setVelocity()` - izdaje naredbe koje postavljaju upravljivo postolje u *pure velocity* način rada

`void setSlaved()` - izdaje naredbe koje postavljaju upravljivo postolje u *slaved* način rada

`void setImmediate()` - izdaje naredbe koje postavljaju upravljivo postolje u *immediate* način rada

`void setPanOfset(double o)` - izdavanje naredbe za relativni otklon kuta zakreta za `o` (u stupnjevima)

`void setTiltOfset(double o)` - izdavanje naredbe za relativni otklon kuta nagiba za `o` (u stupnjevima)

`void setPanSpeed(double ps)` - izdavanje naredbe za postavljanje brzine zakreta `ps` (u stupnjevima u sekundi)

`void setTiltSpeed(double ts)` - izdavanje naredbe za postavljanje brzine nagiba `ts` (u stupnjevima u sekundi)

Klasa: `ccam_ptuBinImp`

Definicija:

```
class ccam_ptuBinImp{
public:
  ccam_ptuBinImp(int port);
  ~ccam_ptuBinImp();
  portstream_fd COMstream;
};
```

Opis:

Klasa čija je jedina funkcija stvaranje i uništavanje struktura vezanih uz serijsku komunikaciju.

Podatkovni članovi:

`COMstream` - broj koji se dobiva prilikom otvaranja serijskog porta, koristi se prilikom zatvaranja serijskog porta

Funkcijski članovi:

`ccam_ptuBinImp(int port)` - otvara serijski port port za komunikaciju

`~ccam_ptuBinImp()` - zatvara otvoreni serijski port

Klasa: `ip_track`

Definicija:

```
class ip_track:
  public ip_base
{
public:
  ip_track(const dib_fmt&);
  virtual ~ip_track();
public:
  virtual const dib_base& dst_dib();
  virtual char const* name();
  static char const* s_name();
private:
  virtual void process(
    const dib_base& src,
    const win_eventVectorProtocol& events,
    win_annotationProtocol& ann);
private:
  virtual void refmt();
  const dib_base& dst_dib_labeled_dp() const;
  win_base& win_labeled_dp();
public:
  virtual void profile(ui_profileDataProtocol& pd);
  virtual void config(cli_wrapProtocol& cli);
private:
  void dumpParams(std::ostream& os, char const* intro);
private:
  ip_track_imp* pimp;
  static ip_creator<ip_track> creator;
};
```

Opis:

Ova klasa predstavlja sučelje prema ljusci, pomoću nje se dodaje postupak obrade u ljusku.

Podatkovni članovi:

pimp - pokazivač na objekt koji sadrži podatke vezane uz postupak obrade

Funkcijski članovi:

`ip_track(const dib_fmt&)` - konstruktor, inicijalizira podatkovni član pimp

`virtual ~ip_track()` - destruktor, ne radi ništa

`virtual const dib_base& dst_dib()` - vraća sliku za prikaz

`virtual char const* name()` - poziva funkcijski član `s_name()` i vraća rezultat poziva

`static char const* s_name()` - vraća naziv postupka obrade, u ovom slučaju `"track"`

`virtual void process(const dib_base& src, const win_eventVectorProtocol& events, win_annotationProtocol& ann)` - funkcija u koju se stavlja obrada, ulaz u funkciju su slika `src`, ulazni događaji `events` (poput pritiska neke tipke ili klika mišem), izlaz su oznake `ann` (grafika ili tekst) koje se stavljaju na sliku za prikaz

`virtual void refmt()` - funkcija kojom se mijenjaju formati slika unutar postupka

`const dib_base& dst_dib_labeled_dp() const` - funkcija koja vraća sliku označene diferencije slika

`win_base& win_labeled_dp()` - funkcija koja vraća prozor koji prikazuje sliku označene diferencije slika

`virtual void profile(ui_profileDataProtocol& pd)` - funkcija koja se poziva kada se zaustavlja obrada, u `pd` se stavljaju informacije o vremenu trajanja pojedinih koraka obrade

`virtual void config(cli_wrapProtocol& cli)` - funkcija koja se poziva kada se žele pregledati i izmijeniti parametri postupka

`void dumpParams(std::ostream& os, char const* intro)` - funkcija koja ispisuje trenutne vrijednosti parametara

Klasa: `ip_track_imp`

Definicija:

```
struct ip_track_imp{
  int dimension;
  int height, width;
  dib_base srcPrev_;
  dib_base dst_;
  dib_base labDp_;
  win_annotation ann_;
  win_wrap win_labeled_dp;
  boost::shared_ptr<track_worker> tWorker_;
  thr::ThreadSync thread_;
public:
  ip_track_imp(const dib_fmt&);
  ~ip_track_imp();
  dib_fmt fmt;
  void waitWorker();
  void startWorker(const dib_base& src);
  void display(win_annotationProtocol& ann);
public:
  const dib_base& dst_dib();
  void reformat(const dib_fmt&);
  void profile(ui_profileDataProtocol& pd);
  void config(cli_wrapProtocol& cli);
};
```

Opis:

Ova klasa upravlja objektom (proslijeđuje ulaznu sliku, pokreće obradu i čeka završetak obrade) klase `track_worker` koji provodi praćenje. Također

služi kao most između klasa `ip_track` (sučelja prema van) i `track_worker` (obrada).

Podatkovni članovi:

`dimension` - broj koji određuje smanjenje u svakoj od dimenzija formata ulazne slike

`height`, `width` - visina i širina smanjene slike

`srcPrev_` - prethodni okvir

`dst_` - okvir koji se prikazuje

`labDp_` - slika označene diferencije slika

`ann_` - oznake koje su rezultat obrade

`win_labeled_dp` - prozor u kojemu se prikazuje slika označene diferencije slika

`tWorker_` - pokazivač na objekt klase `track_worker` koji provodi obradu

`thread_` - dretva uz koju se veže postupak obrade

`fmt` - format ulazne slike

Funkcijski članovi:

`ip_track_imp(const dib_fmt&)` - konstruktor, inicijalizira podatkovne članove

`~ip_track_imp()` - destruktor, ne radi ništa

`void waitWorker()` - čeka semafor koji signalizira završetak obrade, po potrebi iscrtava sliku označene diferencije slika, preuzima oznake od procesa obrade

`void startWorker(const dib_base& src)` - proslijeđuje ulaznu sliku objektu koji provodi obradu i postavlja semafor koji signalizira početak obrade

`void display(win_annotationProtocol& ann)` - proslijeđuje oznake dobivene obradom u ljusku

`const dib_base& dst_dib()` - vraća prikaznu sliku

`void reformat(const dib_fmt&)` - mijenja se format korištenih slika u format zadan argumentom

`void profile(ui_profileDataProtocol& pd)` - funkcija koja se poziva kada se zaustavlja obrada, u pd se stavljaju informacije o vremenu trajanja pojedinih koraka obrade

`void config(cli_wrapProtocol& cli)` - funkcija koja se poziva kada se žele pregledati i izmijeniti parametri postupka

Klasa: `track_worker`

Definicija:

```
class track_worker:
  public thr::ThreadSyncWorker
{
private:
```

```
  thr::Mutex mtx1;
  thr::Mutex mtx2;
  int bsem1_;
  int bsem2_;
private:
  bool exit_;
private:
  int diff_thr;
  int edge_thr;
  int size_thr;
  dib_base srcPrev_;
public:
  bool draw_labeled_dp;
  double pan_speed;
  double tilt_speed;
  ccam_ptuBin ptu;
  FeatureVector prev_corners;
  std::vector<int> tinfo;
public:
  dib_base src_;
  dib_base dib_main_;
  dib_base dibLabDp_;
  dib_base dib_corner_;
  win_annotation ann_;
public:
  int dimension, height, width;
  bool first_frame;
public:
  track_worker(const dib_fmt& fmt, int d);
  ~track_worker();
public:
  void setBSem1();
  void setBSem2();
  void waitBSem1();
  void waitBSem2();
  void waitBSem1NoClr();
public:
  virtual int operator()();
  virtual void cleanup();
public:
  void profile(ui_profileDataProtocol& pd);
  void config(cli_wrapProtocol& cli);
private:
  void dumpParams(std::ostream& os, char const* );
};
```

Opis:

Klasa koja implementira postupke obrade i upravlja upravljivim postoljem.

Podatkovni članovi:

`mtx1` - mutex vezan uz semafor 1

`mtx2` - mutex vezan uz semafor 2

`bsem1_` - varijabla binarnog semafora 1 čije postavljanje označava početak obrade

`bsem2_` - varijabla binarnog semafora 2 čije postavljanje označava završetak obrade

`bool exit_` - varijabla koja služi za zaustavljanje dretve koja provodi obradu

`diff_thr` - prag sivih vrijednosti diferencije slika

`edge_thr` - prag rubnosti za označenu diferenciju slika

`size_thr` - prag veličine za filter veličine

`srcPrev_` - prethodni okvir

`draw_labeled_dp` - varijabla koja označava hoće li se iscrtavati označena diferencija slika

`pan_speed` - trenutno zadana brzina zakreta

`tilt_speed` - trenutno zadana brzina nagiba (ne koristi se)

`ptu` - objekt pomoću kojeg se upravlja upravljivim postoljem

`prev_corners` - lista koja sadrži uglove detektirane u prethodnom okviru

`tinfo` - lista koja sadrži podatke o vremenima trajanja pojedinih faza obrade

`src_` - trenutni okvir

`dib_main_` - okvir za prikaz

`dibLabDp` - slika označene diferencije slika

`dib_base dib_corner_` - ulazna slika s označenim uglovima

`ann_` - oznake dobivene obradom, sadrži opisani pravokutnik praćenog objekta

`dimension` - smanjenje svake dimenzije ulazne slike

`height, width` - širina i visina smanjene ulazne slike

`first_frame` - varijabla koja označava da li se obrađuje prvi okvir

Funkcijski članovi:

`track_worker(const dib_fmt& fmt, int d)` - konstruktor, inicijalizira podatkovne članove

`~track_worker()` - destruktor, ne radi ništa

`void setBSem1()` - postavlja binarni semafor 1

`void setBSem2()`- postavlja binarni semafor 2

`waitBSem1()`- čekanje da se postavi binarni semafor 1, nakon čega se semafor 1 potroši

`void waitBSem2()` - čekanje da se postavi binarni semafor 2, nakon čega se semafor 2 potroši

`void waitBSem1NoClr()` - čekanje da se postavi binarni semafor 1, nakon čega semafor 1 ostaje u postavljenom stanju

`virtual int operator()()` - u ovoj funkciji je smještena sva obrada (tijelo ove funkcije predstavlja dretvu s obradom)

`virtual void cleanup()` - signalizira završetak rada dretvi s obradom tako da postavlja varijablu `exit_`

`void profile(ui_profileDataProtocol& pd)` - funkcija koja se poziva kada se zaustavlja obrada, u `pd` se stavljaju informacije o vremenu trajanja pojedinih koraka obrade

`void config(cli_wrapProtocol& cli)` - funkcija koja se poziva kada se žele pregledati i izmijeniti parametri postupka

`void dumpParams(std::ostream& os, char const* intro)` - funkcija koja ispisuje trenutne vrijednosti parametara

Funkcija: `Feature calculateDisparity(FeatureVector& fv)`

Opis:

Funkcija koja pronađene vektore pomaka značajki najprije grupira po sličnosti, pronalazi grupu s najvećim brojem vektora pomaka značajki i pronalazi sredinu te grupe značajki. Povratna vrijednost je vektor koji predstavlja sredinu.

Argumenti:

`fv` - lista vektora pomaka značajki

Klasa: `Labeling`

Definicija:

```
class Labeling
{
private:
  bool calculated;
  int next_label;
  int label_no;

  struct Similarity
  {
    Similarity();
    Similarity(int _id, int _sameas);
    Similarity(int _id);
    int id, sameas, tag;
    int number;
  };

  bool is_root_label(int id);
  int root_of(int id);
  bool is_equivalent(int id, int as)
  std::vector<Similarity> labels;
  Labeling(unsigned int soft_maxlabels = 50);
  void init();
  bool setEquivalent(int l1, int l2);
  inline int getNextLabel();
  inline int getNoOfLabels();
  inline int getLabelOccNo(int which);
  inline void incLabel(int which);
  void calculateNewLabelsRegions(int threshold);
  inline int getNewLabel(int old);
};
```

Opis:

Klasa koja služi za postavljanje ekvivalencije između oznaka u procesu sekvencijalnog pronalaženja povezanih komponenti i procesu grupiranja na temelju udaljenosti. Također se koristi kao filter veličine jer zanemaruje grupe koje imaju manje elemenata od određenog praga.

Podatkovni članovi:

`calculated` - označava da li su izračunate nove oznake

`next_label` - slijedeća oznaka po redu, broj dosad korištenih oznaka

`label_no` - broj različitih oznaka nakon izračunavanja

`labels` - lista podataka tipa `struct Similarity` koja služi za vođenje podataka o ekvivalenciji oznaka

Funkcijski članovi:

`bool is_root_label(int id)` - funkcija koja vraća istinu ako je promatrana oznaka `id` korijen stabla međusobno ekvivalentnih oznaka

`int root_of(int id)` - funkcija koja vraća korijen stabla oznaka ekvivalentnih oznaci `id`

`bool is_equivalent(int id, int as)` - vraća istinu ako su oznake `id` i `as` ekvivalentne

`Labeling(unsigned int soft_maxlabels = 50)` - konstruktor, inicijalizira podatkovne članove

`void init()` - funkcija za inicijalizaciju objekta, poziva se i iz konstruktora

`bool setEquivalent(int l1, int l2)` - funkcija postavlja ekvivalentnost oznaka `l1` i `l2`

`inline int getNextLabel()` - funkcija dodaje novu oznaku u listu oznaka i vraća novu oznaku

`inline int getNoOfLabels()` - funkcija koja vraća broj oznaka nakon izračunavanja novih oznaka

`inline int getLabelOccNo(int which)` - funkcija vraća ukupan broj pojavljivanja za grupu ekvivalentnih oznaka

`inline void incLabel(int which)` - funkcija koja uvećava broj pojavljivanja oznake `which`

`void calculateNewLabelsRegions(int threshold)` - funkcija provodi izračunavanje novih oznaka tako da svaka grupa ekvivalentnih oznaka (čiji je broj elemenata veći od praga `threshold`) dobiva novu, jedinstvenu oznaku

`inline int getNewLabel(int old)` - funkcija koja za zadanu staru oznaku old vraća novu oznaku dobivenu izračunavanjem.

Način rada

Programska izvedba postupka može se podijeliti u dva glavna dijela: glavna dretva (proizvođač) i dretva za obradu (potrošač).

Glavnu dretvu čine ljuska i funkcijski član `ip_track.process(..)`. Rad glavne dretve može se prikazati slijedećim koracima (koji se neprestano ciklički ponavljaju):

1. *Pribavi sliku*. Ovaj korak izvodi dio ljuske zadužen za pribavljanje slike. Korak završava pozivom funkcijskog člana `ip_track.process(..)` koji svojim argumentom prenosi pribavljenu sliku.
2. *Pokreni obradu*. Postavljanjem podatkovnog člana `track_worker.src_` pribavljena slika proslijeđuje se dretvi za obradu što se signalizira postavljanjem binarnog semafora 1. Ovom koraku odgovara poziv funkcijskog člana `ip_track_imp.startWorker(..)`.
3. *Čekaj završetak obrade*. Čeka se na binarni semafor 2. Kada se prođe taj semafor dostupni su rezultati obrade: slika za prikaz, slika označene diferencije slika i oznake (opisani pravokutnik praćenog objekta). Ako je potrebno, slika označene diferencije slika se iscrtava. Oznake i slika za prikaz se proslijeđuju ljusci koja ih zatim i iscrtava. Ovaj korak odgovara pozivu funkcijskog člana `ip_track_imp.waitWorker()`.

Dretvu za obradu čini funkcijski član `track_worker.operator()()`. Rad ovog dijela programa opisuju slijedeći koraci (koji se neprestano ciklički ponavljaju):

1. *Čekaj znak za početak obrade*. Čeka se na binarni semafor 1. Kada se prođe taj semafor u podatkovnom članu `track_worker.src_` se nalazi ulazna slika.
2. *Smanji ulaznu sliku*. Ulazna slika se, korištenjem funkcije `void subsample( const dib_base& src, int dimension, dib_base& dst)`, koja je dostupna s ljuskom, smanjuje za `dimension` (koristi se vrijednost 2) po svakoj dimenziji. Rezultat smanjivanja ulazne slike `track_worker.src_` je u podatkovnom članu `track_worker.dib_main_`.
3. *Pronađi uglove u sceni*. Korištenjem funkcije `corner(..)` dobiva se lista uglova pronađenih u trenutnom okviru.
4. *Odredi pomake vektora značajki iz prethodnog u trenutni okvir*. Pronalazi se korespodencija uglova pronađenih u prethodnom i trenutnom okviru. Na taj način se dobiva lista vektora pomaka značajki iz jednog okvira u drugi.
5. *Odredi dominantni pomak*. Poziva se funkcija `calculateDisparity(..)` za listu vektora pomaka. Funkcija vraća dominantni vektor pomaka.
6. *Izračunaj označenu diferenciju slika*. Korištenjem podatka o pomaku trenutnog okvira u odnosu na prethodni (dominantni pomak), pomoću funkcije `labeled_dp(..)` izračunava se označena diferencija slika i dobiva se lista aktivnih regija.
7. *Eliminiraj pogrešne aktivne regije*. Iz daljnjeg razmatranja se izuzimaju aktivne regije kod kojih je odnos rubnih elemenata i ukupnog broja elemenata veći od određenog praga.
8. *Grupiraj aktivne regije*. Provodi se grupiranje regija na temelju udaljenosti njima opisanih pravokutnika.
9. *Izdaj naredbe za usmjeravalje prema objektu*. Za objekt se pretpostavlja da odgovara grupi koja sadrži najveći broj aktivnih regija.
10. *Daj znak za završetak obrade*. Postavlja se binarni semafor 2.

Verzije

Osim opisane verzije koja provodi postupak aktivnog praćenja, napravljena je i verzija koja je identična opisanoj verziji osim u dijelu praćenja. Druga verzija namijenjena je samo prikazu faza praćenja na datotekama koje sadrže snimljene sekvence i ne upravlja upravljivim postoljem. Datoteke druge verzije nalaze se u direktoriju track.

Platforme

Verzija koja implementira aktivno praćenje napisana je za operacijski sustav *Linux*. Korišten je *C++* prevodioc *g++* v3.0.2

Verzija koja samo prikazuje rad postupka praćenja napisana je tako da radi pod operacijskim sustavom *MS Windows* (98/2000/XP). Korišten je prevodioc *MSVC* 6.0.

Sustav za izlučivanje slike vozila iz scene - SIVS

Ovdje je dan opis programske implementacije sustava SIVS. Najprije je dan popis (važnijih) korištenih klasa i funkcija po datotekama, nakon čega se detaljno opisuje svaka pojedina klasa i funkcija. Na kraju je opisan način rada postupka.

Korištene klase i funkcije

Datoteke korištene za implementaciju SIVS-a nalaze se unutar ljuske u direktoriju `car_track`. Slijedi popis datoteka te klasa i funkcija sadržanih u njima.

Datoteke `ccam_client.h` i `ccam_server.cpp`:

```
class ccam_client
```

Datoteke `ccam_server.h` i `ccam_server.cpp`:

```
class ccam_IPworker
class ccam_server
```

Datoteke `dump_objects.h` i `dump_objects.cpp`:

```
void dump_objects(util::LayerVector &layers,
                  util::LayerVector &objects,
                  dib_base& dst);
class DumpObjectsWorker
```

Datoteke `ip_car_track.h` i `ip_car_track.cpp`:

```
class ip_car_track
class ip_car_track_imp
```

Datoteke `ip_ccam_server.h` i `ip_ccam_server.cpp`:

```
class ip_ccam_server
```

```
class ip_ccam_server_imp
```

Datoteke `trans_map.h` i `trans_map.cpp`:

```
class TransienceMap
class TmInputWorker
class TmExtractWorker
```

Datoteke `util_labeling.h` i `util_labeling.cpp`:

```
class Labeling
```

Datoteke `util_layer.h` i `util_layer.cpp`:

```
class Layer
```

Datoteka `util_queue.h`:

```
class Queue
```

Datoteke `util_tracked_object.h` i `util_tracked_object.cpp`:

```
class TrackedObject
```

Opis korištenih klasa i funkcija

Klasa: `ccam_client`

Definicija:

```
class ccam_client: public thr::ThreadSyncWorker
{
private:
  thr::Mutex mtx_cond;
  thr::Mutex mtx_cmd;
  thr::CondVar cvRequest;
  bool start;
  bool processing;
  bool finish;
  bool success;
  double dp, dt, dz, df;
  bool sp, st, sz, sf;
  bool gi;
  std::string host;
  int port;
private:
  sock::StreamClient* connect();
private:
  bool exit;
public:
  ccam_client(const char* h, int p;
  ~ccam_client();
  bool setPan(double angle);
```

```
  bool setTilt(double angle);
  bool setZoom(double angle);
  bool setFocus(double focus);
  bool setPanTilt(double pan, double tilt);
  bool setPanTiltZoom(double pan, double tilt, double zoom);
  bool getImage();
  bool finished();
  bool succeeded();
public:
  virtual int operator()();
  virtual void cleanup();
};
```

Opis:

Klasa implementira klijenta koji omogućuje slanje, preko TCP/IP, asinkronih zahtjeva serveru za postavljanjem parametara upravljive kamere. Asinkrono ovdje znači da kada se postavi neki zahtjev, ne čeka se na njegovo izvršenje, već se u nekom drugom trenutku provjerava da li je zahtjev obrađen. Zahtjevi se izdaju pozivanjem funkcijskih članova ove klase, nakon čega zasebna dretva šalje zahtjeve putem TCP/IP veze i čeka potvrdu o završetku. Sinkronizacija te dretve provodi se korištenjem mutexa i uvjetnih varijabli.

Podatkovni članovi:

`mtx_cond` - mutex vezan uz uvjetnu varijablu

`mtx_cmd` - mutex koji osigurava kritičan odsječak zadavanja i čitanja naredbi

`cvRequest` - uvjetna varijabla koja označava da je zahtjev postavljen

`start` - varijabla koja označava postavljanje zahtjeva

`processing` - varijabla koja označava da se zahtjev obrađuje

`finish` - varijabla koja označava da je obrada zahtjeva gotova

`success` - varijabla koja označava da li je zahtjev izvršen

`dp`, `dt`, `dz`, `df` - varijable u koje sadrže vrijednosti zahtjeva za podešavanjem kuta zakreta, kuta nagiba, kuta zooma i fokusa

`sp`, `st`, `sz`, `sf` - varijable koje označavaju da je postavljen zahtjev za podešavanjem kuta zakreta, kuta nagiba, kuta zuma i fokusa

`gi` - varijabla koje označava zahtjev za snimanjem trenutne slike kamere na strani servera

`host` - ime računala na koje se klijent spaja

`port` - port na kojem radi server

`exit` - varijabla kojom se označava kraj rada dretvi koja šalje zahtjeve i čeka njihovo izvršenje

Funkcijski članovi:

`sock::StreamClient* connect()` - stvara novi objekt klase StreamClient, koji implementira TCP/IP vezu, i vraća pokazivač na njega

`ccam_client(const char* h, int p)` - konstruktor, inicijalizira podatkovne članove

`~ccam_client()` - destruktor, ne radi ništa

`bool setPan(double angle)` - postavljaju se varijable zahtjeva za postavljanjem kuta zakreta i signalizira se uvjetna varijabla `cvRequest`

`bool setTilt(double angle)` - postavljaju se varijable zahtjeva za postavljanjem kuta nagiba i signalizira se uvjetna varijabla `cvRequest`

`bool setZoom(double angle)` - postavljaju se varijable zahtjeva za postavljanjem kuta zuma i signalizira se uvjetna varijabla `cvRequest`

`bool setFocus(double focus)` - postavljaju se varijable zahtjeva za postavljanjem fokusa i signalizira se uvjetna varijabla `cvRequest`

`bool setPanTilt(double pan, double tilt)` - postavljaju se varijable zahtjeva za postavljanjem kuta zakreta i kuta nagiba i signalizira se uvjetna varijabla `cvRequest`

`bool setPanTiltZoom(double pan, double tilt, double zoom)` - postavljaju se varijable zahtjeva za postavljanjem kuta zakreta, kuta nagiba i kuta zuma i signalizira se uvjetna varijabla `cvRequest`

`bool getImage()` - postavlja se varijabla koja označava izdavanje naredbe za uzimanje slike i signalizira se uvjetna varijabla `cvRequest`

`bool finished()` - provjerava se varijabla koja označava završetak obrade zahtjeva

`bool succeeded()` - provjerava se varijabla koja označava uspješnost završetka obrade zahtjeva

`virtual int operator()()` - sadrži dretvu za slanje i čekanje zahtjeva

`virtual void cleanup()` - pozivom se označava dretvi da mora završiti s radom

Klasa: `ccam_IPworker`

Definicija:

```
class ccam_IPworker:
  public thr::IPserver::CommonWorker
{
private:
  bool processing;
  int timeout;
  ccam_visca *ccam;
  int port;
  thr::Mutex mtx;
  thr::Mutex mtx_cond;
  thr::CondVar cv_get_img;
  bool get_img;
private:
  ccam_IPworker(int sp, int to)
  bool accepting();
  void setAccepting();
public:
  int handleConnection(thr::IPserver::PerConnectionWorker& cw,
      sock::Simple& s);
public:

  bool getImage();
  static ccam_IPworker* create(int sp, int to);
};
```

Opis:

Klasa koja implementira TCP/IP server, prima zahtjeve i obrađuje ih izdavanjem naredbi za postavljanje parametara upravljive kamere.

Podatkovni članovi:

`processing` - označava da li je obrada zahtjeva još u tijeku

`timeout` - period neaktivnosti veze nakon kojeg se veza zatvara

`ccam` - pokazivač na objekt klase ccam_visca pomoću kojeg se izdaju `naredbe` za promjene parametara upravljive kamere

`port` - port na kojemu se osluškuju zahtjevi

`mtx` - mutex koji štiti kritične odsječke čitanja i postavljanje varijable `processing`

`mtx_cond` - mutex vezan uz uvjetnu varijablu `cv_get_img`

`cv_get_img` - uvjetna varijabla kojom se signalizira naredba za uzimanje slike kamere

`get_img` - varijabla kojom se signalizira naredba za uzimanje slike kamere

Funkcijski članovi:

`ccam_IPworker(int sp, int to)` - konstruktor, inicijalizira podatkovne članove

`bool accepting()` - vraća istinu ako se trenutno ne obrađuje neki zahtjev

`void setAccepting()` - označava da se trenutno ne obrađuje nikakav zahtjev

`int handleConnection(thr::IPserver::PerConnectionWorker& cw, sock::Simple& s)` - obrađuje pojedine pristigle zahtjeve ili javlja da ne može prihvatiti zahtjev iz razloga što se neki drugi trenutno već obrađuje

`bool getImage()` - čeka se, određeno vrijeme, na uvjetnu varijablu `cv_get_image`, ako se uvjetna varijabla u međuvrementu signalizira, vraća istinu, a ako ne, vraća laž

`static ccam_IPworker*create(int sp, int to)` - kreira novi objekt ove klase i vraća pokazivač na njega

Klasa: `ccam_server`

Definicija:

```
class ccam_server
{
private:
  boost::shared_ptr<ccam_IPworker> cw;
  boost::shared_ptr<thr::IPserver> pips;
  thr::ThreadSync thr_ips;
public:
  ccam_server(int p, int sp, int to = 10000);
  ~ccam_server();
  bool done();
  inline bool getImage();
};
```

Opis:

Predstavlja sučelje implementacije servera prema van i sadrži članove koji implementiraju sam server.

Podatkovni članovi:

`cw` - pokazivač na objekt klase `ccam_IPworker` kojim se obrađuje pojedini pristigli zahtjev

`pips` - pokazivač na objekt klase `IPserver` koji implementira server koji prima zahtjeve i proslijeđuje ih na obradu

`thr_ips` - dretva koja se postavlja na objekt na koji pokazuje `pips`

Funkcijski članovi:

`ccam_server(int p, int sp, int to = 10000)` - konstruktor, inicijalizira podatkovne članove

`~ccam_server()` - destruktor, ne radi ništa

`bool done()` - vraća istinu ako je server završio s radom

`inline bool getImage()` - poziva `cw->getImage()`

Funkcija: `void dump_objects(util::LayerVector &layers, util::LayerVector &objects, dib_base& dst);`

Opis:

Funkcija služi kao sučelje za iscrtavanje maski svih objekata koji se trenutno nalaze u sceni. Samu obradu, iscrtavanje, odrađuje objekt klase `DumpObjectsWorker`.

Argumenti:

`layers` - referenca na listu objekata koji su zaustavljeni

`objects` - referenca na listu objekata koji se kreću

`dst` - izlazna slika u kojoj je maska svakog objekta iscrtana različitom nijansom sive

Klasa: `DumpObjectsWorker`

Definicija:

```
class DumpObjectsWorker{
private:
  util::LayerVector& layers;
  util::LayerVector& objects;
public:
  typedef const dib_base& Param1;
  typedef dib_base& Param2;
public:
   DumpObjectsWorker(util::LayerVector &lyr, util::LayerVector
&obj);
  template <class PixelSrc, class PixelDst>
  inline int operator() (
```

```
    const dib_base& dib_src,
    dib_base& dib_dst,
    const PixelSrc& dummy1, const PixelDst& dummy2);
};
```

Opis:

Implementira iscrtavanje maski objekata.

Podatkovni članovi:

`layers` - referenca na listu objekata zaustavljenih u sceni

`objects` - referenca na listu objekata u sceni koji se kreću

Funkcijski članovi:

`inline int operator() (const dib_base& dib_src, dib_base& dib_dst, const PixelSrc& dummy1, const PixelDst& dummy2)` - provodi obradu, odnostno iscrtavanje izlazne slike `dib_dst` (prvi parametar `dib_src` se ne koristi već je stavljen zato što je potreban ovakav oblik funkcije)

Klasa: `ip_car_track`

Definicija:

```
class ip_car_track:
  public ip_base
{
public:
  ip_car_track(const dib_fmt&);
  virtual ~ip_car_track();
public:
  virtual const dib_base& dst_dib();
  virtual char const* name();
  static char const* s_name();
private:
  virtual void process(
    const dib_base& src,
    const win_eventVectorProtocol& events,
    win_annotationProtocol& ann);
private:
  virtual void refmt();
  const dib_base& dst_dib_motion() const;
  const dib_base& dst_dib_background() const;
  const dib_base& dst_dib_regions() const;
  win_base& win_motion();
  win_base& win_background();
  win_base& win_regions();
public:
  virtual void profile(ui_profileDataProtocol& pd);
  virtual void config(cli_wrapProtocol& cli);
private:
  void dumpParams(std::ostream& os, char const* intro);
private:
  ip_car_track_imp* pimp;
  static ip_creator<ip_car_track> creator;
};
```

Opis:

Klasa predstavlja sučelje prema ljusci i implementaciju postupka obrade, vezanog uz segmentaciju, praćenje i izlučivanje slike vozila, u ljusku. Pomoću ove klase postupak se dodaje u ljusku.

Podatkovni članovi:

`pimp` - pokazivač na objekt koji sadrži podatke vezane uz implementaciju postupka obrade

Funkcijski članovi:

`ip_car_track(const dib_fmt&)` - konstruktor, inicijalizira podatkovne članove

`virtual ~ip_car_track()` - destruktor, uništava dinamički alocirani objekt

`virtual const dib_base& dst_dib()` - vraća sliku za prikaz

`virtual char const* name()` - poziva funkcijski član `s_name()` i vraća rezultat poziva

`static char const* s_name()` - vraća naziv postupka obrade, u ovom slučaju `"car_track"`

`virtual void process(const dib_base& src, const win_eventVectorProtocol& events, win_annotationProtocol& ann)` - funkcija u koju se stavlja obrada, ulaz u funkciju su slika `src`, ulazni događaji `events` (poput pritiska neke tipke ili klika mišem), izlaz su oznake `ann` (grafika ili tekst) koje se stavljaju na sliku za prikaz

`const dib_base& dst_dib_motion() const` - vraća sliku na kojoj su izdvojeni objekti iznad pozadine

`const dib_base& dst_dib_background() const` - vraća sliku pozadine

`const dib_base& dst_dib_regions() const` - vraća sliku maski objekata iznad pozadine

`win_base& win_motion()` - vraća prozor koji sadrži sliku izdvojenih objekata iznad pozadine

`win_base& win_background()` - vraća prozor koji sadrži sliku pozadine

`win_base& win_regions()` - vraća prozor koji sadrži maske objekata iznad pozadine

`virtual void refmt()` - funkcija kojom se mijenjaju formati slika unutar postupka

`virtual void profile(ui_profileDataProtocol& pd)` - funkcija koja se poziva kada se zaustavlja obrada, u `pd` se stavljaju informacije o vremenu trajanja pojedinih koraka obrade

`virtual void config(cli_wrapProtocol& cli)` - funkcija koja se poziva kada se žele pregledati i izmijeniti parametri postupka

`void dumpParams(std::ostream& os, char const* intro)` - funkcija koja ispisuje trenutne vrijednosti parametara

Klasa: `ip_car_track_imp`

Definicija:

```
struct ip_car_track_imp{
  ip_car_track_imp(const dib_fmt&);
  ~ip_car_track_imp();
  void reformat(const dib_fmt&);
  int dimension;
  int height, width;
  int size_thr;
  dib_fmt fmt;
  dib_base dib_car_track;
  dib_base dib_motion;
  dib_base dib_background;
  dib_base dib_subset;
  dib_base dib_regions;
  bool firstFrame;
  float height_thr;
  float width_thr;
  int q_size;
  int mt_thr;
  int sm_thr;
  int br_thr;
  TransienceMap *tm;
  bool proc;
  bool crosshair;
  ccam_visca ccam;
  double n_zoom;
  double n_pan;
  double n_tilt;
  double zoom;
  bool neutral;
  bool shot;
  int waitFrame;
  bool color;
  boost::shared_ptr<ccam_client> cc;
  thr::ThreadSync thread_;
  util::LayerVector lv;
  util::TrackedObjectVector objects;
  win_base& win_motion;
  win_base& win_background;
  win_base& win_subset;
  win_base& win_regions;
};
```

Opis:

Klasa sadrži sve podatke (parametre, klase za obradu, slike i prozore za prikaz) relevantne za postupak obrade (segmentacija, praćenje i izlučivanje slike vozila). Sadrži i funkcije koje inicijaliziraju i mijenjaju podatkovne članove.

Podatkovni članovi:

`dimension` - broj koji određuje smanjenje u svakoj od dimenzija formata ulazne slike

`height`, `width` - visina i širina smanjene ulazne slike

`size_thr` - prag veličine korišten za filter veličine

`fmt` - format ulazne slike

`dib_car_track` - slika za prikaz

`dib_motion` - slika koja sadrži izdvojene objekte iznad pozadine

`dib_background` - slika pozadine

`dib_subset` - izdvojena slika vozila

`dib_regions` - slika koja sadrži maske objekata iznad pozadine

`firstFrame` - varijabla koja označava da li je trenutni okvir prvi okvir, služi za inicijaliziranje podatkovnih struktura

`height_thr` - prag visine objekta za izdvajanje, predstavlja postotak visine ulazne slike

`width_thr` - prag širine objekta za izdvajanje, predstavlja postotak širine ulazne slike

`q_size` - broj okvira koji se pamte u prošlost

`mt_thr` - prag okidača pokreta

`sm_thr`- prag mjere stabilnosti

`br_thr` - prag razlike od pozadine

`tm` - pokazivač na objekt klase TransienceMap koji provodi detekciju pokreta i segmentaciju

`proc` - varijabla koja označava da li se ulazna slika obrađuje ili ne

`crosshair` - varijabla koja označava hoće li se iscrtavati mali križić u središtu slike za prikaz

`ccam` - objekt koji se koristi za upravljanje upravljivom kamerom

`n_zoom` - varijabla koja sadrži vrijednost kuta zuma u početnom položaju

`n_pan` - varijabla koja sadrži vrijednost kuta zakreta u početnom položaju

`n_tilt` - varijabla koja sadrži vrijednost kuta nagiba u početnom položaju

`zoom` - trenutna vrijednost kuta zooma
`neutral` - varijabla koja označava da li se upravljiva kamera nalazi u početnom položaju

`shot` - varijabla koja označava treba li izlučiti sliku objekta iz trenutnog okvira

`waitFrame` - varijabla koja označava koliko okvira treba pričekati prije no što se nastavi s obradom kako ne bi došlo do pogreške u obradi uslijed vraćanja kamere u početni položaj

`color` - varijabla koja označava hoće li se oznake iscrtavati crno bijelo ili u boji

`cc` - pokazivač na objekt klase `ccam_client` pomoću kojeg se serveru izdaju zahtjevi za postavljanjem kamere i naredbe za uzimanje slike

`thread_` - dretva uz koju se veže objekt klase `ccam_client`

`lv` - lista slojeva iznad pozadine, odnosno regija koje odgovaraju zaustavljenim objektima

`objects` - lista praćenih objekata

`win_motion` - prozor u kojem se prikazuje slika izdvojenih objekata iznad pozadine

`win_background` - prozor u kojemu se prikazuje slika pozadine

`win_subset` - prozor u kojem se prikazuje izlučena slika vozila

`win_regions` - prozor u kojem se prikazuju maske objekata izdvojenih iznad pozadine

Funkcijski članovi:

`ip_car_track_imp(const dib_fmt&)` - konstruktor, inicijalizira podatkovne članove

`~ip_car_track_imp()` - destruktor, uništava dinamički alocirane objekte

`void reformat(const dib_fmt&)` - mijenja format korištenih slika na format zadan argumentom

Klasa: `ip_ccam_server`

Definicija:

```
class ip_ccam_server:
  public ip_base
{
public:
  ip_ccam_server(const dib_fmt&);
  virtual ~ip_ccam_server();
public:
  virtual const dib_base& dst_dib();
  virtual char const* name();
  static char const* s_name();
private:
  virtual void process(
    const dib_base& src,
    const win_eventVectorProtocol& events,
    win_annotationProtocol& ann);
private:
  virtual void refmt();
public:
  virtual void profile(ui_profileDataProtocol& pd);
  virtual void config(cli_wrapProtocol& cli);
private:
  void dumpParams(std::ostream& os, char const* intro);
private:
  ip_ccam_server_imp* pimp;
  static ip_creator<ip_ccam_server> creator;
};
```

Opis:

Ova klasa predstavlja sučelje prema ljusci i implementaciju postupka koji implementira TCP/IP server koji zaprima zahtjeve za upravljanjem upravljivom kamerom i uzimanje slike. Korištenjem ove klase postupak se dodaje u ljusku.

Podatkovni članovi:

`pimp` - pokazivač na objekt koji sadrži podatke vezane uz implementaciju postupka obrade

Funkcijski članovi:

`ip_ccam_server(const dib_fmt&)` - konstruktor, inicijalizira podatkovni član

`virtual ~ip_ccam_server()` - destruktor, uništava dinamički alocirani objekt

`virtual const dib_base& dst_dib()` - vraća sliku za prikaz

`virtual char const* name()` - poziva funkcijski član `s_name()` i vraća rezultat poziva

`static char const* s_name()` - vraća naziv postupka obrade, u ovom slučaju `"ccam_server"`

`virtual void process(const dib_base& src, const win_eventVectorProtocol& events, win_annotationProtocol& ann)` - funkcija u koju se stavlja obrada, ulaz u funkciju su slika `src`, ulazni događaji `events` (poput pritiska neke tipke ili klika mišem), izlaz su oznake `ann` (grafika ili tekst) koje se stavljaju na sliku za prikaz

`virtual void refmt()` - funkcija kojom se mijenjaju formati slika unutar postupka

`virtual void profile(ui_profileDataProtocol& pd)` - funkcija koja se poziva kada se zaustavlja obrada, u `pd` se stavljaju informacije o vremenu trajanja pojedinih koraka obrade

`virtual void config(cli_wrapProtocol& cli)` - funkcija koja se poziva kada se žele pregledati i izmijeniti parametri postupka

`void dumpParams(std::ostream& os, char const* intro)` - funkcija koja ispisuje trenutne vrijednosti parametara

Klasa: `ip_ccam_server_imp`

Definicija:

```
struct ip_ccam_server_imp{
  ip_ccam_server_imp(const dib_fmt&);
  ~ip_ccam_server_imp();
  void reformat(const dib_fmt&);
  dib_fmt fmt;
  dib_base dib_ccam_server;
  bool get_image;
  bool shot;
  ccam_server cs;
};
```

Opis:

Ova struktura sadrži podatke vezane uz implementaciju postupka servera upravljive kamere.

Podatkovni članovi:

`fmt` - format ulazne slike

`dib_ccam_server` - slika za prikaz

`get_image` - varijabla koja određuje hoće li se slika pribavljati i prikazivati kontinuirano ili samo u trenutku kada serveru stigne zahtjev za uzimanje slike

`shot` - varijabla koja se postavlja na true kada treba uzeti sliku

`cs` - objekt klase ccam_server koji zaprima zahtjeve

Funkcijski članovi:

`ip_ccam_server_imp(const dib_fmt&)` - konstruktor, inicijalizira podatkovne članove

`~ip_ccam_server_imp()` - destruktor, ne radi ništa

`void reformat(const dib_fmt&)` - mijenja format slike

Klasa: `TransienceMap`

Definicija:

```
class TransienceMap
{
private:
  int height, width;
  UCharQueue ***pixelQ;
  int Qsize;
  unsigned char **M;
  int mt_thr, sm_thr;
  int br_thr;
  int size_thr;
public:
  TransienceMap(int h, int w, int qs, int m_t, int s_t, int b_t,
int si_t);
  ~TransienceMap()
  void inputImage(const dib_base& dib_in, util::LayerVector& lv,
dib_base&
                            dib_bkg);
  void extractObjects(const dib_base& dib_in, dib_base& dib_out);
};
```

Opis:

Predstavlja sučelje prema implementaciji postupaka detekcije promjene i segmentacije te postupku za dobivanje slike izdvojenih objekata iznad pozadine.

Podatkovni članovi:

`height`, `width` - visina i širina ulazne slike

`pixelQ` - trostruki pokazivač, koji implementira dvodimenzionalno polje, promjenjivih dimenzija, pokazivača na redove u kojima su sadržane vrijednosti svakog pojedinog slikovnog elementa u prošlim okvirima i trenutnom okviru

`Qsize` - duljina reda za svaki pojedini slikovni element

`M` - dvostruki pokazivač koji implementira polje, promjenjivih dimenzija, u kojem se čuvaju podaci o klasifikaciji svakog pojedinog slikovnog elementa (mapa promjenjivosti)

`mt_thr` - prag okidača pokreta

`sm_thr` - prag mjere stabilnosti

`br_thr` - prag razlike od pozadine

`size_thr` - prag veličine za filter veličine

Funkcijski članovi:

`TransienceMap(int h, int w, int qs, int m_t, int s_t, int b_t, int si_t)` - konstruktor, inicijalizira podatkovne članove, zauzima memoriju za polja

`~TransienceMap()` - destruktor, oslobađa zauzetu memoriju

`void inputImage(const dib_base& dib_in, util::LayerVector& lv, dib_base& dib_bkg)` - kreira objekt klase `TmInputWorker` koji obrađuje ulaznu sliku `dib_in`, korištenjem slike pozadine `dib_bkg`, formira mapu promjenjivosti i provodi grupiranje kako kojim se formira lista regija koje odgovaraju objektima iznad pozadine `lv`

`void extractObjects(const dib_base& dib_in, dib_base& dib_out)` - stvara objekt klase `TmExtractWorker` koji izdvaja slikovne elemente ulazne slike `dib_in` za koje je u mapi promjenjivosti označeno da ne pripadaju pozadini u izlaznu sliku `dib_out`

Klasa: `TmInputWorker`

Definicija:

```
class TmInputWorker{
private:
  unsigned char **M;
  UCharQueue ***pixelQ;
  int height;
  int width;
  int Qsize;
  int br_thr;
  int mt_thr;
  int sm_thr;
  int size_thr;
  util::LayerVector *lv;
private:
  int calcMotionTrigger(UCharQueue& cq);
  int calcStabilityMeasure(UCharQueue& cq);
public:
  typedef const dib_base& Param1;
  typedef dib_base& Param2;
public:
  TmInputWorker(unsigned char **m, UCharQueue ***pQ,
                          int h, int w, int qs,
                        int bt, int mt, int smt,
                        int st, util::LayerVector *layers);
  template <class PixelSrc, class PixelDst>
  inline int operator() (
    const dib_base& dib_in, dib_base& dib_bkg,
    const PixelSrc& dummy1, const PixelDst& dummy2);
};
```

Opis:

Implementira formiranje mape promjenjivosti i puni redove slikovnih elemenata.

Podatkovni članovi:

`M` - pokazivač na mapu promjenjivosti

`pixelQ` - pokazivač na redove vrijednosti slikovnih elemenata

`height`, `width` - visina i širina ulazne slike

`Qsize` - duljina redova slikovnih elemenata

`br_thr` - prag razlike od pozadine

`mt_thr` - prag okidača pokreta

`sm_thr` - prag mjere stabilnosti

`size_thr` - prag veličine filtera veličine

`lv` - lista pronađenih regija koje odgovaraju objektima iznad pozadine

Funkcijski članovi:

`int calcMotionTrigger(UCharQueue& cq)` - računa vrijednost okidača pokreta za pojedini slikovni element čija je prošlost sadržana u redu `cq`

`int calcStabilityMeasure(UCharQueue& cq)` - računa vrijednost mjere stabilnosti za pojedini slikovni element čija je prošlost sadržana u redu `cq`

`TmInputWorker(unsigned char **m, UCharQueue ***pQ, int h, int w, int qs, int bt, int mt, int smt, int st, util::LayerVector *layers)` - konstuktor, inicijalizira podatkovne članove

`inline int operator() (const dib_base& dib_in, dib_base& dib_bkg, const PixelSrc& dummy1, const PixelDst& dummy2)` - implementira postupak obrade, za ulaznu sliku `dib_in`, korištenjem slike pozadine `dib_bkg`, formira mapu promjenjivosti i provodi grupiranje kako kojim se formira lista regija koje odgovaraju objektima iznad pozadine `lv`

Klasa: `TmExtractWorker`

Definicija:

```
class TmExtractWorker{
private:
  unsigned char **M;
  int height;
  int width;
public:
  typedef const dib_base& Param1;
  typedef dib_base& Param2;
public:
  TmExtractWorker(unsigned char **m, int h, int w);

  template <class PixelSrc, class PixelDst>
  inline int operator() (
    const dib_base& dib_in, dib_base& dib_out,
    const PixelSrc& dummy1, const PixelDst& dummy2);
};
```

Opis:

Iz ulazne slike, na temelju mape promjenjivosti, izdvaja slikovne elemente koji pripadaju objektima iznad pozadine u izlaznu sliku.

Podatkovni članovi:

`M` - pokazivač na mapu promjenjivosti,

`height`, `width` - visina i širina ulazne slike

Funkcijski članovi:

`TmExtractWorker(unsigned char **m, int h, int w)` - konstruktor, inicijalizira podatkovne članove

`inline int operator() (const dib_base& dib_in, dib_base& dib_out, const PixelSrc& dummy1, const PixelDst& dummy2)` - izdvaja slikovne elemente ulazne slike `dib_in` za koje je u mapi promjenjivosti označeno da ne pripadaju pozadini u izlaznu sliku `dib_out`

Klasa: `Labeling`

Definicija:

```
class Labeling
{
private:
  bool calculated;
  int next_label;
  int label_no;

  struct Similarity
  {
    Similarity();
    Similarity(int _id, int _sameas);
    Similarity(int _id);
    int id, sameas, tag;
    int number;
  };

  bool is_root_label(int id);
  int root_of(int id);
  bool is_equivalent(int id, int as)
  std::vector<Similarity> labels;
  Labeling(unsigned int soft_maxlabels = 50);
  void init();
  bool setEquivalent(int l1, int l2);
  inline int getNextLabel();
  inline int getNoOfLabels();
  inline int getLabelOccNo(int which);
  inline void incLabel(int which);
  void calculateNewLabelsRegions(int threshold);
  inline int getNewLabel(int old);
};
```

Opis:

Klasa koja služi za postavljanje ekvivalencije između oznaka u procesu sekvencijalnog pronalaženja povezanih komponenti. Također se koristi kao filter veličine jer zanemaruje grupe koje imaju manje elemenata od određenog praga.

Podatkovni članovi:

`calculated` - označava da li su izračunate nove oznake

`next_label` - slijedeća oznaka po redu, broj dosad korištenih oznaka

`label_no` - broj različitih oznaka nakon izračunavanja

`labels` - lista podataka tipa `struct Similarity` koja služi za vođenje podataka o ekvivalenciji oznaka

Funkcijski članovi:

`bool is_root_label(int id)` - funkcija koja vraća istinu ako je promatrana oznaka `id` korijen stabla međusobno ekvivalentnih oznaka

`int root_of(int id)` - funkcija koja vraća korijen stabla oznaka ekvivalentnih oznaci `id`

`bool is_equivalent(int id, int as)` - vraća istinu ako su oznake `id` i `as` ekvivalentne

`Labeling(unsigned int soft_maxlabels = 50)` - konstruktor, inicijalizira podatkovne članove

`void init()` - funkcija za inicijalizaciju objekta, poziva se i iz konstruktora

`bool setEquivalent(int l1, int l2)` - funkcija postavlja ekvivalentnost oznaka `l1` i `l2`

`inline int getNextLabel()` - funkcija dodaje novu oznaku u listu oznaka i vraća novu oznaku

`inline int getNoOfLabels()` - funkcija koja vraća broj oznaka nakon izračunavanja novih oznaka

`inline int getLabelOccNo(int which)` - funkcija vraća ukupan broj pojavljivanja za grupu ekvivalentnih oznaka

`inline void incLabel(int which)` - funkcija koja uvećava broj pojavljivanja oznake `which`

`void calculateNewLabelsRegions(int threshold)` - funkcija provodi izračunavanje novih oznaka tako da svaka grupa ekvivalentnih oznaka (čiji je broj elemenata veći od praga `threshold`) dobiva novu, jedinstvenu oznaku

`inline int getNewLabel(int old)` - funkcija koja za zadanu staru oznaku old vraća novu oznaku dobivenu izračunavanjem.

Klasa: `Layer`

Definicija:

```
class Layer
{
private:
  int _top, _left;
  int _height, _width;
  char **pixmap;
  unsigned char **pixels;
  int statNo, transNo;
  int br_thr;
public:
  int tag;
  Layer();
  Layer(int h, int w);
  Layer(int t, int l, int h, int w);
  Layer(int t, int l, int h, int w, int b_thr);
```

```
  Layer(const Layer& l1);
  ~Layer();
  void setBrThr(int bt);
  void setPixel(int row, int col, unsigned char value, char
type);
  void getPixel(int row, int col, unsigned char& value, char&
type);
  int getStatNo();
  int getTransNo();
  int getPixelNo();
  int getTop();
  int getLeft();
  int getBottom();
  int getRight();
  int getHeight();
  int getWidth();
  int getType(int size_thr = 30);
  inline bool empty();
  bool intersecting(const Layer& l1);
  float intersectionRatio(const Layer& l1);
  void calcRect();
  Layer operator-(const Layer& l1);
  Layer& operator=(const Layer& l1);
  float Similar(const Layer& l1);
  bool SubsetOf(const Layer& l1);
  int getCenterRow();
  int getCenterCol();
  bool inLayer(int r, int c);
  void sizeFilter(int size_thr, LayerVector& v);
};
```

Opis:

Klasa opisuje regiju koja odgovara objektu iznad pozadine. Regija je opisana najmanjim pravokutnikom koji sadrži sve elemente regije, položajem tog pravokutnika u slici, maskom regije koja svaki pojedini slikovni element pravokutnika svrstava u pozadinu ili u regiju (dodatno kao stacionarni ili promjenjivi) i vrijednostima pripadnih slikovnih elemenata. Nadalje, implementira operatore oduzmanja i pridruživanja regija. Prisutne su i funkcije kojima se dobivaju svojstva regija, svojstva međusobnog odnosa regija i odnosa točke i regije.

Podatkovni članovi:

`_top`, `_left` - koordinate gornjeg lijevog ugla u slici regiji opisanog pravokutnika

`_height`, `_width` - visina i širina regiji opisanog pravokutnika

`pixmap` - maska koja opisuje pripadnost regiji pojedinog slikovnog elementa unutar opisanog pravokutnika

`pixels`- polje vrijednosti slikovnih elemenata koji pripadaju regiji

`statNo` - broj stacionarnih slikovnih elemenata regije

`transNo` - broj promjenjivih slikovnih elemenata regije

`br_thr` - prag kojim se utvrđuje različitost slikovnih elemenata dvaju regija prilikom njihovog oduzimanja

`tag` - pomoćna varijabla koja se može koristiti na bilo koji način

Funkcijski članovi:

`Layer()` - podrazumijevani konstruktor, inicijalizira podatkovne članove

`Layer(int h, int w)` - konstruktor, inicijalizira podatkovne članove, zauzima potrebnu memoriju

`Layer(int t, int l, int h, int w)` - konstruktor, inicijalizira podatkovne članove, zauzima potrebnu memoriju

`Layer(int t, int l, int h, int w, int b_thr)` - konstruktor, inicijalizira podatkovne članove, zauzima potrebnu memoriju

`Layer(const Layer& l1)` - konstruktor kopije

`~Layer()` - destruktor, oslobađa zauzetu memoriju

`void setBrThr(int bt)` - postavlja br_thr na vrijednost zadanu argumentom

`void setPixel(int row, int col, unsigned char value, char type)` - postavlja vrijednost i tip (value u polje `pixels`, type u polje `pixmap`) slikovnog elementa regije s koordinatama (u slici, ne regiji) `row` i `col`

`void getPixel(int row, int col, unsigned char& value, char& type)` - za slikovni element s koordinatama (u slici) `row` i `col` vraća njegovu vrijednost u `value` i tip u `type` (vrijednost se nalazi u polju `pixels`, a tip u polju `pixmap`)

`int getStatNo()` - vraća vrijednost podatkovnog člana `statNo`, broj stacionarnih slikovnih elemenata regije

`int getTransNo()` - vraća vrijednost podatkovnog člana `transNo`, broj promjenjivih slikovnih elemenata regije

`int getPixelNo()` - vraća ukupan broj slikovnih elemenata regije

`int getTop()` - vraća vrijednost podatkovnog člana `_top`, koordinatu retka gornjeg lijevog ugla opisanog pravokutnika

`int getLeft()` - vraća vrijednost podatkovnog člana `_left`, koordinatu stupca gornjeg lijevog ugla opisanog pravokutnika

`int getBottom()` - vraća koordinatu retka donjeg desnog ugla opisanog pravokutnika

`int getRight()` - vraća koordinatu stupca donjeg desnog ugla opisanog pravokutnika

`int getHeight()` - vraća vrijednost podatkovnog člana `_height`, visinu opisanog pravokutnika

`int getWidth()` - vraća vrijednost podatkovnog člana `_width`, širinu opisanog pravokutnika

`int getType(int size_thr = 30)` - vraća tip regije koji ovisi o broju promjenjivih i stacionarnih elemenata regije (`size_thr` predstavlja toleranciju)

`inline bool empty()` - vraća istinu ako regija ne sadrži niti jedan slikovni element

`bool intersecting(const Layer& l1)` - vraća istinu ako je presjek opisanih pravokutnika ove regije i regije `l1` različit od praznog skupa

`float intersectionRatio(const Layer& l1)` - vraća vrijednost između 0,0 i 1,0 koja opisuje razinu presjeka regije (0,0 nema presjeka, 1,0 jednake regije)

`void calcRect()` - proračunava novi opisani pravokutnik, zauzima memoriju dovoljnu za prikaz novog pravokutnika i puni njen sadržaj (polja `pixels` i `pixmap`)

`Layer operator-(const Layer& l1)` - od ove regije oduzima regiju `l1` tako da se u rezultirajućoj regiji nalaze svi oni slikovni elementi koji se nalaze u ovoj regiji a ne nalaze se, ili su različiti, u regiji `l1`

`Layer& operator=(const Layer& l1)` - operator pridruživanja, ova regija postaje kopija regile `l1`, vraća se referenca na ovaj objekt

`float Similar(const Layer& l1)` - određuje mjeru sličnosti dviju regija na temelju sličnosti opisanih pravokutnika i sličnosti broja elemenata

`bool SubsetOf(const Layer& l1)` - vraća istinu ako se opisani pravokutnik ove regije može smjestiti unutar opisanog pravokutniga regije `l1`

`int getCenterRow()` - vraća koordinatu retka središta opisanog pravokutnika

`int getCenterCol()` - vraća koordinatu stupca središta opisanog pravokutnika

`bool inLayer(int r, int c)` - vraća istinu ako se točka s koordinatama `r` i `c` nalazi u regiji opisanom pravokutniku

`void sizeFilter(int size_thr, LayerVector& v)` - provodi prostorno grupiranje (tj. pronalaženje povezanih komponenti) slikovnih elemenata regije i primjenjuje filter veličine s pragom veličine `size_thr`, preko argumenta `v` vraća listu pronađenih regija

Klasa: `Queue`

Definicija:

```
template<class T>
class Queue
{
private:
  T *field;
  int max_size;
  int m_front;
  int m_back;
public:
  Queue();
  Queue(int s);
  Queue(const Queue &q);
  ~Queue();
  bool push(const T& el);
  void force_push(const T& el);
  T pop();
  int size();
  T front();
  T back();
  const T& operator[](int n);
};
```

Opis:

Predložak klase koji implementira ograničeni red poljem.

Podatkovni članovi:

`field` - pokazivač koji pokazuje na memorijski prostor gdje se nalazi polje

`max_size` - maksimalna veličina reda

`m_front` - indeks prvog elementa u redu

m_back - indeks poslijednjeg elementa u redu

Funkcijski članovi:

`Queue()` - podrazumijevani konstruktor

`Queue(int s)` - konstruktor, inicijalizira podatkovne članove i rezervira memoriju za pohranjivanje reda

`Queue(const Queue &q)` - konstruktor kopije

`~Queue()` - destruktor, oslobađa memoriju

`bool push(const T& el)` - pokušava dodati `el` u red, vraća `true` ako je akcija uspjela, `false` ako je red već pun

`void force_push(const T& el)` - dodaje `el` na kraj reda i istiskuje element s početka reda ako je red već pun

`T pop()` - izbacuje element s početka reda i vraća njegovu vrijednost

`int size()` - trenutna veličina reda

`T front()` - vraća element na početku reda

`T back()` - vraća element na kraju reda

`const T& operator[](int n)` - vraća `n` - ti element reda

Klasa: `TrackedObject`

Definicija:

```
class TrackedObject : public Layer
{
private:
  static ids;
  int occNo;
  int frameNo;
  int state;
  int occlusion;
  int id;
  int memberNo;
  int memberOf;
  std::vector<int> members;
  int imgHeight, imgWidth;
public:
  bool shot;
  TrackedObject();
  TrackedObject(const Layer la);
  TrackedObject(const TrackedObject& t);
  static inline void resetIds();
  inline void occured();
```

```
  inline void notOccured();
  inline int getFrameNo();
  inline int getOccNo();
  inline void setImgHeight(int h);
  inline void setImgWidth(int w);
  inline int getImgHeight();
  inline int getImgWidth();
  inline int getId();
  inline int getState();
  inline int getOcclusion();
  inline void setOcclusion(int o);
  inline int getMemberNo();
  inline void setMemberOf(int i);
  inline void clearMemberOf();
  inline int getMemberOf();
  inline bool alone();
  inline std::vector<int> getMembers();
  void addMember(TrackedObject& o);
  bool removeMember(TrackedObject& o);
  TrackedObject& operator=(const TrackedObject& t);
  TrackedObject& operator=(const Layer& t);
  void dumpString(char* s);
  void calcState();
  bool valid();
  bool inImgCenter();
};
```

Opis:

Klasa se koristi za opis praćenih objekata. Vodi podatke o regiji objekta (nasljeđuje klasu `Layer`) i o pojavljivanju objekta, stanju objekta, podobjektima i sl.

Podatkovni članovi:

`ids` - statični podatkovni član, sadrži broj instanci ove klase, koristi se prilikom pridjeljivanja identifikacijskog broja objekta

`occNo` - broj okvira u kojima je registrirano pojavljivanje objekta

`frameNo` - broj okvira proteklih od pojave objekta u sceni

`state` - stanje objekta (ulazi, kompletan u sceni, izlazi)

`occlusion` - stanje zaklanjanja (ne koristi se)

`id` - identifikacijski broj objekta

`memberNo` - broj podobjekata (0 - nema ih)

`memberOf` - `id` objekta čiji je ovaj objekt podobjekt (`id` ovog objekta ako nije podobjekt niti jednog objekta)

`members` - lista identifikacijskih brojeva podobjekata

`imgHeight`, `imgWidth` - visina i širina slike

`shot` - varijabla koja govori da li je uzeta slika objekta

Funkcijski članovi:

`TrackedObject()` - podrazumijevani konstruktor

`TrackedObject(const Layer la)` - konstruktor, inicijalizira podatkovne članove

`TrackedObject(const TrackedObject& t)` - konstruktor kopije

`static inline void resetIds()` - postavlja `ids` na nulu

`inline void occured()` - povećava `occNo` i `frameNo` za jedan (ako postoje podobjekti povećava te varijable i njima), označava da se objekt pojavio u sceni

`inline void notOccured()` - povećava `frameNo` za jedan (ako postoje podobjekti povećava ta varijabla se povećava i njima), označava da se objekt nije pojavio u sceni

`inline int getFrameNo()` - vraća vrijednost varijable `frameNo`

`inline int getOccNo()` - vraća vrijednost varijable `occNo`

`inline void setImgHeight(int h)` - postavlja vrijednost varijable `imgHeight` na vrijednost `h`

`inline void setImgWidth(int w)` - postavlja vrijednost varijable `imgWidth` na vrijednost `w`

`inline int getImgHeight()` - vraća vrijednost varijable `imgHeight`

`inline int getImgWidth()` - vraća vrijednost varijable `imgWidth`

`inline int getId()` - vraća `id`

`inline int getState()` - vraća vrijednost varijable `state`

`inline int getOcclusion()` - vraća vrijednost varijable `occlusion`

`inline void setOcclusion(int o)` - postavlja vrijednost varijable occlusion na `o`

`inline int getMemberNo()` - vraća, vrijednost varijable `member_no`, broj podobjekata

`inline void setMemberOf(int i)` - postavlja vrijednost varijable `member_of` na `i`, tj. objekt postaje podobjekt objekta `i`

`inline void clearMemberOf()` - postavlja vrijednost varijable `member_of` na vlastiti `id`, prestaje biti ičiji podobjekt

`inline int getMemberOf()` - vraća vrijednost varijable `member_of`, `id` objekta čiji je podobjekt

`inline bool alone()` - vraća `true` ako objekt nije ničiji podobjekt, `false` inače

`inline std::vector<int> getMembers()` - vraća listu identifikacijskih brojeva podobjekata ovog objekta

`void addMember(TrackedObject& o)` - dodaje se podobjekt objektu

`bool removeMember(TrackedObject& o)` - uklanja podobjekt iz objekta

`TrackedObject& operator=(const TrackedObject& t)` - ovaj objekt postaje kopija objekta `t`

`TrackedObject& operator=(const Layer& t)` - regija ovog objekta postaje kopija regije `t`

`void dumpString(char* s)` - u znakovni niz `s` formatirano ispisuje `id` objekta ili, ako ih ima, identifikacijske brojeve podobjekata, i njegovo stanje

`void calcState()` - na temelju položaja opisanog pravokutnika postavlja varijablu `state`

`bool valid()` - vraća istinu ako je objekt valjan, tj. ako je broj pojavljivanja `occNo` dovoljno velik u odnosu na broj okvira `frameNo` otkako se pojavio

`bool inImgCenter()` - vraća istinu ako opisani pravokutnik objekta obuhvaća središte slike

Konfiguracije

Postupak izlučivanja slike vozila razlikuje se za pojedine konfiguracije, a time se razlikuje i programska implementacija za svaku od njih. Program za pojedinu konfiguraciju stvara se definiranjem predprocesorskih direktiva. Te direktive su (nalaze se na početku datoteke `ip_car_track_cpp`):

1. `#define STATIONARY_CAM` - za konfiguraciju koja koristi stacionarnu kameru;
2. `#define LOCAL_ZOOM` - za konfiguraciju koja koristi upravljivu kameru;
3. `#define REMOTE_ZOOM` - za konfiguraciju koja koristi dvije kamere.

Prilikom prevođenja samo jedna od ovih direktiva smije biti definirana.

Klase `ip_ccam_server` i `ip_ccam_server_imp` implementiraju postupak koji se koristi za treću konfiguraciju. U tom slučaju pokreću se dva programa `cvsh`, jedan s postupkom `ccam_server`, a druga s postupkom `car_track` (o tome više riječi u prilogu B koji daje upute za rad).

Način rada

Izvedba postupka može se podijeliti na dva dijela, svaki odgovara glavnim fazama rada:

1. segmentacija scene i praćenje;
2. izlučivanje slike vozila.

Faza segmentacije i praćenja jednaka je za sve tri konfiguracije sustava. Može se prikazati slijedećim koracima:

1. *Pribavi sliku*. Korak obavlja dio ljuske zadužen za pribavljanje slike. Korak završava pozivom funkcijskog člana `ip_car_track.process(..)` koji pomoću svog argumenta prenosi ulaznu sliku na obradu.
2. *Smanji sliku*. Kako bi se smanjila računska složenost, odnosno povećala brzina obrade ulazna slika se može smanjiti. Koristi se funkcija void `void subsample( const dib_base& src, int dimension, dib_base& dst)` koja smanjuje ulaznu sliku `src` za `dimension` čime se dobiva smanjena slika `dst`.
3. *Formiraj mapu promjenjivosti i pronađi regije koje odgovaraju objektima iznad pozadine*. Ulazna slika se dodaje u red u kojem se pamti prošlost te se na temelju ulazne slike, prošlih slika i slika pozadine formira mapa promjenjivosti. Nakon toga se korištenjem podataka iz mape promjenjivosti provodi grupiranje slikovnih elemenata i pronalaženje regija koje odgovaraju

objektima iznad pozadine. Korak odgovara pozivu funkcijskog člana `TransienceMap.inputImage(..)`.

4. *Na temelju novih regija i regija koje odgovaraju zaustavljenim objektima odredi nove regije objekata*.
5. *Za svaki praćeni objekt pronađi novu regiju*. Na temelju sličnosti regija praćenih objekata i novih regija pronalaze se odgovarajuća nova regija za svaki objekt. Pronalaze se i rješavaju slučajevi spajanja regija uslijed zaklanjanja.
6. *Ukoliko postoji objekt koji je u potpunosti u sceni, koji se zaustavio i koji ne sudjeluje u zaklanjanju prijeđi na fazu izlučivanja slike*.

Faza izlučivanja slike vozila razlikuje se za pojedine konfiguracije sustava.

Konfiguracija koja koristi stacionarnu kameru implementira ovu fazu jednostavnim izrezivanjem slike vozila iz scene. Koristi se funkcija `void dib_access_copyWithOffset(int from_x, int from_y, const dib_base& src, dib_base& dst)` koja kopira dio ulazne slike `src`, s pomacima `from_x` i `from_y`, u izlaznu sliku čiji je format veličine vozilu opisanog pravokutnika.

Kod konfiguracije koja koristi upravljivu kameru, ova faza je izvedena tako da se upravljivoj kameri zadaju naredbe zauzimanja kuta zakreta, nagiba i zuma, kako bi se u slici nalazilo samo vozilo.

Za konfiguraciju koja koristi dvije kamere koriste se dva programa s pokrenutim postupcima `ccam_server` i `car_track`. U postupku `car_track`, se preko objekta klase `ccam_client` predaje zahtjev postupku `ccam_server`, kojeg zaprima objekt klase `ccam_server`, za postavljanjem kuta zakreta, nagiba i zooma kako bi se u slici koju daje kamera vezana uz taj postupak (program) nalazilo samo vozilo. Kada server signalizira klijentu da je obradio zahtjeve, odnosno postavio tražene parametre upravljive kamere, postupak `car_track` signalizira, opet putem veze objekata klasa `ccam_client` i `ccam_server`, postupku `ccam_server` da uzme sliku sa kamere (i prikaže ju).

Platforme

Programi su napisani za operacijski sustav *MS Windows* (98/2000/XP). Korišten je prevodioc *MSVC* 6.0.

Prilog B:
Upute za korištenje

Ovdje će biti dane upute za instalaciju potrebnih programa, spajanje potrebne opreme, pokretanje i i korištenje programa za sustave SAPPO i SIVS. Pošto oba sustava koriste ljusku CVSH, u posebnom poglavlju dane su upute za njeno korištenje.

Sustav za aktivno praćenje pokretnih objekata - SAPPO

Instalacija

Verzija koja implementira aktivno praćenje

Ova verzija napravljena je za operacijski sustav *RedHat Linux 7.2*, custom kernel 2.4.18 sa slijedećom programskom i sklopovskom opremom (minimalna konfiguracija):

- kamera: *Basler a301f*, digitalna kamera na sabirnici *IEEE 1394*;
- upravljivo postolje: *Directed Perception PTU-46-17.5*;
- pribavljanje slike: *PCI* kartica, sučelje za sabirnicu *IEEE 1394*, standardno programsko sučelje;
- računalo: 2×*AMD Athlon Palomino MP* @ 1.4 GHz.

Na računalo se, u bilo koji direktorij, jednostavno kopira datoteka izvršne verzije programa `cvsh`, koja se nalazi u direktoriju `/linux/` priloženog CD-a. U istom direktoriju se nalazi i skripta `invoke.sh` koja omogućuje poziv programa bez navođenja parametara komandne linije.

Verzija koja samo prikazuje postupak praćenja

Ova verzija napravljena je za operacijski sustav *MS Windows* (98/2000/XP) i može raditi praktički na svakom osobnom računalu (pošto samo za prikaz nije nužan rad u stvarnom vremenu).

Instalacija se sastoji samo od kopiranja izvršne verzije programa `cvsh.exe`, koja se nalazi u direktoriju `\win\show\` priloženog CD-a, u bilo koji direktorij.

Spajanje opreme

Ovaj dio se odnosi samo na verziju koja implementira aktivno praćenje.

Spajanje kamere

Potrebno je spojiti jedan kraj kabela za sabirnicu *IEEE 1394* u kameru, a drugi kraj bilo koju od utičnica *PCI* kartice, sučelja za sabirnicu *IEEE 1394*. Nakon toga

potrebno je iz komandne linije pokrenuti program `gscanbus` i odabrati opciju *force bus reset*. Nakon toga kamera je spremna za rad.

Spajanje upravljivog postolja

Upravljivo postolje se s računalom povezuje korištenjem serijskog porta 0.

Upute za korištenje

Verzija koja implementira aktivno praćenje

Kada je oprema ispravno spojena, sustav se stavlja u pogon pokretanjem programa `cvsh` na slijedeći način:

```
./cvsh -sg=unix1394 -a=track -i
```

ili korištenjem skripte:

```
./invoke.sh
```

Nakon toga se pokreće program, ljuska CVSH s postupkom `track`. Pozivom na ovaj način ulazi se u interaktivni mod ljuske. U komandnoj liniji ljuske potrebno je upisati naredbu

```
p n -1
```

kojom se pokreće obrada. Obrada se može u svakom trenutku prekinuti pritiskom tipke `<Esc>` dok je fokus na prozoru koji prikazuje sliku praćenja, nakon čega se ponovo ulazi u komandnu liniju ljuske. Iz komandne linije se izlazi izdavanjem naredbe `q`.

Ovdje su opisane samo osnovne upute za korištenje programa. Detaljniji opis naredbi ljuske dan je u posebnom poglavlju ovog priloga.

Verzija koja samo prikazuje postupak praćenja

Za ovu verziju potrebno je imati datoteke u kojima se nalaze već snimljene sekvence (tipa `.srs` ili `.avi`). Datoteke sekvenci se nalaze u direktoriju `\sekvence\` priloženog CD-a.

Primjer poziva programa za datoteku `setnja.srs`:

```
cvsh.exe -sf=setnja.srs -a=track_show -i
```

U slučaju da se datoteka `setnja.srs` ne nalazi u istom direktoriju kao i program, potrebno je navesti cijelu stazu do datoteke.

Nakon poziva, pokreće se program, ljuska CVSH u interaktivnom modu. U komandnoj liniji može se pozvati naredba:

```
p n -1
```

Ova naredba će obraditi cijelu sekvencu, nakon čega će se vratiti na početak sekvence. Naredbom:

```
p n 1
```

može se redom obrađivati po jedan okvir sekvence.

Ovdje su opisane samo osnovne upute za korištenje programa. Detaljniji opis naredbi ljuske dan je u posebnom poglavlju ovog priloga.

Parametri postupka

Način postavljanja parametara postupka dan je u poglavlju uputa za korištenje ljuske CVSH. Ovdje je dan opis parametara postupka.

parametar	naziv u ljusci	tip
prag sivih razina SUSAN detektora uglova, *t*	`bright_thr`	`int`
prag eliminacije pogrešnih aktivnih regija, *r*	`edge_number`	`float`
prag diferencije slika, τ	`diff_thr`	`int`
prag veličine, *N*	`size_thr`	`int`
prag udaljenosti za grupiranje, *d*	`distance`	`int`
smanivanje ulazne slike po svakoj dimenziji	`dim`	`int`
iscrtavanje pomoćnog prozora	`draw`	`bool`
iscrtavanje vektora pomaka uglova ili samo uglova	`draw vectors`	`bool`

Sustav za izlučivanje slike vozila iz scene - SIVS

Instalacija

Konfiguracija koja koristi statičnu kameru

Ova verzija programa namijenjena je prvenstveno prikazu rada sustava na snimljenim sekvencama. Ipak, ukoliko se kao ulaz u program navede digitalizator program će raditi kao konfiguracija koja koristi statičnu kameru.Program je napravljen za operacijski sustav *MS Windows* (98/2000/XP) i može raditi praktički na svakom osobnom računalu (pošto samo za prikaz nije nužan rad u stvarnom vremenu).

Ukoliko se koristi kamera, potrebno je imati kameru koja daje analogni izlaz po *PAL* standardu i opremu za pribavljanje slike *Imagenation PXC 200 AL* (PCI kartica, digitalizator PAL signala, posebno sučelje).

Instalacija se sastoji samo od kopiranja izvršne verzije programa cvsh.exe, koja se nalazi u direktoriju `\win\show\` priloženog CD-a, u bilo koji direktorij.

Konfiguracija koja koristi upravljivu kameru

Ova verzija programa napisana je za posebnu opremu, koja je korištena za razvoj. Potrebna je minimalna konfiguracija:

- upravljiva kamera: *Sony EVI-D31*, analogna kamera s video izlazom po *PAL* standardu s integriranim upravljivim postoljem;
- pribavljanje slike: *Imagenation PXC 200 AL*, PCI kartica, digitalizator PAL signala, posebno sučelje;

- računalo: *Pentium 4* @ 2 GHz;
- operacijski sustav: *MS Windows 2000*.

Instalacija se sastoji samo od kopiranja izvršne verzije programa `cvsh.exe`, koja se nalazi u direktoriju `\win\1cam\` priloženog CD-a, u bilo koji direktorij.

Konfiguracija koja koristi dvije kamere

Ova verzija programa napisana je za posebnu opremu, koja je korištena za razvoj. Potrebna je minimalna konfiguracija:

- 2 × upravljiva kamera: *Sony EVI-D31*, analogna kamera s video izlazom po *PAL* standardu s integriranim upravljivim postoljem;
- 2 × pribavljanje slike: *Imagenation PXC 200 AL*, PCI kartica, digitalizator PAL signala, posebno sučelje;
- računalo: *Pentium 4* @ 2 GHz;
- operacijski sustav: *MS Windows 2000*.

Instalacija se sastoji samo od kopiranja izvršne verzije programa `cvsh.exe`, koja se nalazi u direktoriju `\win\2cam\` priloženog CD-a, u bilo koji direktorij.

Spajanje opreme

Konfiguracija koja koristi statičnu kameru

Potrebno je spojiti video izlaz kamere na ulaz digitalizatora (npr. *S-VIDEO* kablom).

Konfiguracija koja koristi upravljivu kameru

Potrebno je spojiti video izlaz kamere na ulaz digitalizatora pomoću *S-VIDEO* kabela. Upravljivo postolje spaja se na računalo pomoću *VISCA* kabela na serijski port 0.

Konfiguracija koja koristi upravljivu kameru

Svaka kamera se spaja na jedan digitalizator pomoću *S-VIDEO* kabela. Prva kamera, koja se koristi kao statična, s računalom se povezuje pomoću *VISCA* kabela na serijski port 0. Druga kamera se s računalom povezuje pomoću *VISCA* kabela na serijski port 1. Kamere se moraju staviti u ravnini jedna do druge, s paralelnim nultim položajima.

Upute za korištenje

Konfiguracija koja koristi statičnu kameru

Kako je već rečeno ova verzija se može koristiti s kamerom i digitalizatorom ili s datotekama snimljenih sekvenci.

Ukoliko se koristi ulaz iz digitalizatora (čiji je redni broj 0), program se poziva na slijedeći način:

```
cvsh.exe -sg=win32PXC@0.1:320x240 -a=car_track -i
```

Ukoliko se koristi datoteka, npr. `parking2.srs`, poziv programa treba biti slijedeći:

```
cvsh.exe -sf=parking2.srs -a=car_track -i
```

Nakon poziva, pokreće se program, ljuska CVSH, u interaktivnom modu. U komandnoj liniji može se pozvati naredba:

```
p n -1
```

Ova naredba će kontinuirano obrađivat okvire pristigle iz digitalizatora, odnosno cijelu sekvencu, nakon čega će se vratiti na početak sekvence. Ako se koristi ulaz iz datoteke, ima smisla koristiti naredbu:

```
p n 1
```

kojom se obrađuje po jedan okvir.

Ovdje su opisane samo osnovne upute za korištenje programa. Detaljniji opis naredbi ljuske dan je u posebnom poglavlju ovog priloga.

Konfiguracija koja koristi upravljivu kameru

Za ovu konfiguraciju vrijedi isto što i za prethodnu, uz iznimku da nema smisla koristiti ulaz iz sekvence.

Kada se pokrene program, sustav se nalazi u stanju u kojem se ne provodi obrada. U tom stanju se mogu podešavati parametri upravljive kamere. Podešavanje se provodi dok je fokusiran glavni prozor za prikaz. Podešavati se može na slijedeće načine:

- podešavanje kuta zakreta i nagiba: drži se pritisnuta tipka `<Shift>` i pritom se klikne na neku točku slike; podešava se kut zakreta i nagiba tako da se ta točke nalazi u središtu slike;
- podešavanje fokusa: pritiskom na tipku `0` fokus se podešava na veću daljinu, a pritiskom na tipku `1` na manju daljinu;
- uključivanje i isključivanje automatskog fokusa: pritiskom na tipku `A` uključuje se i isključuje opcija automatskog fokusa;
- uključivanje i isljučivanje prikaza križića na sredini slike: pritiskom na tipku `C` uključuje se i isključuje prikaz križića na sredini slike;
- podešavanje zooma: pritiskom na tipku `+` smanjuje se širina vidnog polja, a pritiskom na tipku `–` širina vidnog polja se povećava;

Postupak obrade uključuje se i isključuje pritiskom tipke `P`.

Konfiguracija koja koristi dvije kamere

Za ovu konfiguraciju potrebno je pokrenuti dva programa.

Prvi program je ljuska s postupkom obrade `ccam_server`. Program se poziva na slijedeći način (ukoliko je upravljiva kamera koja je spojena na serijski port 1 spojena na digitalizator 1):

```
cvsh.exe -sg=win32PXC@1.1:320x240  -a=ccam_server
```

Na taj način poziva se ljuska u neinteraktivnom modu. Za ovaj program može se podešavati hoće li se slika prikazivati kontinuirano ili samo kad postupak obrade iz drugog programa izda naredbu za uzimanje slike. Opcije se izmjenjuju pritiskom na tipku I (dok je fokus na prozoru za prikaz slike).

Drugi program je ljuska s postupkom obrade `car_track`. Za ovaj program vrijedi isto kao i za konfiguraciju koja koristi samo jednu upravljivu kameru s jednom dodatnom opcijom. Ukoliko je program u stanju u kojem se ne provodi obrada,

može se upravljati zakretom i nagibom druge kamere tako da se drži pritisnuta tipka <Ctrl> i klikne na neku toču slike. Druga kamera se tada postavlja u položaj tako da joj se ta točka nalazi u središtu slike.

Parametri postupka

Način postavljanja parametara postupka dan je u poglavlju uputa za korištenje ljuske CVSH. Ovdje je dan opis parametara postupka.

parametar	naziv u ljusci	tip
prag okidača pokreta, *m*	`height threshold`	`int`
prag mjere stabilnosti, *s*	`similarity measure threshold`	`int`
prag razlike od pozadine, τ	`brightness threshold`	`int`
prag veličine, *N*	`size threshold`	`int`
broj prošlih okvira koji se pamte, *q*	`queue size`	`int`
prag visine objekta	`height threshold`	`float`
prag širine objekta	`width threshold`	`float`
smanivanje ulazne slike po svakoj dimenziji	`subsample dimension`	`int`
iscrtavanje oznaka u boji (samo za konfiguraciju korišenjem stacionarne kamere)	`annotation color`	`bool`

Ljuska CVSH

Poziv programa

Ljuska omogućuje dva temeljna načina rada: neinteraktivni i interaktivni. U slučaju neinteraktivnog rada, parametri komandne linije u potpunosti određuju tijek izvođenja, dok u slučaju interaktivnog rada, samo definiraju početne vrijednosti koje se kasnije mogu promijeniti.

Komandnom linijom može se utjecati na slijedeće parametre izvođenja programa:

1. izvor slijeda slika (opcija `-s`);
2. postupak obrade (opcija `-a`);
3. konfiguracijski niz (opcija `-c`);
4. ograničavanje brzine obrade (opcija `-m`).

Kod interaktivnod rada, moguće je dodatno specificirati početnu naredbu, dok se kod neinteraktivnog rada mogu navesti još jedan ili više odredišta slijeda slika. Sasvim općenito, program se može pozvati upotrebom jednog od slijedećih sintaksnih oblika.

1. Neinteraktivni rad:

   ```
   cvsh -s<Source>[#<Range>] [-a=<Algorithm>] [-c=<ConfigString>] [-d<Dest> ...]
   ```

2. Interaktivni rad:

   ```
   cvsh -s<Source>[#<Range>] [-a=<Algorithm>] [-c=<ConfigString>] -i[=<CommandString]
   ```

Sintaksa pojedinih opcija opisana je u nastavku. Kraći opis sintakse i opcija može se dobiti pozivanjem programa bez argumenata.

Opcija `-s<Source>[#Range]`

Specificira se izvor slijeda slika `<Source>`, te interval slika `<Range>` koji se želi obraditi. Općenito, izvor slika može biti datoteka ili digitalizator slike pa prema tome sintaksa tijela opcije `<Source>` može imati jedan od slijedećih oblika:

1. `f=filename`, gdje `filename` predstavlja valjanu stazu do željene izvorišne datoteke (dozvoljeni formati datoteka dani su u jednom od slijedećih poglavlja);
2. `g=name[@board[.channel[.mode]][:wxh[xszpix]]]`, gdje je `name` ime digitalizatora (npr. `unix1394` ili `win32PXC`), `board` broj ploče, `channel` broj ulaznog kanala, mode način rada, `wxh` format ulazne slike (širina `x` visina), a `szpix` broj bajta po pixelu (`x1` - siva slika, `x3` slika u boji).

Interval slika `<Range>` se uzima u obzir samo ako je izvor slike datoteka i služi za ograničavanje obrade na slike datoteka čiji su redni brojevi sadržani u intervalu. `<Range>` ima sintaksu `#From-To`, gdje su `From` i `To` redni brojevi prve, odnosno neposrednog sljedbenika poslijednje slike iz intervala kojeg je potrebno obraditi.

Opcija `-a=<Algorithm>`

Znakovni niz `<Algorithm>` definira postupak obrade. Moguće je odabrati jedan od ugrađenih postupaka obrade (npr. `track`, `track_show`, `car_track` ...) ili valjanu putanju izvršne datoteke s vanjskim postupkom. U poslijednjem slučaju, izvršna datoteka mora biti biblioteka predviđena za dinamičko povezivanje, te sadržavati standardne funkcije sučelja čiji prototipovi su dokumentirani u izvornoj datoteci `ip_dll.cpp`.

Opcija `-c=<ConfigString>`

Specificira se konfiguracijski niz za postupak određen opcijom `-a`.

Opcija `-d<Dest>`

Specificira se odredište slijeda slika `<Dest>` koje, općenito, može biti datoteka ili prikazna jedinica pa prema tome sintaksa tijela opcije `<Source>` može imati jedan od slijedećih oblika:

1. `f=filename`, gdje filename predstavlja valjanu putanju do željene odredišne datoteke (dozvoljeni formati datoteka dani su u jednom od slijedećih poglavlja);
2. `w`, odredište je prikazna jedinica.

Opcija `-i[=<CommandString>]`

Specificira se interaktivni način rada i, eventualno, prva interaktivna naredba. Popis interaktivnih naredbi dan je u jednom od slijedećih poglavlja.

Formati ulaznih datoteka

Kod opcija `-sf=...` i `-df=...`, ekstenzija specificirane datoteke mora odgovarati njenom formatu, kao što slijedi:

- ekstenzija `.avi`: datoteka je u *Microsoftovom AVI* formatu (format nije podržan na ne-*Microsoftovim* operacijskim sustavima);
- ekstenzija `.srs`: datoteka se sastoji od slijeda slika u nekomprimiranom *Sun raster* formatu;
- ekstenzije `.ppm`, `.bmp`, `.sr`: datoteka se sastoji od jedne slike u *portable pixmap*, *Microsoft bmp*, odnostno *Sun raster* formatu;
- ekstenzija `.txt`: datoteka je tekstualna i specificira više nizova slika iz drugih datoteka.

Interaktivne naredbe ljuske

Ako je u komandnoj liniji specificiran interaktivni način rada (opcija `-i`), ljuska prilikom pokretanja izvršava početnu naredbu (ako je zadana) i prepušta daljnje upravljanje korisniku, preko tekstualnog korisničkog sučelja. Naredbe koje sučelje podržava mogu se svrstati u tri skupine:

- naredbe za podešavanje parametara izvođenja (`algorithm`, `source` i `config`);
- naredbe za prikaz i obradu slika ulaznog slijeda (`process` i `show`);
- upravljačke naredbe (`help` i `quit`).

Naredbe iz prve skupine utječu na parametre čije se početne vrijednosti zadaju iz komandne linije (izvor slijeda slika, postupak obrade i konfiguracija postupka obrade). Sve naredbe je moguće skraćivati; u slučaju višeznačne kratice, ljuska odabire prvu naredbu prema leksikografskom uređaju. Tako su i `proc` i `p` valjani oblici naredbe `process`, `s` određuje naredbu `show`, a minimalni zapis naredbe `source` je `so`. Slijedi opis dozvoljenih sintaksnih oblika pojedinih naredbi.

Naredba `source`

Naredba omogućava promjenu izvora slijeda slika. Sintaksa naredbe je `source <Source>`, gdje `<Source>` može poprimiti iste oblike kao i kod `-s` opcije komandne linije.

Naredba `algorithm`

Ovom naredbom se može specificirati drugi postupak obrade. Sintaksa naredbe je `algorithm <Algorithm>`, gdje `<Algorithm>` može poprimiti iste oblike kao i kod `-a` opcije komandne linije.

Naredba `config`

Naredba omogućava promjenu parametara postupka obrade. Sintaksa naredbe je `config [<ConfigString>]`, gdje je `<ConfigString>` konfiguracijski niz i ovisi o postupku obrade. Ako konfiguracijski niz nije zadan, konfiguracija postupka se obavlja interaktivno.

Naredba `process`

Naredbom se pokreće postupak obrade slika iz izvornog slijeda. Interval koji je potrebno obraditi određen je argumentima naredbe i može biti apsolutan ili relativan u odnosu na poziciju poslijednje obrađene slike iz slijeda *SSP*. Sintaksa

naredbe je process <Location>, a podržani formati argumenta <Location> sažeti su u slijedećoj tablici:

oblik naredbe	interval obrade
next [<howMany>]	[SSP+1, SSP+<howMany>+1>
previous [<howMany>]	[SSP - <howMany>, SSP> (unatrag!)
this [<howMany>]	[SSP, SSP+1> (<howMany> ponavljanja)
address <frame>	[<frame>, <frame>+1]

Podrazumijevana vrijednost za [<howMany>] je 1. Ukoliko je neka od granica dobivenog intervala obrade manja od nule ili veća od broja slika izvornog slijeda, ona se svodi na dozvoljenu vrijednost operacijom modulo n_{IS}, gdje je n_{IS} broj slika izvornog slijeda. Specijalni slučaj parametra <howMany> je vrijednost -1, tada se kao vrijednost parametra podrazumijeva n_{IS}. Tako je naredbom process next -1 moguće postići obradu svih slika iz slijeda točno jednom. Izvođenje neke druge naredbe može se opozvati pritiskom tipke <Esc> u odredišnom prozoru ljuske, tj. u prozoru čiji naslov sadržava naziv izvora slijeda slika i brzinu obrade. Tipka <Esc> se može koristiti i za prekidanje neinteraktivne obrade.

Ključne riječi next, previous, this i address se mogu skraćivati kao i naredbe ljuske. Kod izvora slika bez mogućnosti pozicioniranja (npr. digitalizator slike), naredbe s ključnim riječima previous, this i address se interpretiraju kao i odgovarajuća naredba s ključnom riječi next. Numerički parametar [<howMany>] kod riječi this označava koliko puta za redom će se obraditi tekuća slika iz slijeda. To može biti korišteno sa svrhom preciznog mjerenja trajanja izvođenja postupka obrade.

Naredba `show`

Naredba ima istu sintaksu kao i naredba process, a razlikuje se od nje samo u tome što se slike iz intervala izvornog slijeda prikazuju bez obrade.

Naredba `help`

Ovom naredbom se ispisuje sažeti popis interaktivnih naredbi ljuske.

Naredba `quit`

Naredbom se prekida rad programa i vraća upravljanje ljusci operacijskog sustava.

Prilog C:
Korišteni radovi

U stranicama koje slijede priložen je jedan od korištenih radova.

Ostali korišteni radovi nalaze se, u elektroničkom obliku, na CD-u priloženom uz rad.