SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 871

SUSTAV ZA UPRAVLJANJE AUTOMATIZIRANIM POSTOLJEM TELESKOPA U SVRHU PRAĆENJA ZVIJEZDA

Marko Vraničar

Zagreb, lipanj 2013.

Diplomski zadatak :

Sustav za upravljanje automatiziranim postoljem teleskopa u svrhu praćenja zvijezda

U okviru diplomskog rada potrebno je razviti sustav za automatsko usmjeravanje astronomskog teleskopa na zadanu zvijezdu i njeno automatsko praćenje tijekom procesa oslikavanja. Informacija o trenutnom pogledu, odnosno o pomaku pogleda zbog pogreški vođenja u praćenju po osi desnog uzdizanja i osi deklinacije potrebno je odrediti iz pomoćnog sustava oslikavanja koji je optički kolimiran s glavnim sustavom. Istražiti mogućnosti korištenja Web kamere ili jednostavnijeg digitalnog fotoaparata za akviziciju slike iz pomoćnog sustava. Razviti algoritam za automatsko određivanje pozicija zvijezda u slici, pomaka njihovih pozicija između dva susjedna vremenska okvira, te određivanje komande za korekciju pogrešaka u praćenju. Sustav temeljiti na ugradbenom računalu Beagleboard, a programsku potporu razviti korištenjem odgovarajućih razvojnih alata (C/C++, Java, Matlab) za rad pod operacijskim sustavom Linux. Za dodatne informacije obratiti se mentoru.

Zahvaljujem se prof. dr. sc. Davoru Petrinoviću na savjetima i mentorstvu

Sadržaj

Uvod

Cilj diplomskog rada je razviti pomoćni sustav za automatsko usmjeravanje astronomskog teleskopa na zadani dio neba i automatsko praćenje istog. Općenito se očekuje da se prate svijetli objekti na tamnoj pozadini, a to je najčešće slučaj kod astrofotografije. Kod astrofotografije se podrazumijeva da se nebo promatra noću, dok su vidljive zvijezde i ostali nebeski objekti prema kojima se sustav može referencirati. Kompletni sustav se sastoji od dva podsustava, glavnog i pomoćnog, koji su optički kolimirani. Sustavi rade neovisno jedan o drugome, no konačni rezultat značajno ovisi o radu i performansama oba.

Zadatak glavnog sustava je snimanje neba uz što manji šum, izobličenja i vanjskih smetnji. To uključuje različito filtriranje slike, obradu i kombiniranje sekvence slika radi postizanja što bolje kvalitete. Dakle naglasak kod glavnog sustava je dobivanje što kvalitetnije slike.

Zadatak pomoćnog sustava je da na temelju informacija o trenutnom pogledu, odnosno o pomaku pogleda zbog pogreški vođenja u praćenju po osi desnog uzdizanja i osi deklinacije, korigira pomak promatranih nebeskih objekata tako da glavni sustav može bez većih odstupanja pratiti određeni dio neba. Također ovakav sustav može poslužiti i kao glavni sustav za praćenje neba, no kvaliteta konačne slike bi u tom slučaju bila lošija zbog samog principa rada. U prvom redu to podrazumijeva praćenje određenog dijela neba, a ne snimanje što kvalitetnijih fotografija.

Kod implementacije pomoćnog sustava u radu se koristi Beagleboard razvojni sustav sa OMAP3530 procesorom za obradu slike i kompenzaciju pomaka. U svrhu hvatanja slike se koristi web kamera sa mogućnošću mijenjanja vremena aktivnosti senzora (ekspozicije). Dodatno rad istražuje različite metode izračuna potrebnih parametara što uključuje DSP jezgru na sustavu i frakcionalnu aritmetiku te uspoređuje performanse samog sustava kod različitih metoda.

1. Astrofotografija

Astrofotografija je vrsta fotografije koja se konkretno bavi fotografiranjem neba i nebeskih objekata kao što su planeti, zvijezde i slično. Njome se bave astronomi i spada pod astronomiju, pa joj odatle i ime. Postoji više metoda snimanja astronomskih fotografija, no koja god metoda se koristila, svim fotografijama je zajedničko da se na njima nalaze zabilježeni objekti i pojave koje se događaju u svemiru. U većini slučajeva se podrazumijeva da su na fotografijama snimke vidljivog dijela spektra neba, no ponekad je poželjno zabilježiti i dio spektra izvan vidljivog. U prvom redu to uključuje infracrveni i ultraljubičasti dio koji su na fotografijama pomaknuti i prikazani kao vidljiv dio spektra.

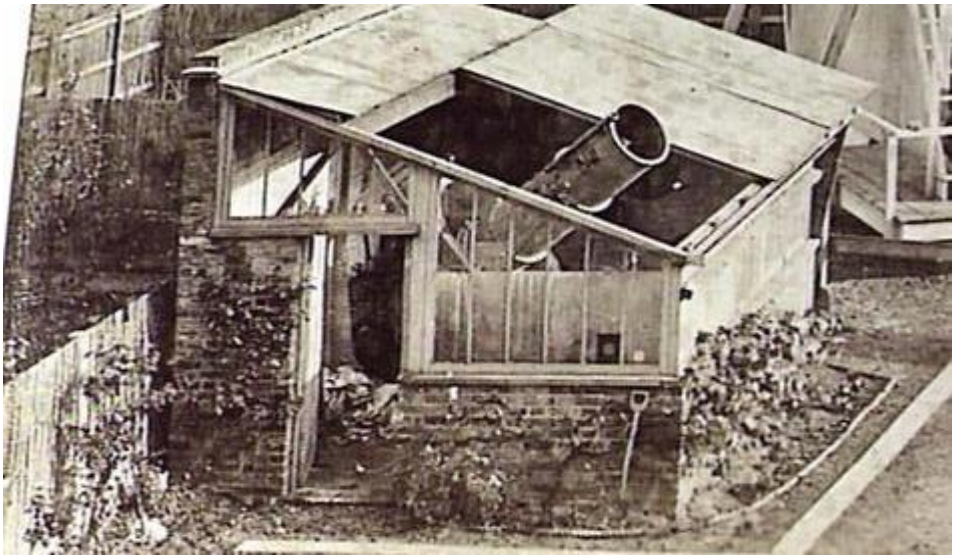
Kad se gleda fotografija u znanosti, tada je astrofotografija jedna od najstarijih vrsta fotografije korištene u znanstvene svrhe. Gotovo od samog početka je podijeljena u poddiscipline koja svaka ima određeni cilj uključujući kategorizaciju i klasifikacije zvijezda, kartografiju, astrometriju, fotometriju, spektroskopiju, polarometriju te otkriće novih objekata kao što asteroidi, meteori, kometi pa čak i nepoznati planeti. Sve te zadaće zahtijevaju specijaliziranu opremu poput teleskopa namijenjenih za precizno snimanje sa širokim vidnim poljem ili za snimanje samo određenih valnih duljina svjetla za što se koriste specijalizirani optički filtri. Korišteni CCD senzori čak mogu biti dodatno hlađeni radi smanjenja toplinskog šuma i što boljeg snimanja visokokvalitetnih slika. Hlađenje također omogućuje detektiranje različitih dijelova spektra poput infracrvenog i ultraljubičastog.

1.1. Povijest astrofotografije

Astrofotografija se prvi puta pojavila sredinom 19. Stoljeća, što su bile najvećim dijelom eksperimentalne fotografije neba noću. Zbog vrlo dugog vremena snimanja fotografije radi hvatanja relativno slabo osvijetljenih objekata, bilo je potrebno prevladati mnogo tehnoloških problema. To je uključivalo izradu teleskopa koji ne bi izašao iz fokusa tijekom snimanja objekata, izrada mehanizama kojim bi se mogao rotirati teleskop konstantnom brzinom i razvoj metoda kako bi teleskop bio usmjeren prema jednoj fiksnoj točki na nebu tijekom vremena snimanja fotografije. Rane metode razvijanja samih fotografija su također imala svoja ograničenja.

Prvi poznati pokušaj snimanja astronomske fotografije zbio se 1839. godine dok je Louis Jacques Daguerre Mandé pokušao fotografirati mjesec. Zbog nepreciznog praćenja

Mjeseca, odnosno vođenju teleskopa tijekom dugog izlaganja zračenju objekta, konačni rezultat je bila nejasna fotografija na kojoj se mogla vidjeti zamućena mrlja. John William Draper, profesor kemije na Sveučilištu u New York-u, liječnik i znanstveni istraživač, uspio je kasnije dana 23. ožujka 1840. prvi put uspješno snimiti fotografiju Mjeseca uz vrijeme izloženosti od 20 minuta pomoću 5-inčnog (13 cm) reflektirajućeg teleskopa. Prva poznata fotografija zvijezda je bila fotografija zvijezde Vega koju su snimili astronom William Cranch Bond te fotograf i istraživača John Adams Whipple 16. i 17. srpnja 1850. iz opservatorija na Harvard Sveučilištu. Godine 1863. su engleski kemičar William Allen Miller i engleski astronom amater Sir William Huggins snimili prvu fotografiju spektrogram zvijezdi Sirius i Capella. Kasnije 1872. godine je američki liječnik Henry Draper, sin John William Drapera, snimio prvi spektrogram zvijezde Vega da bi istaknuo pojedine spektralne linije apsorpcije. Jedno od važnijih dostignuća u području astrofotografije se zbio 1883. godine kada je astronom amater Andrew Ainslie uspio snimiti nekoliko fotografija maglice Orion sa vremenom izloženosti od 60 minuta korištenjem 36-inčnog (91 cm) reflektirajućeg teleskopa, kojeg je sam izradio u dvorištu svoje kuće u Ealing-u izvan Londona. Na slikama su prvi puta bile prikazane udaljene zvijezde koje je nemoguće vidjeti golim okom.



Sl 1.1 Reflektirajući Teleskop astronoma Andrew Ainslie Common u Ealingu Engleska, snimljeno oko 1880. godine *(http://museumvictoria.com.au)*

Sl 1.2 Slika maglice Orion na kojoj su prvi puta prikazane zvijezde koje nisu vidljive golim okom, snimljeno oko 1883. godine *(https://en.wikipedia.org)*

Početkom 20. stoljeća počela je masovna konstrukcija velikih sofisticiranih reflektirajućih teleskopa posebno dizajniranih za astrofotografiju. Sredinom stoljeća, sagrađeni su divovski teleskopi poput 200-inčnog (5 metara) Hale teleskopa i 48-inčnog Samuel Oschin teleskopa na Palomar opservatoriju. Određeni napredak postignut je i u samoj tehnici snimanja, kao i u metodama razvijanja fotografija, no 1970. godine, nakon izuma CCD senzora počinje razdoblje digitalne fotografije. Digitalni CCD senzori imaju daleko veću osjetljivost na svjetlo koje ne opada tijekom duge izloženosti osvjetljenju na način kao što opada kod klasičnog analognog filma. Osim toga podržavaju snimanja u puno širem spektru i omogućuju jednostavnu pohranu i manipulaciju snimljenih podataka. Još jedna od prednosti digitalizacije je jednostavno podešavanje parametara pa tako teleskopi počinju koristiti različite konfiguracije CCD senzora. Pojavom digitalnih senzora počela je masovna digitalizacija u stručnim opservatorijima. Od kraja 20. stoljeća pa sve do danas teleskopi koriste CCD ili CMOS senzore za hvatanje slika. Usavršene su metode praćenja neba i sve veću ulogu poprima sama obrada prikupljenih digitalnih podataka sa senzora radi što boljeg praćenja nebeskih objekata i poboljšanja kvalitete slike.

1.2. Vrijeme ekspozicije

Budući da se kod astrofotografije većina slika snima noću dok su vidljivi nebeski objekti poput zvijezda i planeta, ključan faktor je količina svijetlosti koja dolazi do fotoosjetljivog senzora. Iz tog razloga se kod snimanja astronomskih fotografija koristi dugo vrijeme izloženosti senzora svjetlosnom zračenju nebeskih objekata što se još naziva vrijeme ekspozicije. Analogni film kao i korišteni digitalni senzori posjeduju moć sumiranja i akumuliranja fotona tijekom dugog vremenskog razdoblja izloženosti zračenju. Količina svjetla koja padne na film ili digitalni senzor se također povećava povećanjem promjera korištene optike (objektiva). Problemi koji se mogu javiti kod snimanja su posebno izraženi u urbanim područjima koja proizvode svjetlosno zagađenje. Iz tog razloga su oprema i specijalizirani opservatoriji najčešće smješteni na udaljenijim lokacijama kako bi se omogućilo korištenje dugih ekspozicija pa se time smanjuje utjecaj parazitskog svjetla prisutnog u urbanim područjima.

1.3. Kretanje nebeskih objekata i kompenzacija pomaka

Budući da Zemlja neprestano rotira oko svoje osi, teleskopi i oprema se moraju okretati u suprotnom smjeru kako bi se kompenziralo kretanje nebeskih objekata. Kompenzacija se može postići korištenjem ekvatorijalnih ili računalo upravljanih alt-azimutnih teleskopskih nosača, odnosno montaža. Zbog svojih nesavršenosti svi teleskopski montažni sustavi pate od pogrešaka u praćenju. Najčešći uzrok pogrešaka su nesavršeni motorni pogoni i mehaničke nesavršenosti zupčaničnog prijenosa nosača.

Pogreške se mogu ispraviti korištenjem drugog teleskopa montiranog uz postojeći koji je usmjeren upravo prema nekom svijetlom objektu na nebu koji se želi pratiti. Slike koje bi uhvatio drugi teleskop bi tada služile za određivanje pomaka u određenom smjeru i korekciju gibanja pratećeg objekta. Takvi dodatni teleskopi kojima je zadaća isključivo praćenje se nazivaju "vodiči". U povijesti se vođenje teleskopa radilo ručno. Tijekom izlaganja, promatrač bi ručno pomicao teleskop i time korigirao pomak objekta. Razvojem sve bržih moćnijih računalnih sustava, upravljanje i korekcija pomaka se postiže korištenjem automatiziranih sustava koji se danas mogu naći u profesionalnoj kao već i u amaterskoj opremi.

1.4. Korištene tehnologije

Tokom razvoja astrofotografije koristilo se mnogo različitih tehnologija snimanja i izrade samih fotografija. U 20. stoljeću se kao glavni senzor za hvatanje slika u astrofotografiji i fotografiji općenito koristio film. Vrijeme ekspozicije filma može varirati u rasponu od 10 minuta do više od sat vremena. Komercijalno dostupne filmske trake u boji podliježu smanjenju osjetljivosti tokom dužeg vremena ekspozicije. Tokom vremena izloženosti, osjetljivost na svjetlost određenih valnih duljina počinje različito opadati, što dovodi do pomaka boja u slici. Ovaj nedostatak se nadoknađuje tako da se snimanju slike samo na različitim valnim duljinama svijetlosti koje se onda kombiniraju u konačnu fotografiju sa korektnim odnosom boja. Budući da film puno sporije reagira od digitalnih senzora, sitne pogreške u praćenju se mogu lakše ispraviti bez puno primjetnog utjecaja na konačnu sliku. Korištenje filma u astrofotografiji i fotografiji općenito postalo je sve manje popularno pojavom jeftinih digitalnih senzora, čime se također znatno smanjuju i troškovi razvijanja samih fotografija.

1.4.1. CCD i CMOS optički senzori

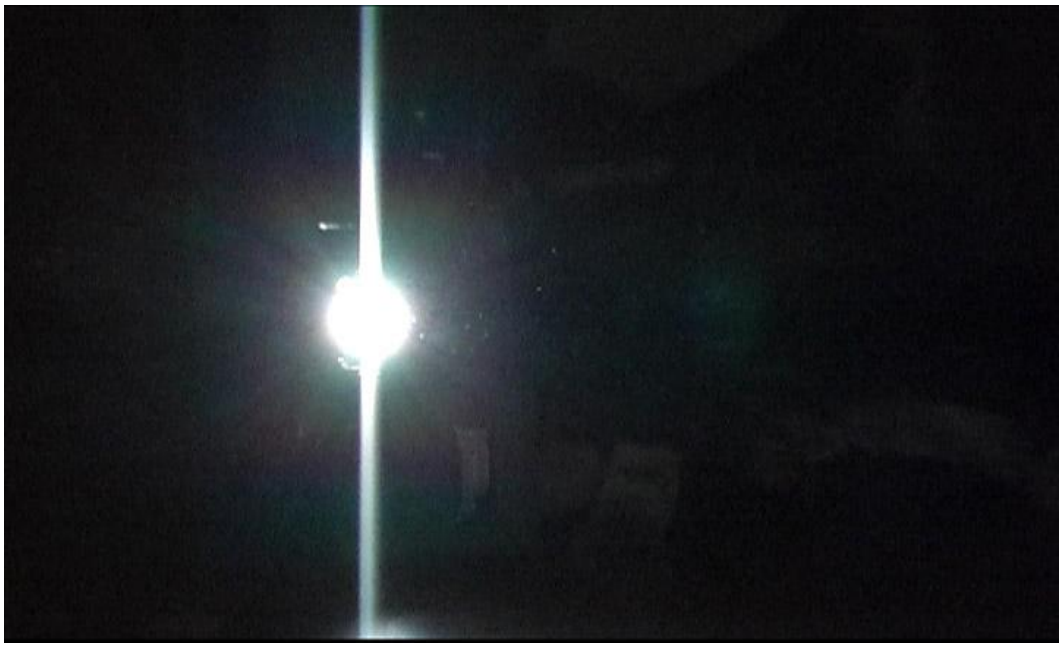
Danas su najčešće korišteni digitalni senzori u astrofotografiji i fotografiji općenito digitalni CCD i CMOS senzori. CCD i CMOS senzori su osjetljiviji od filmskih traka, čime se za kraće vrijeme ekspozicije postiže isti efekt, te imaju linearni odziv na svjetlost različitih valnih duljina. Moguće je snimiti više slika sa kraćim vremenom ekspozicije i njihovim kombiniranjem stvoriti dojam duge ekspozicije. Digitalni fotoaparati također nemaju puno pomičnih dijelova kao analogni, čime je smanjena pojava vibracija. Također je uređajima moguće upravljati daljinski.

Svaka ćelija CCD senzora je jedan analogni sklop. Kada svjetlost obasja senzor dolazi do nakupljanja naboja u svakoj ćeliji. Naboj se tada unutar senzora pomiče do sklopovlja gdje se pretvara u napon. Pretvorba se radi za svaki piksel posebno za vrijeme kad se vrijednosti čitaju sa čipa. Dodatno sklopovlje tada pretvara napon u digitalnu informaciju.

CMOS optički senzori spadaju u kategoriju aktivnih piksel senzora što znači da se sastoje od matrice piksel senzora gdje svaki piksel sadrži fotodetektor i aktivno pojačalo. Temelje se na CMOS poluvodičkoj tehnologiji. Slično kao kod CCD senzora, dodatno sklopovlje pored svakog piksel senzora pretvara svjetlosnu energiju na napon i naknadno u digitalni podatak.

Ni jedna od digitalnih tehnologija nema jasnu prednost u kvaliteti slike u odnosu na drugu, no postoje različiti nedostaci i karakteristike koje bi se kod određenih aplikacija trebale uzeti u obzir.

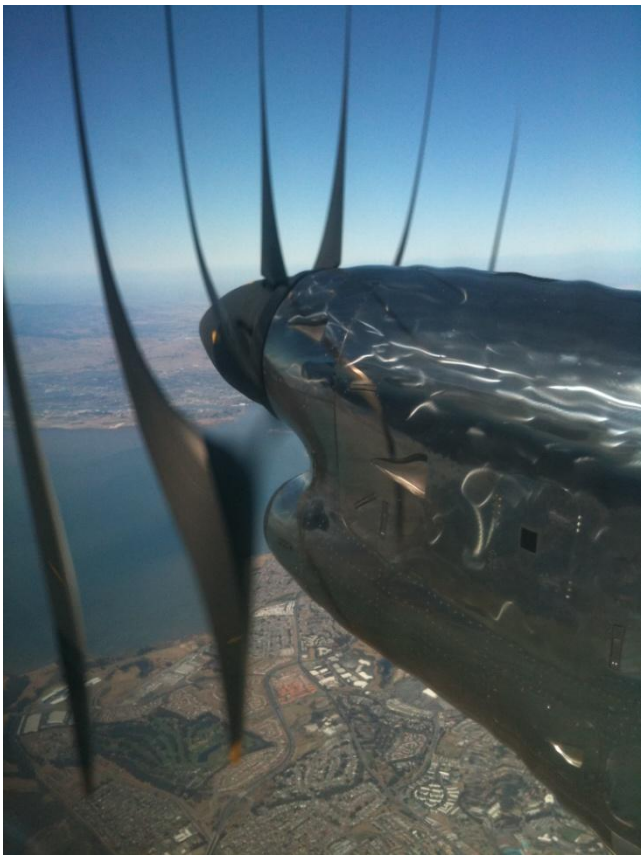
CCD senzori su više osjetljiv na vertikalno razmazivanje kod izloženosti jakim izvorima svjetlosti, kad je senzor preopterećen. Ova pojava se još naziva *vertical smear*. Do razmazivanja dolazi tijekom pomicanja naboja iz pojedinih ćelija tokom procesa čitanja slike sa senzora. Ako je senzor za vrijeme pomicanja naboja pojedinih ćelija i dalje izložen jakom osvjetljenju, naboj se i dalje nakuplja i dolazi do tuneliranja nositelja naboja u dio senzora namijenjen za pomicanje naboja. Vertikalna linija na slici iznad izvora svijetlosti je generirana pomicanjem prethodne slike, a linija ispod je generirana pomicanjem trenutne slike. Razmazivanje se može u potpunosti ukloniti tako da se senzor izloži osvjetljenju samo za vrijeme ekspozicije, a između hvatanja slike da bude zaklonjen od izvora svijetlosti. Drugi način uklanjanja ovog nedostatka je brže pomicanje naboja i čitanje slike sa senzora, no to rezultira većim šumom na kompletnoj slici.



Sl 1.3 Primjer vertikalnog razmazivanja (*vertical smear*) kod CCD optičkih senzora (*https://en.wikipedia.org*)

CMOS senzori ne pate od ovog problema. S druge strane, CMOS senzori su osjetljiviji na neželjene učinke koji dolaze kao rezultat tzv. *rolling shutter* efekta. *Rolling shutter* je metoda hvatanja slike gdje se kompletna slika ne uzima u jednom vremenskom trenutku, već

skeniranjem kadra veritkalno po stupcima ili horizontalno po recima. Ovom metodom nisu svi dijelovi slike uhvaćeni u istom vremenskom trenutku iako se dobivena slika prikazuje i interpretira kao takva. Konkretno ova metoda predstavlja problem kod brzo gibajućih objekata ili bljeskova koji generiraju distorziju na slici. Nedostatak se može ukloniti jedino bržim skeniranjem, no time se ujedno smanjuje i vrijeme ekspozicije.



Sl 1.4 Primjer *rolling shutter* efekta kod CMOS optičkih senzora *(https://en.wikipedia.org)*

Općenito CMOS senzori se mogu implementirati korištenjem manjeg broja potrebnih komponenti, troše manje energije i digitalni podaci budu brže spremni za čitanje za razliku od CCD senzora. Međutim CCD senzori postižu bolje rezultate kad je u pitanju snimanje fotografija u uvjetima smanjene vidljivosti, tj. manjka svijetlosti, što je konkretno slučaj kod astrofotografije.

1.5. Post-processing

Kod astrofotografije kao i kod fotografije općenito važnu uloga igra naknadna obrada slike. Bilo da je u pitanju digitalna ili analogna fotografija, obrada najčešće podrazumijeva

mijenjanje svjetline (*brightness*) i manipulacija boja u svrhu povećanja ili smanjenja kontrasta. Za razliku od klasične fotografije, astrofotografija je podložnija šumu kao i raznim drugim neželjenim efektima što znatno otežava poboljšavanje kvalitete slike. Često jedna fotografija nije dovoljna da bi se postigao zadovoljavajući učinak. Naprednije metode obrade slike uključuju više uhvaćenih fotografija koje se zajedno kombiniraju i obrađuju sa ciljem izoštravanja slike. Kod tih metoda nije neobično da se kombinira i do tisuću fotografija. Kombiniranjem i usporedbom više slika moguće je korigirati pogreške nastale tokom praćenja nebeskih objekata, istaknuti slabije, oku nevidljive, objekte sa lošim odnosom signal-šum, filtrirati svjetlosno zagađenje na slikama koje je često prisutno u urbanim područjima i ublažiti utjecaj atmosferskih smetnji.

Atmosferske smetnje predstavljaju velik problem jer stvaraju distorziju na slici, odnosno izobličuju je. Još veći problem predstavlja činjenica da se distorzija mijenja kroz vrijeme ovisno o trenutnom stanju atmosfere. Početkom 90-ih godina 20. stoljeća razvijene su razne adaptivne metode za suzbijanje utjecaja atmosfere, no ni jedna ne uklanja distorziju u potpunosti. Česta, iako primitivna, metoda koja se još u tu svrhu koristi je tzv. *lucky imaging* metoda kod koje se slike hvataju sa manjim vremenom ekspozicije tako da su atmosferske smetnje tokom ekspozicije minimalne. Nakon više uhvaćenih slika odabiru se one kod kojih je utjecaj najmanji i zajedno se kombiniraju radi postizanja bolje kvalitete. Neželjeni efekt se još može ublažiti promatranjem uže regije neba.

Digitalne slike također trebaju daljnju obradu radi uklanjanja šuma nastalog zbog dugog vremena ekspozicije. Tehnika koje se u tu svrhu koristi je tzv. oduzimanje "tamnog okvira" (*dark-frame subtraction*). *Dark-frame subtraction* se oslanja na činjenicu da je dio nastalog šuma jednak i ponavlja se na svakoj slici, što se još naziva *fixed pattern noise*. Ponovljivost proizlazi iz malih razlika u odzivu i osjetljivosti individualnih senzora u matrici piksela, odnosno ćelija, na svijetlost. To može biti posljedica razlika u veličini piksela, korištenom materijalu ili smetnjama uzrokovanim ostalim sklopovljem. Ovaj oblik šuma je izraženiji kod dužih vremena ekspozicije i promjena temperature. Budući da se ova vrsta šuma uvijek pojavljuje na istom mjestu, tehnika uključuje hvatanje jedne ili više slika sa zaklonjenim optičkim senzorom (tzv. tamni okvir). Slike se tada kombiniraju u jednu i odbijaju od originalne slike.

Još jedna metoda suzbijanja šuma i općenito poboljšavanja kvalitete slike je tzv. *Shift-and-add* metoda (još nazivana *image-stacking*). Kod metode se uzima niz slika pratećeg

objekta koje su uhvaćene sa kratkim vremenom ekspozicije. Budući da se nebeski objekt pomiče tokom vremena, slike su relativno pomaknute jedna u odnosu na drugu. Metoda uključuje određivanje iznosa i smjera pomaka pojedinih slika, oslanjajući se na činjenicu da su kod astrofotografije na slikama u većini slučajeva zvijezde koje se tada koriste kao referenca u određivanju pomaka. Nakon toga se fotografije usklađuju, odnosno poravnavaju, te se pojedini pikseli na slikama usrednjuju. Usrednjavanjem piksela, ukupni odnos signal-šum na konačnoj slici se povećava za faktor jednak drugom korijenu broja slika.

2. Postupak obrade slike

Pod pretpostavkom da sliku uglavnom čine tamni pikseli koji nisu od značaja, zvijezde na slici čine nakupine svijetlih piksela. Tamni pikseli se mogu smatrati šumom, dok nakupine svijetlih piksela sadržavaju informaciju o zvijezdi i njenom kretanju. Takve svijetle mrlje na slici se mogu aproksimirati *gauss*-olikim zvonom, odnosno funkcijom. Također nebo se snima noću, tako da općenito jako malo svijetlosti dolazi do senzora na teleskopu. Da bi uopće bilo moguće vidjeti objekte na nebu koji emitiraju slabu svijetlost kao što su zvijezde, nužno je da senzor određeno vrijeme bude izložen njihovom zračenju. To se postiže postavljanjem duže ekspozicije senzora odnosno kamere. U slučaju astrofotografije, vrijeme ekspozicije može iznositi do nekoliko desetaka minuta. Za to vrijeme dok je senzor aktivan zvijezde ne miruju već se kreću brzinom određenom zemljinom rotacijom. Tako snimljene zvijezde na slici postaju izdužene i gube svoj kružni oblik te više nalikuju elipsama. Kod dužih ekspozicija može čak doći i do deformacije elipse budući da nebeski svod rotira oko zemljine osi rotacije što je prikazano na slici Sl 2.1. U radu se pretpostavlja da deformacija ne dolazi do izražaja zbog kraćeg vremena ekspozicije pa se stoga taj efekte zanemaruje.

Sl 2.1 Primjer deformacije zvijezdi zbog rotiranja nebeskog svoda (*www.dpreview.com*)

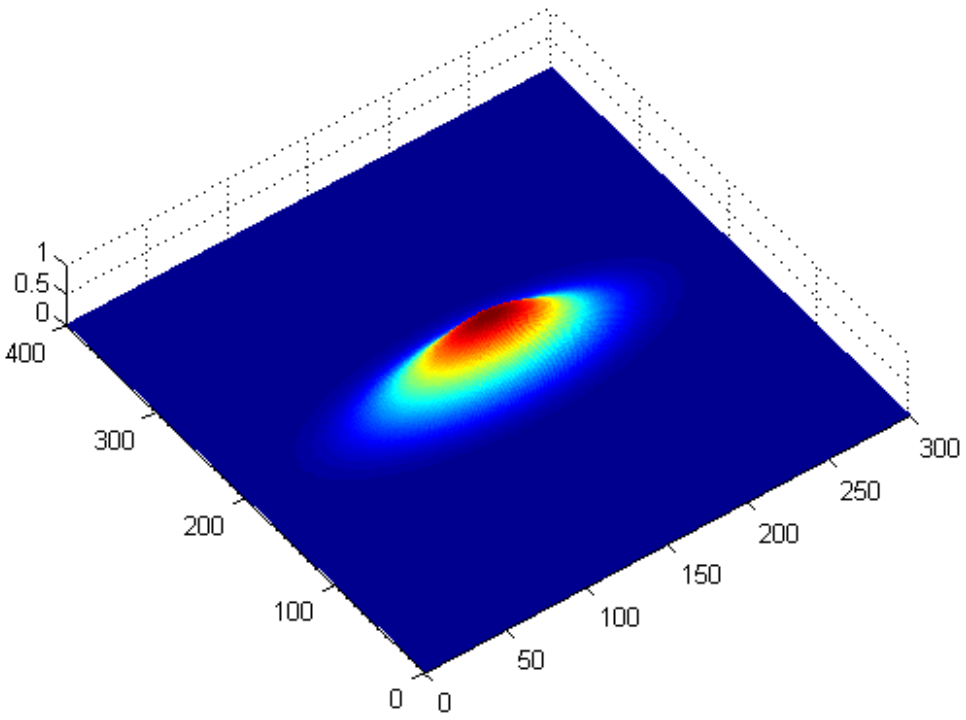
Za vrijeme dok je senzor aktivan može pokupiti različite smetnje od neželjenih izvora svijetlosti što se očituje kao šum na slici. U slučaju kad je razina šuma značajna, dijelovi pozadine se čak mogu krivo interpretirati kao objekti na slici. Šum je posebno izražen kod snimanja noćnih fotografija u urbanim područjima gdje različita svijetla na ulici i ulična rasvjeta mogu utjecati na senzor. Ako šum nije toliko izražen može se ukloniti filtriranjem, no time se gubi dio informacije o stvarnim zvijezdama.

Svi spomenuti negativni efekti imaju za posljedicu da su zvijezde na slici zašumljeni i izduženi eliptički objekti koji se mogu aproksimirati *gauss*-olikom funkcijom. Oblik funkcije je sljedeći:

$$f(x,y) = Aexp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)\right) \tag{1}$$

Parametri kojima se funkcija opisuje su:

$$x_0, y_0 - \ koordinate\ težišta\ funkcije$$
$$A - vrijednost\ funkcije\ f(x_0, y_0)$$
$$\sigma_x, \sigma_y - \ standardne\ devijacije\ po\ x\ i\ y\ osi$$



Sl 2.2 *Gaussian*, $\boldsymbol{x_0 = 150}$, $\boldsymbol{y_0 = 200}$, $\boldsymbol{A = 0.9}$, $\boldsymbol{\sigma_x = 40}$, $\boldsymbol{\sigma_y = 20}$

U slučaju da je funkcija zarotirana za neki kut θ, dolazi se do općenitog izraza:

$$f_{\varphi}(x,y) = Aexp\left(-(a(x-x_0)^2 + 2b(x-x_0)(y-y_0) + c(y-y_0)^2)\right) \tag{2}$$

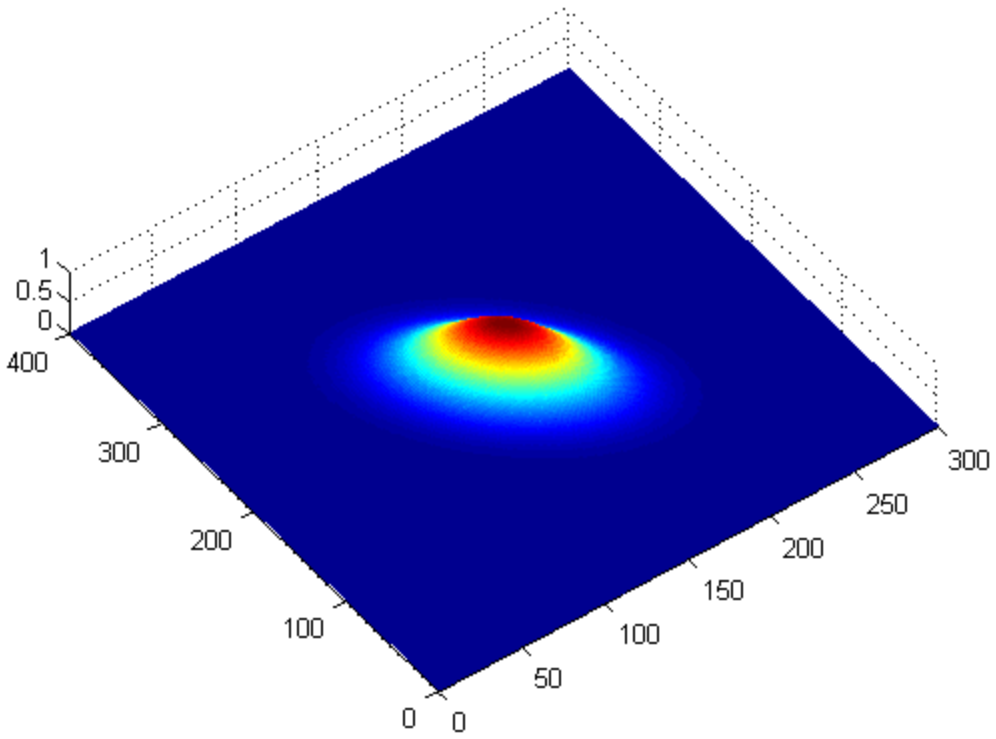
Gdje su:

$$a = \frac{\cos^2\varphi}{2\sigma_x^2} + \frac{\sin^2\varphi}{2\sigma_y^2} \tag{3}$$

$$b = -\frac{\sin 2\varphi}{4\sigma_x^2} + \frac{\sin 2\varphi}{4\sigma_y^2} \tag{4}$$

$$c = \frac{\sin^2\varphi}{2\sigma_x^2} + \frac{\cos^2\varphi}{2\sigma_y^2} \tag{5}$$

$$\varphi - kut\ rotacije$$



Sl 2.3 *Gaussian*, $\boldsymbol{x_0 = 150}$, $\boldsymbol{y_0 = 200}$, $\boldsymbol{A = 0.9}$, $\boldsymbol{\sigma_x = 40}$, $\boldsymbol{\sigma_y = 20}$, $\boldsymbol{\varphi = 60°}$

Gauss-olike funkcije koje se očekuju na slici su zarotirane za neki kut koji predstavlja smjer kretanja zvijezda na nebu.

2.1. Estimacija parametara *gussian*-a

Pod pretpostavkom da je funkcija idealna i bez aditivnog šuma moguće je odrediti parametre *gauss*-a. Računajući prvi moment funkcije određuje se težište *gaussian*-a te samim time i lokacija zvijezde na slici. Izrazi za izračunavanje koordinata pojedine zvijezde na slici su slijedeći:

$$x_0 = \frac{\sum_x \sum_y x f_\varphi(x,y)}{\sum_x \sum_y f_\varphi(x,y)} \tag{6}$$

$$y_0 = \frac{\sum_x \sum_y y f_\varphi(x,y)}{\sum_x \sum_y f_\varphi(x,y)} \tag{7}$$

Gdje su x i y koordinate, a $f_\varphi(x,y)$ je zarotirani *gaussian*, odnosno vrijednost piksela na slici za dane koordinate. Bitno je da suma ne obuhvaća ostale zvijezde, odnosno da se odnosi samo na izolirani dio slike sa zvijezdom. Općenito težište ne ovisi o kutu rotacije *gaussian*-a φ, uvijek je konstantno.

Računajući drugi moment dobije se informacija o obliku funkcije nad kojom se računa. Drugi moment ne zarotiranog ($\varphi = 0°$) *gaussian*-a $f(x,y)$ jednak je:

$$m_{20} = \frac{\sum_x \sum_y x^2 f(x,y)}{\sum_x \sum_y f(x,y)} = \frac{\sum_x \sum_y \left(x^2 Aexp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2} \right) \right) \right)}{\sum_x \sum_y \left(Aexp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2} \right) \right) \right)} \tag{8}$$

$$m_{02} = \frac{\sum_x \sum_y y^2 f(x,y)}{\sum_x \sum_y f(x,y)} = \frac{\sum_x \sum_y \left(y^2 Aexp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2} \right) \right) \right)}{\sum_x \sum_y \left(Aexp\left(-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2} \right) \right) \right)} \tag{9}$$

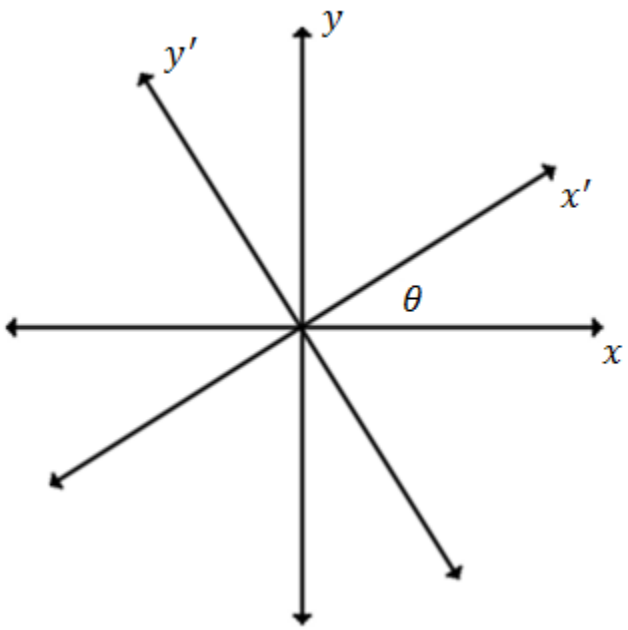
$$m_{20} = \sigma_x^2 \tag{10}$$

$$m_{02} = \sigma_y^2 \tag{11}$$

Budući da drugi moment ovisi o obliku funkcije nad kojom se računa, promjenom kuta rotacije *gaussian*-a φ mijenja se i iznos drugog momenta. Može se dokazati da je drugi moment elipse ili *gaussian*-a maksimalan u smjeru veće poluosi ili σ_x, odnosno minimalan u smjeru manje poluosi ili σ_y. To svojstvo se može iskoristiti kod računanja varijanci zarotiranog *gaussian*-a. Koordinate zarotiranog koordinatnog sustava za kut θ, kao što je prikazano na slici Sl 2.4, su:

$$x' = x\cos\theta + y\sin\theta \tag{12}$$

$$y' = -x\sin\theta + y\cos\theta \tag{13}$$



Sl 2.4 Odnos Kartezijevog koordinatnog sustava i sustava zakrenutog za kut $\boldsymbol{\theta}$

Slijedi izraz za drugi moment $m_{x'}$ zarotiranog *gaussian*-a $f_\varphi(x,y)$ duž zakrenute osi x':

$$m_{x'} = \frac{\sum_x \sum_y x'^2 f_\varphi(x,y)}{\sum_x \sum_y f_\varphi(x,y)} \tag{14}$$

$$m_{x'} = \frac{\sum_x \sum_y x^2 f_\varphi(x,y)\cos^2\theta + \sum_x \sum_y y^2 f_\varphi(x,y)\sin^2\theta + \sum_x \sum_y xy f_\varphi(x,y)\sin 2\theta}{\sum_x \sum_y f_\theta(x,y)} \tag{15}$$

$$m_{x'} = m_{20}\cos^2\theta + m_{02}\sin^2\theta + m_{11}\sin 2\theta \tag{16}$$

U danom izrazu (16) m_{20} je drugi moment zarotiranog *gaussian*-a duž osi x, m_{02} je drugi moment duž osi y, a m_{11} je prvi moment duž osi x i y.

Iz izraza (16) je vidljivo da se općenito drugi moment neke funkcije u smjeru zarotirane osi x' može odrediti poznavajući momente duž x i y.

Slično se može izvesti i izraz za drugi moment $m_{y'}$ zarotiranog *gaussian*-a $f_\varphi(x,y)$ duž zarotirane osi y':

$$m_{y'} = \frac{\sum_x \sum_y y'^2 f_\varphi(x,y)}{\sum_x \sum_y f_\varphi(x,y)} \tag{17}$$

$$m_{y'} = \frac{\sum_x \sum_y x^2 f_\varphi(x,y)\sin^2\theta + \sum_x \sum_y y^2 f_\varphi(x,y)\cos^2\theta - \sum_x \sum_y xy f_\varphi(x,y)\sin 2\theta}{\sum_x \sum_y f_\varphi(x,y)} \tag{18}$$

$$m_{y'} = m_{20}\sin^2\theta + m_{02}\cos^2\theta - m_{11}\sin 2\theta \tag{19}$$

Budući da kod *gaussian*-a drugi moment ima ekstrem u smjeru poluosi σ_x i σ_y, lako se može odrediti kut φ za koji je funkcija zarotirana. Dakle, potrebno je naći ekstreme funkcija $m_{x'}$ i $m_{y'}$.

$$\frac{d(m_{x'})}{d\theta} = -\sin 2\theta\, m_{20} + \sin 2\theta\, m_{02} + 2\cos 2\theta\, m_{11} \tag{20}$$

$$\frac{d(m_{y'})}{d\theta} = \sin 2\theta\, m_{20} - \sin 2\theta\, m_{02} - 2\cos 2\theta\, m_{11} \tag{21}$$

$$\frac{d(m_{x'})}{d\theta} = 0 \tag{22}$$

$$\frac{d(m_{y'})}{d\theta} = 0 \tag{23}$$

Oba izraza (22) i (23) daju jednako rješenje:

$$\varphi = \frac{1}{2}\tan^{-1}\left(\frac{2m_{11}}{m_{20} - m_{02}}\right) \tag{24}$$

Kod računanja parametara *gauss*-a na slici, nije poznat kut za koji je funkcija zarotirana, no iz izraza (24) se on može izračunati. Ekstremi funkcija $m_{x'}$ i $m_{y'}$ su upravo σ_x^2 i σ_y^2.

$$m_{x'}(\varphi) = \sigma_x^2 \tag{25}$$

$$m_{y'}(\varphi) = \sigma_y^2 \tag{26}$$

Izrazi (16) i (19) se mogu prikazati i u matričnom obliku:

$$R * D * R^T = C \tag{27}$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{28}$$

$$D = \begin{bmatrix} m_{x'} & 0 \\ 0 & m_{y'} \end{bmatrix} \tag{29}$$

$$C = \begin{bmatrix} m_{20} & m_{11} \\ m_{11} & m_{02} \end{bmatrix} \tag{30}$$

R je matrica rotacije, D je dijagonalna matrica vlastitih vrijednosti, a C je matrica kovarijance originalnog koreliranog modela. Za $\theta = \varphi$ dobije se:

$$D = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{31}$$

Kako bi se ubrzao proces izračunavanja σ_x^2 i σ_y^2, kut φ se ne uvrštava direktno u izraze (16) i (19), već se kombiniranjem i sređivanjem izraza (16), (19), (25) i (26) dobije jednostavniji oblik za sumu i razliku:

$$\sigma_x^2 + \sigma_y^2 = m_{20} + m_{02} \tag{32}$$

$$\sigma_x^2 - \sigma_y^2 = \sqrt{(m_{20} - m_{02})^2 + 4m_{11}^2} \tag{33}$$

Posljednji parametar koji je potrebno izračunati je A, vrijednost funkcije $f_\varphi(x,y)$ u težištu x_0 i y_0. A se dobije poznavanjem volumena V ispod *gaussian*-a.

$$V = \sum_x \sum_y f_\varphi(x,y) = 2A\pi\sigma_x\sigma_y \tag{34}$$

$$A = \frac{V}{2\pi\sigma_x\sigma_y} \tag{35}$$

Time su svi izrazi potrebni za izračun parametara *gauss*-a izvedeni.

2.2. Izdvajanje pojedinih zvijezdi na slici

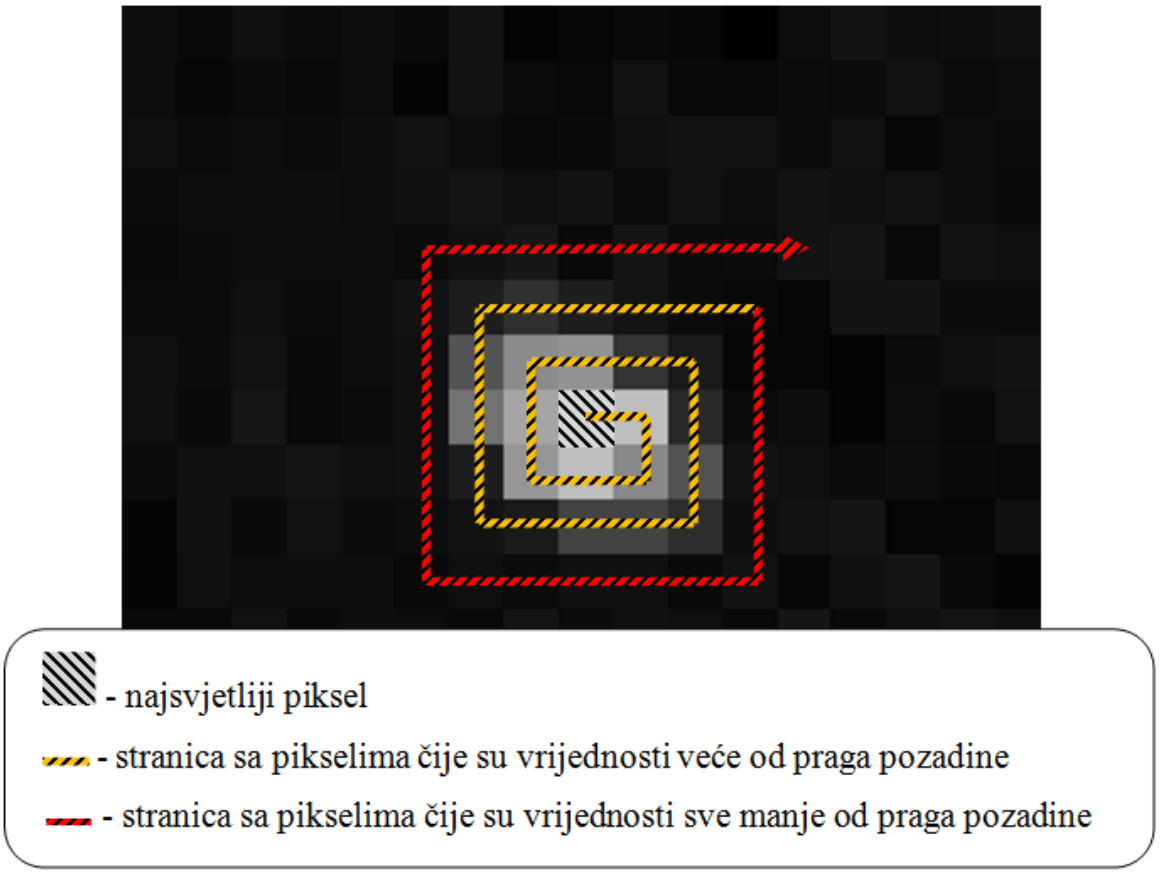
Na astronomskim slikama se uglavnom očekuje više zvijezda. Jedan od problema koji se javlja prije nego se krene u određivanje parametara zvijezdi je određivanje regija pojedinih zvijezdi. Svaka zvijezda mora biti izolirana i imati svoju regiju nad kojom se računaju njeni parametri. Ako bi bilo više zvijezdi u jednoj regiji tada bi se više *gauss*-olikih objekata pokušalo aproksimirati jednim, što daje za rezultat parametre nepostojeće zvijezde.

Još jedan problem se javlja kod prisutnosti šuma na slici. Naime, pozadina na slici je u idealnom slučaju bez šuma crna, odnosno vrijednosti piksela pozadine su 0. Kod prisutnosti šuma potrebno je odrediti prag ispod kojeg se piksel slike smatra pozadinom, a iznad kojeg se interpretira kao dio zvijezde. Budući da je veći dio slike pozadina, odnosno pozadinski šum, uzeto je da se prag određuje iz postotka najsvjetlijih piksela. Npr. može se smatrati da 2% najsvjetlijih piksela pripada zvijezdama, a ostalo je pozadina. Iz postotka se tada računa vrijednost praga. Pikseli zvijezda se tada prepoznaju na osnovu izračunatog praga.

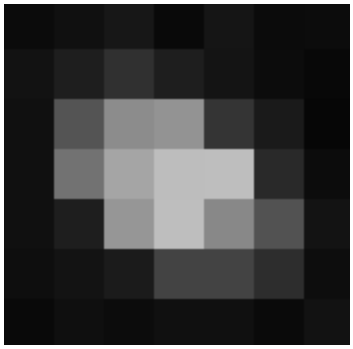
Budući da se ne zna koliko zvijezdi ima i gdje se nalaze na slici, potrebno ih je naći i odrediti regiju svake zvijezde. Jednostavna metoda koja se može koristiti je traženje najsvjetlijeg piksela na slici. Ako je vrijednost piksela veća od praga pozadine, piksel se može smatrati dijelom neke zvijezde. Pošto je pretpostavljeno da je zvijezda *gauss*-olikog oblika, nađeni najsvjetliji piksel bi se trebao nalaziti blizu težišta *gaussian*-a (Sl 2.5), dakle nađen je piksel koji se nalazi oko samog centra zvijezde. Ta činjenica se može iskoristiti kod nalaženja regije zvijezde. Pretpostavka je da se nađeni piksel također nalazi i u središtu regije zvijezde.

Korišteni algoritam koji traži regiju zvijezde kreće upravo od nađenog najsvjetlijeg piksela te se on koristi kao referenca. Algoritam spiralno u smjeru kazaljke na satu ispituje

sve susjedne piksele i time spiralno širi regiju. Ako je vrijednost piksela veća od praga pozadine on se interpretira kao zvijezda. Algoritam traži tako dugo dok se ne nađu 4 uzastopne stranice pravokutnika čiji se svi pikseli interpretiraju kao pozadina, tj vrijednosti svih piksela su ispod praga pozadine. Time se smatra da su pronađeni rubovi regije, odnosno zvijezde.



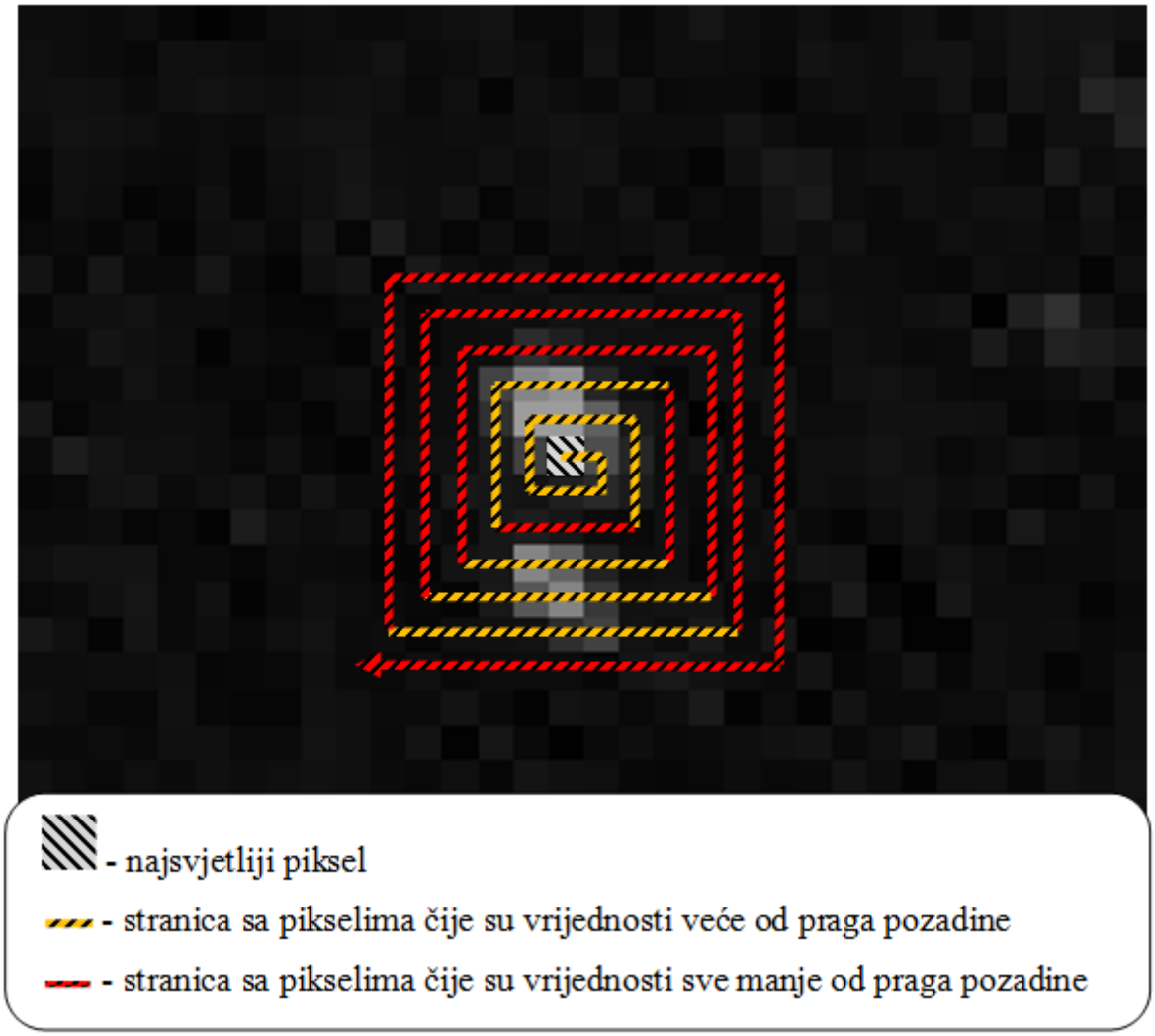
Sl 2.5 Postupak određivanja regije pojedine zvijezde



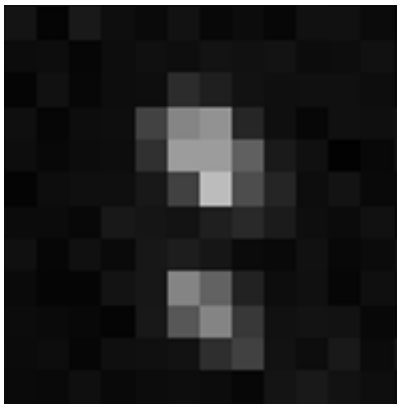
Sl 2.6 Izdvojena regija zvijezde nad kojom se računaju parametri *gaussian*-a

Rezultat algoritma je izdvojena regija zvijezde koja je kvadratnog oblika sa referentnim (najsvjetlijim) pikselom u centru. Ako se nakupina piksela nalazi na rubu same slike ili ako algoritam spiralno proširi regiju preko ruba slike, pretraživanje se prekida i zvijezda se zanemaruje. Tako nađena zvijezda se zanemaruje jer njena aproksimacija *gaussian*-om ne bi bila točna, budući da očito dio zvijezde koji se nalazi preko ruba slike nedostaje pa se i zanemaruje kod proračuna parametara.

Prednosti ove metode pretraživanja su da je jednostavna i može se koristiti u slučaju kad se ne zna koji dio neba se promatra, odnosno kad se ne zna pozicija i broj zvijezdi na slici. Nedostaci su da se pretpostavlja da se najsvjetliji piksel zvijezde nalazi upravo u njenom centru, što je uglavnom točno. Ta pretpostavka stvara jedino problem kad je su dvije zvijezde vrlo blizu jedna drugoj i ako se najsvjetliji piksel ne nalazi u središtu prve zvijezde već na njenom rubu. U tom slučaju postoji mogućnost da se kod pretraživanja okoline referentnog piksela regija proširi i na susjednu zvijezdu i time bude u regiju uključena i susjedna zvijezda. U većini slučajeva pretpostavka važi i najsvjetliji piksel se doista nalazi oko težišta zvijezde, pa se ovaj nedostatak može zanemariti. Ako doista dođe do toga da u regiju upadne i susjedna zvijezda, najvjerojatnije će nakon izračuna njen kut zakreta *gaussian*-a φ, kao i omjer σ_x / σ_y značajno odstupati od većine zvijezda pa se lako eliminira tako nađena zvijezda. Nedostatak se može ukloniti mijenjanjem praga pozadine.



Sl 2.7 Određivanje regije zvijezde sa referentnim pikselom na rubu, u blizini susjedne zvijezde



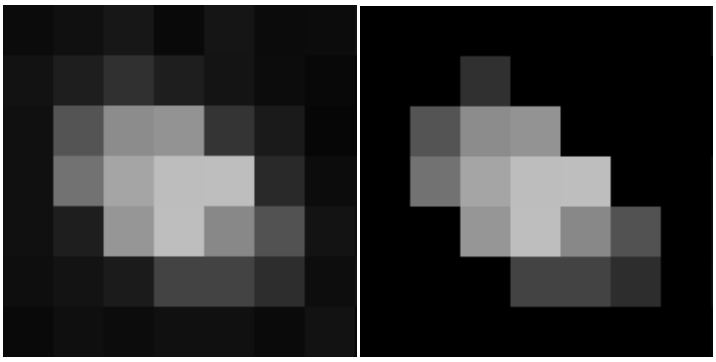
Sl 2.8 Izdvojena regija koja greškom obuhvaća dvije zvijezde

2.3. Točnost aproksimacije zvijezde *gaussian*-om

Budući da na slici postoji šum, parametri *gauss*-a se samo aproksimiraju. Da bi se ublažio utjecaj šuma slika bi se mogla filtrirati prije računanja parametara. Filtriranjem se osim šuma također utječe i na kvalitetu slike pa tako i na zvijezde na slici. Teško je u potpunosti ukloniti šum, a da dio slike bez šum ostane nepromijenjen. U većini slučajeva se filtriranjem uklanja samo jedan dio šuma.

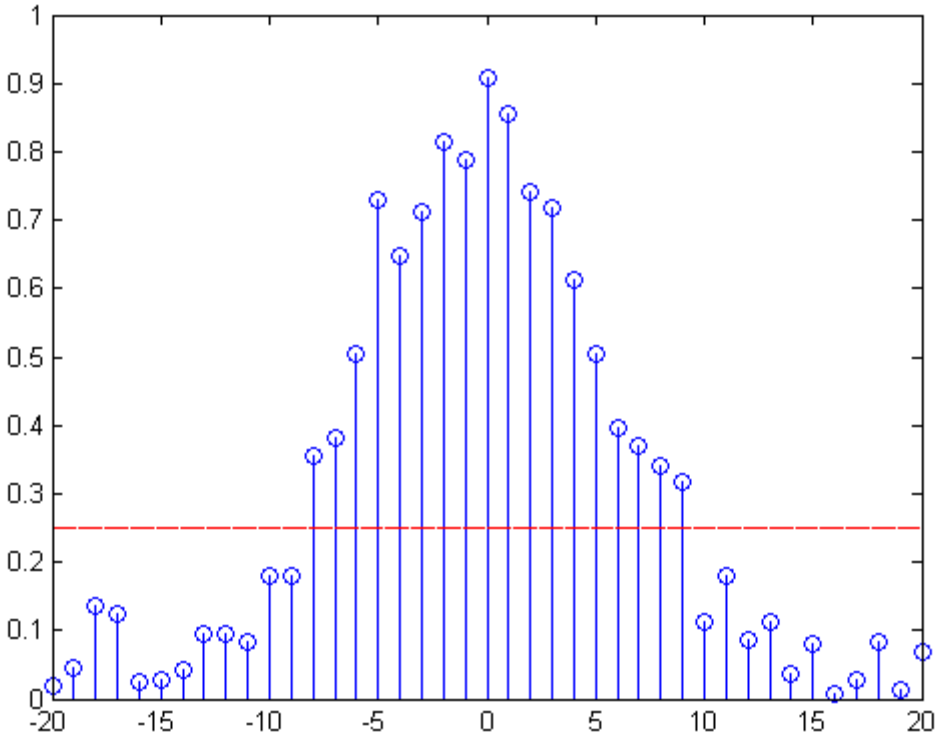
Drugi način na koji se može ublažiti utjecaj šuma je određivanjem praga pozadine. Ako je vrijednost piksela ispod praga pozadine, smatra se da je bez značaja,tj pozadina. Ako je pak iznad praga pozadine interpretira se kao dio zvijezde. Kod izračuna parametara *gauss*-a u obzir se uzimaju samo pikseli koji pripadaju zvijezdama, znači samo pikseli čije vrijednosti su iznad praga pozadine. Ostali pikseli čije vrijednosti su ispod praga pozadine se zanemaruju, bez obzira nalazili se oni u regiji zvijezde.

Da bi se utjecaj pozadine eliminirao potrebno je ukloniti sve pripadne piksele, tj. njihovu vrijednost postaviti u 0 (crna boja).

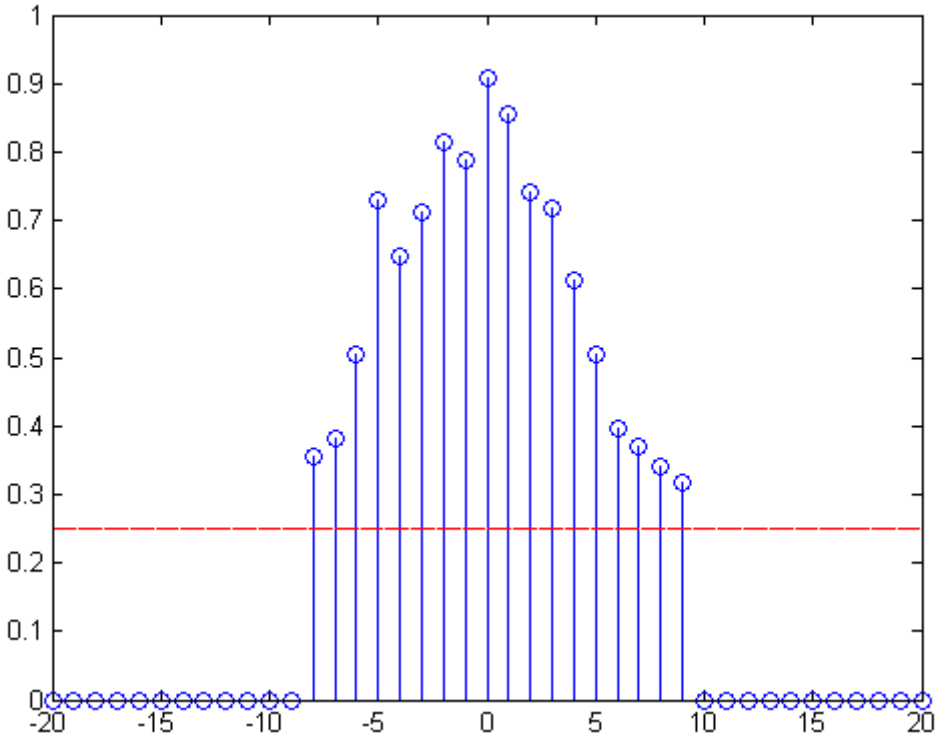


Sl 2.9 Izdvojena regija zvijezde prije i nakon uklanjanja piksela pozadine

Time što je uklonjena pozadina, uklonjen je i taj dio *gaussian*-a pa ne ulazi u izračun parametara. Parametri koji se na osnovu ostalih uzoraka tako pokušavaju izračunati u nekim slučajevima mogu značajno odstupati od stvarnih. Pogrešci još dodatno doprinosi šum koji nije uklonjen na pikselima koji nisu dio pozadine, tj. koji se interpretiraju kao dio zvijezde. Što je prag pozadine veći, veći dio regije zvijezde će biti nuliran i neće ulaziti u proračun. Očito je da prag pozadine znatno utječe na konačne parametre i nužno je da se dobro procijeni kako bi se utjecaj šuma uklonio do te mjere da znatno ne narušava točnost izračunatih parametara.



Sl 2.10 Jednodimenzionalni *gauss* sa aditivnim šumom i označenim pragom pozadine



Sl 2.11 Jednodimenzionalni *gauss* sa aditivnim šumom i uklonjenim vrijednostima koje su ispod praga pozadine

2.4. Određivanje iznosa i smjera pomaka

Da bi sustav mogao pratiti uvijek isti dio neba, potrebno je osigurati da se postolje sa kamerom zakreće u smjeru gibanja zvijezda na nebu. Iz tog razloga potrebno je odrediti pomak trenutne slike, odnosno zvijezda, u odnosu na prethodnu sliku. Iz iznosa i smjera pomaka se tada određuje kut zakreta kamere. Ako je pomak dobro aproksimiran, kamera će u sljedećoj iteraciji pratiti isti dio neba i postupak se ponavlja. Pomak se može odrediti na temelju kompletne slike ili na osnovu pomaka pojedinih zvijezdi. U daljnjem tekstu opisan je drugi način, tj. na osnovu pomaka pojedinih zvijezdi.

Većina zvijezda na trenutnoj slici ima ekvivalent na prethodnoj. Slučaj kada to ne važi su zvijezde koje su na trenutnoj slici tek ušle u kadar i nisu bile u kadru na prethodnoj. Još jedan slučaj su krivo prepoznate zvijezde, tj. nakupine piksela koje nisu posljedica zračenja zvijezde već šuma, a koje nisu bile uklonjene variranjem praga pozadine. Takve nakupine se interpretiraju kao zvijezde, no lako ih je prepoznati i ukloniti, budući da izračunati parametri *gauss*-a za takvu nakupinu znatno odstupaju od srednjih parametara svih nađenih zvijezdi. Kad se pomak određuje na temelju pomaka svake nađene zvijezde, tada je prvo potrebno pronaći ekvivalentnu zvijezdu na prethodnoj slici. Taj zadatak nije trivijalan, budući da se za velike pomake može desiti da ni jedna zvijezda na trenutnoj slici ne odgovara prethodnoj, tj. da se u trenutnoj iteraciji promatra sasvim drugi dio neba. Ovakvi ekstremni slučajevi se ne mogu jednostavno detektirati i korigirati bez da postoji nekakva statistika prema kojoj se pomak može u grubo pretpostaviti. Ako se promatrani dio neba na trenutnoj slici preklapa sa dijelom iz prethodne slike tada je moguće naći ekvivalentne zvijezde, no ni u tom slučaju se ne može sa sigurnošću tvrditi da se radi o ekvivalentnoj zvijezdi. Što je pomak manji u odnosu na veličinu same slike, time je veća vjerojatnost da je na prethodnoj slici pronađen točan ekvivalent pojedine zvijezde.

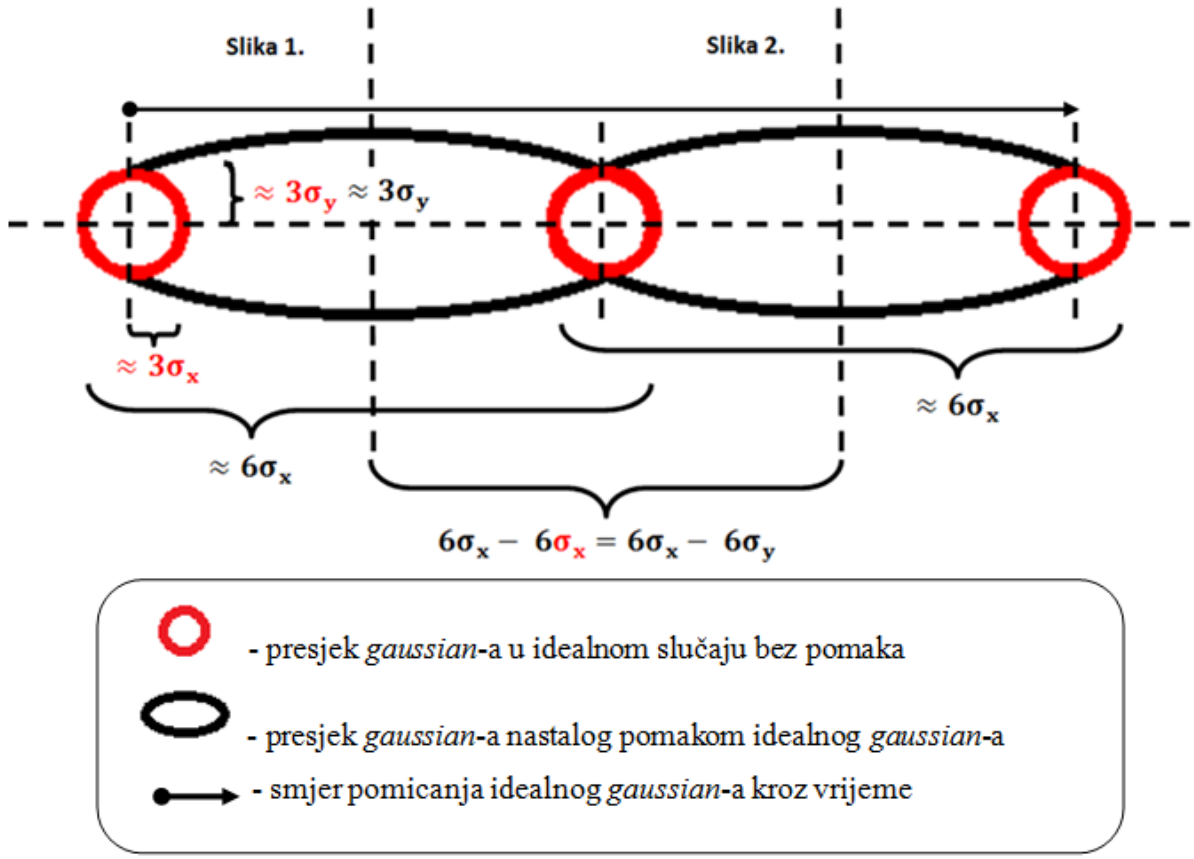
Ako ne postoji statistika prema kojoj bi se pomak mogao pretpostaviti, postoji jednostavna metoda prema kojoj se može aproksimirati iz izračunatih parametara *gauss*-a za svaku zvijezdu. Metoda može biti izuzetno korisna kod prvih nekoliko iteracija, odnosno slika ili u svrhu uhodavanja sustava. Ako ne bi bilo pomaka, zvijezde bi u idealnom slučaju bili *gaussian*-i sa jednakim σ_x i σ_y, tj. presjek *gaussian*-a bi bila kružnica. Gibanjem zvijezda prvobitni "kružni" *gaussian*-i svojim micanjem ostavljaju trag koji bude zabilježen na slici. Kad pomak postoji, σ_x i σ_y se razlikuju i presjek *gaussian*-a je elipsa. Budući da se zna da je pomakom generirani "eliptički" *gaussian* nastao jednostavnim pomicanjem idealnog

"kružnog" *gaussian*-a koji je ostavio trag u smjeru σ_x, očito je da tada oba *gaussian*-a imaju približno jednaki σ_y koji je neovisan o pomaku. σ_x nastalog "eliptičkog" *gaussian*-a je određen duljinom pomaka. Ta činjenica se može iskoristiti u prvoj aproksimaciji duljine pomaka.

Još jedan bitan čimbenik kod ove metode aproksimacije je prag pozadine. Naime, izračunati parametri *gaussian*-a σ_x i σ_y se mijenjaju variranjem praga pozadine. Što je prag pozadine veći, manje piksela ulazi u proračun parametara, tj. *gaussian* postaje uži i kraći pa su samim time i vrijednosti σ_x i σ_y manje. U idealnom slučaju kad prag pozadine ne bi postojao, dovoljno bi bilo uzeti da širina presjeka *gaussian*-a u smjeru σ_x iznosi oko $6\sigma_x$, a u smjeru σ_y oko $6\sigma_y$. Pomak koji se traži je pomak težišta *gaussian*-a na dvije uzastopne slike. Budući da idealni "kružni" *gaussian* i pomakom nastali "eliptički" *gaussian* imaju jednaki σ_y, prema slici Sl 2.12 se lako može pokazati da je očekivani pomak tada jednak:

$$d = 6\sigma_x - 6\sigma_y \tag{36}$$

Povećanjem praga pozadine smanjuju se vrijednosti σ_x i σ_y. Pošto je σ_y očekivano uvijek manji od σ_x, smanjenjem vrijednosti parametara se smanjuje i aproksimirani pomak d. Iz navedenog je očito da točnost aproksimacije znatno ovisi o pragu pozadine.



Sl 2.12 Aproksimacija pomaka

Da bi takva aproksimacija pomaka imala smisla potrebno je pretpostaviti da je senzor aktivan za vrijeme cijelog postupka promatranja određenog dijela neba. Pretpostavka uglavnom vrijedi, jer vrijeme, koje je potrebno za obradu između dvije uhvaćene slike, je zanemarivo u odnosu na vrijeme ekspozicije senzora. Dok traje obrada slike zvijezde se i dalje miču, no taj pomak nije toliko velik da bi opovrgnuo pretpostavku. Pretpostavka uglavnom vrijedi dok se aproksimirani *gaussian*-i određene zvijezde na dvije uzastopne slike preklapaju.

Još je potrebno pretpostaviti da su izračunati parametri σ_x i σ_y za određenu zvijezdu jednaki na dvije uzastopne slike, odnosno da su jednaki za sve slike na kojim se zvijezda pojavljuje. Pretpostavka vrijedi ako je vrijeme ekspozicije senzora približno jednako za svaku sliku, a to znači da se zvijezde tokom hvatanja svake slike pomaknu približno za isti iznos.

Za aproksimaciju pomaka na ovaj način dovoljna je samo jedna slika i može poslužiti kao referenca kod traženja stvarnog pomaka usporedbom uzastopnih slika. Ovom metodom se

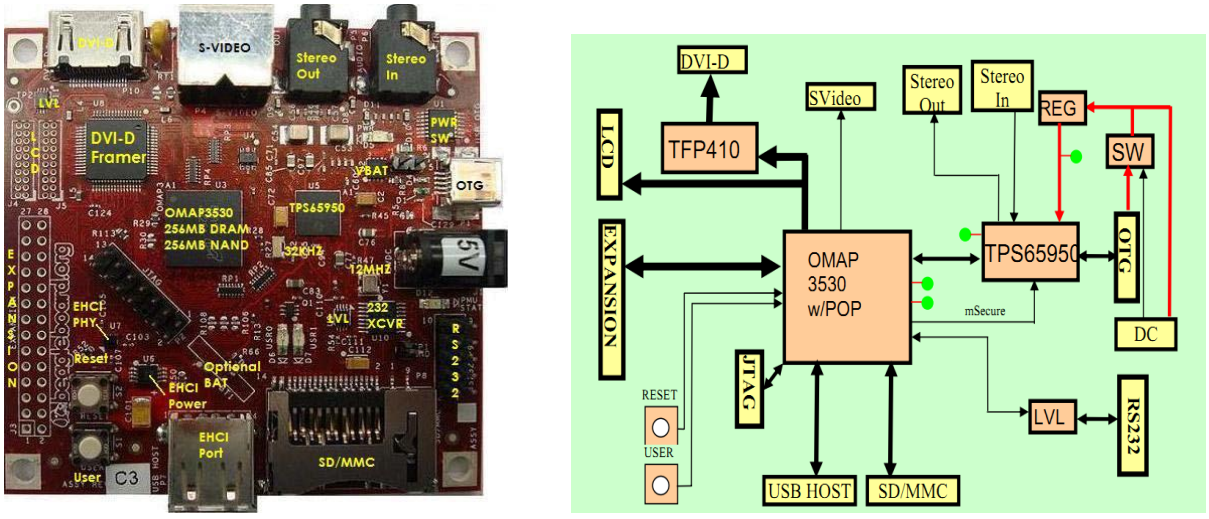
može odrediti samo iznos pomaka, a ne i smjer. Zbog toga je nužno usporediti dvije uzastopne slike da bi se našao pomak. Smjer pomaka se određuje iz stvarnog pomaka težišta konkretne zvijezde, no opet nije nužno da je iznos tako određenog pomaka stvaran, budući da su izračunati parametri svakog *gaussian*-a samo aproksimacija zvijezde *gauss*-olikom funkcijom. Iz tog razloga se kod konačne aproksimacije smjera i iznosa pomaka trebaju uzeti u obzir i parametri *gauss*-a i parametri dobiveni računanjem pomaka usporedbom dvije uzastopne slike. Prije navedena aproksimacija pomaka na osnovu jedne slike u tom slučaju opet može poslužiti kao referenca.

3. Realizacija sustava

Za konkretnu realizaciju sustava za automatsko praćenje zvijezda se u radu koristio Beagleboard razvojni sustav. Kao senzor za hvatanje slike se koristila web kamera umjesto integrirane teleskopske kamere. Rezultati analize slika prikazuju se u terminalu Beagleboard radne površine i u SIMULINK sučelju.

3.1. Beagleboard razvojni sustav

BeagleBoard je ugradbeni računalni sustav koji se zasniva na OMAP3530 sustavu na čipu (SoC, *system on chip*). Podržava većinu funkcija stolnog osobnog računala pa se može govoriti tzv. *mini* stolnom računalu vrlo male potrošnje. Razvijen je prvenstveno za *Open Source* zajednicu te je karakterističan po svojoj pristupačnoj cijeni i vrlo moćnim performansama što ga posebno čini zanimljivim za korištenje u edukacijske svrhe i u *low-budget* projektima. Potrošnja sustava doseže nevjerojatno malih 2W pa može biti napajan preko USB priključka, no osiguran je i dodatan priključak za 5V napajanje. Glavna značajka sustava je OMAP3530 sustav na čipu koji u zajedničkom kućištu ima implementirani ARM mikroprocesor temeljen na Cortex-A8 arhitekturi, PowerVR SGX 2D/3D grafički procesor i TMS320C64x DSP procesor za obradu digitalnih signala.



Sl 3.1 Beagleboard razvojni sustav i pripadna blok shema

Temeljne karakteristike BeagleBorad sustava su:

- OMAP3530DCBB72 SoC (720MHz)

- 256 MB NAND i 256 MB MDDR SDRAM (166 MHz) memorije
- Podrška za *debug* : 14-pin JTAG, UART
- USB sučelje:
    - HS USB 2.0 OTG Port
    - USB HOST priključak s podrškom za USB HUB (grananje)
- Sučelje za SD/MMC kartice
- Video izlaz S-video i digitalni DIV-D
- Stereo izlaz i stereo ulaz
- RS232 konektor
- 2 priključka za LCD

Kao što je rečeno BeagleBoard sustav je osmišljen da se što bolje približi performansama stolnog računala. U prvoj liniji je namijenjen za rad pod operacijskim sustavom, pa tako kao i stolno računalo podržava nadogradnju različitim perifernim jedinicama i sklopovima.

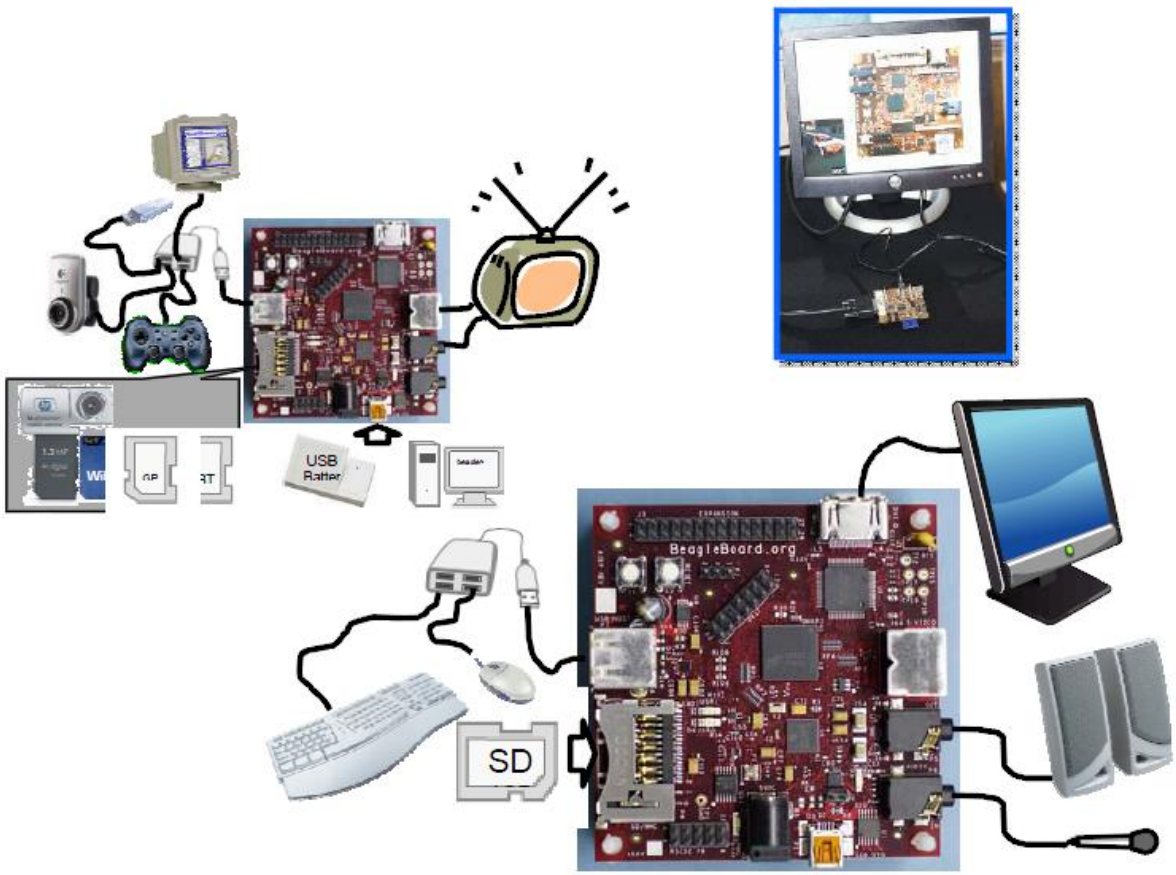
Moguća proširenja su:

- Priključenje digitalnog računalnog zaslona preko DVI-D priključka
- Kompatibilnost sa velikim skupom USB periferija uključujući HUB (grananje), tipkovnice, miša, WiFi, Bluetooth, web kamere itd.
- MMC+/SD/SDIO sučelje za priključivanje vanjske memorije, a može se koristiti i za priključenje uređaja koji podržavaju SDIO (*Secure Digital Input Output*) kao što su GPS prijemnici, Wi-Fi ili Bluetooth adapteri, modemi, Ethernet adapteri, barkod čitači, FM radio prijemnici, digitalni fotoaparati itd.

Sl 3.2 Primjeri uređaja koji podržavaju SDIO

- S-Video izlaz za priključenje NTSC ili PAL televizijskog sustava
- Stereo audio ulaz i izlaz za spajanje mikrofona i zvučnika

Sustav je moguće i proširiti dodatnim sklopovljem preko sučelja za nadogradnju (*expansion header*) spajanjem dodatnog *daughterboard*-a.



Sl 3.3 Moguća proširenja Beagleboard sustava

3.2. Logitech c310 Web kamera

Korištena web kamera je Logitech c310 kojom je moguće hvatanje slika visoke razlučivosti. Kamera se temelji na CMOS optičkom senzoru, sa fokalnom duljinom 4.4mm i kutom pogleda od 60°. Maksimalna moguća rezolucija slike je 1280x960 piksela, a maksimalni podržani *frame rate* 30 slika po sekundi.

Povezivost	High Speed USB 2.0
Mikrofon	Ugrađeni sa potiskivanjem šuma
Leća i tip senzora	Plastična leća, CMOS optički senzor
Tip focusa	Fiksni
Kut pogleda (FOV)	60°
Focalna duljina	4.4mm
Razlučivost	1280 x 960 VGA
Hvatanje slike (4:3 SD)	320x240, 640x480, 1.2MP, 5.0MP
Hvatanje slike (16:9 W)	360p, 480p, 720p
Hvatanje videa (4:3 SD)	320x240, 640x480, 800x600
Hvatanje videa (16:9 W)	360p, 480p, 720p
***Frame Rate* (max)**	30fps @ 640x480
Indikator (LED)	Aktivnost/Power

Tablica 3.1 Osnovne značajke *Logitech c310 kamere*

Kamera se koristila iz razloga što je podržana od strane Linux Angstrom distribucije operacijskog sustava na Beagleboard sustavu. Dodatno kamera podržava konfiguriranje vremena ekspozicije što je kod astrofotografije od iznimne važnosti.

4. Programska podrška

Beagleboard je posebno zanimljiv jer se osim ARM procesora u istom čipu nalazi i DSP procesorska jezgra. Ako se osim ARM jezgre želi koristiti i DSP jezgra potrebno je imati dvije odvojene aplikacije koje se paralelno izvršavaju. Također bi bilo poželjno da postoji nekakva vrsta komunikacije između jezgri te se u tu svrhu može koristiti *DSP/BIOS Link* programsko sučelje sa pripadajućim *library*-em.

4.1. Programska podrška za ARM

U projektu se koristio MATLAB programski paket za Windows operacijski sustav (OS). Dio MATLAB okruženja koji je bio najviše korišten je SIMULINK. Posebno zanimljiv dio SIMULINK-a čine *Embedded coder* i *Simulink coder* te je navedena podrška glavni razlog zašto se koristi MATLAB okruženje za razvoj programske podrške. *Embedded coder* i *Simulink coder* generiraju C ili C++ programski kod prema modelu u SIMULINK-u. Dobiveni kod je prilagođen određenoj platformi odnosno za rad na određenom procesoru ili mikrokontroleru ovisno o konfiguraciji. Podržane su brojne platforme i procesori kao npr. OMAP3530 na Beagleboard sustavu ili Blackfin DSP procesori. Još jedna dobra značajka je što MATLAB nudi podršku za stvaranje potpunog projekta u nekom od komercijalno dostupnih okruženja specijalizirana za razvoj programske podrške za pojedine procesore kao što su Analog Devices VisualDSP++, Code Composer Studio ili Eclipse. Po potrebi se može dodati i vlastiti kod sa potrebnim *library*-ima koji će kod generiranja projekta biti ukomponiran u dio koda koji generira *Embedded coder* ili kao datoteka uključen u projekt.

U SIMULINK-u postoje posebni blokovi koji su prilagođeni isključivo za generiranje C ili C++ koda. To su blokovi koji služe za komunikaciju između ugradbenog (*target*) sustava i računala (*host*), blokovi koji generiraju kod za rad pod *embedded* linux OS na target sustavu ili u slučaju DSP procesora blokovi za digitalnu obradu signala koji generiraju optimirani kod za pojedinu porodicu. Svi navedeni blokovi dolaze sa *Embedded coder* i *Simulink coder* podrškom. Kod kreiranja modela u SIMULINK-u mogu se koristiti i standardni blokovi kao što su npr. različite matematičke operacije, no u tom slučaju generirani kod ne mora biti optimalan i najčešće se sporije izvode pojedine operacije koje predstavljaju pojedini blokovi. Preporučljivo je proučiti *help* ako se koristi blok koji izvodi zahtjevnije operacije ili koji nije standardan.

Prednost ovakvog načina razvoja programske podrške je što uvelike ubrzava sam razvoj. Kod je pregledniji i dobro komentiran, te se može pratiti gdje koji blok završava i slijedeći počinje. Komentare generira MATLAB, no po potrebi se mogu i isključiti.

Još jedna funkcionalnost koju nudi *Simulink coder* je *Real-Time External Mode*. U tom modu moguće je pratiti signale i izlazne vrijednosti pojedinih blokova na *host* računalu dok se na *target* sustavu izvršava generirani program. U tu svrhu se koriste tzv. S*inks* blokovi kao što su npr. *Scope* ili *Vector Scope*. Osim toga moguće je u direktno mijenjati parametre pojedinih blokova kao što su npr. pojačanje ili frekvencije generiranih signala. Promijenjeni parametri se prosljeđuju *target* sustavu. Na taj način se neposredno utječe na *target* sustav za vrijeme dok se program izvršava te samim tim nudi niz pogodnosti i značajno ubrzava razvoj konačnog sustava. Komunikacija između sustava je sinkronizirana sa *real-time clock*-om i time sinkronizirana sa *Sample Time*-om (vrijeme uzorkovanja). U *External mode*-u se na *host* računalu izvršava aplikacija koja koristi I/O (ulazno-izlazne) *driver*-e koji komuniciraju sa *target* sustavom i spremaju primljene podatke u međuspremnik smješten u dijelu memorije kojoj SIMULINK ima pristup. Dok se međuspremnik napuni, SIMULINK premješta podatke iz međuspremnika u MATLAB-ovo okruženje. Nakon što je premještanje obavljeno, SIMULINK iscrtava podatke npr. u *Scope* bloku, ili podaci mogu biti spremljeni u tzv. MAT datoteku ako je uključena *data archiving* opcija u *external mode*-u.

Aplikaciju za ARM jezgru je moguće generirati iz SIMULINK modela, no ne mogu se koristiti *DSPlink* blokovi iz razloga što se isti blokovi ne koriste kod aplikacije za DSP već se za DSP posebno piše vlastiti kod koristeći samo *DSP/BIOS Link library*. Budući da postoji više načina komunikacije i modula kod *DSP/BIOS Link*-a, kod ručno pisanog koda mogu postojati bitne razlike u funkcionalnosti i samoj komunikaciji u odnosu na izvedbu *DSPlink* blokovima. Iz navedenog razloga se u generirani kod za ARM jezgru moraju dodati dijelovi koda koji se odnose na komunikaciju između jezgri zajedno sa pripadnim *DSP/BIOS Link library*-em.

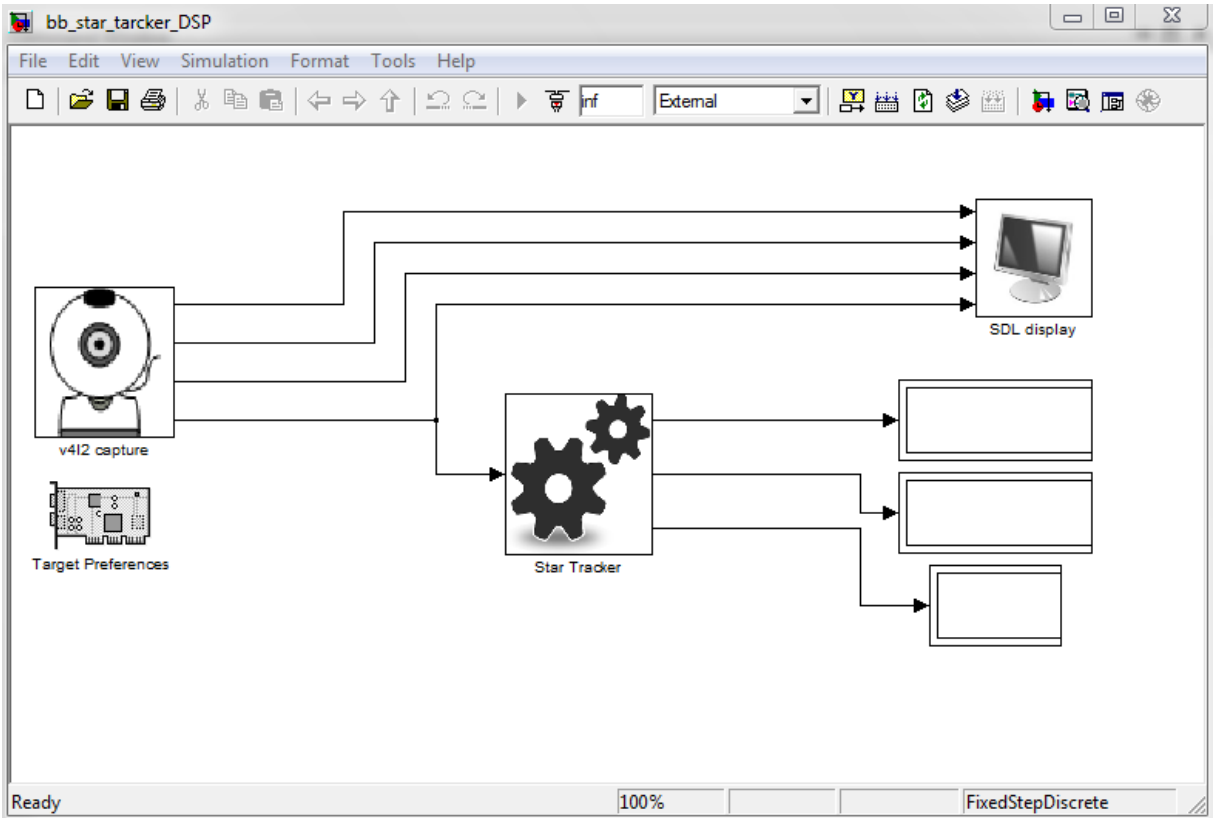
4.1.1. Eclipse

U prijašnjem tekstu je rečeno da MATLAB nudi podršku za stvaranje potpunog projekta u nekom od komercijalno dostupnih okruženja specijalizirana za razvoj programske podrške za pojedine procesore. U ovom projektu je kao razvojno okruženje (IDE) odabran *Eclipse*. *Eclipse* je programska razvojna okolina pisana u Javi, a može se koristiti za razvoj aplikacija u raznim programskim jezicima kao što su Java, Ada, C, C++, COBOL, Perl, PHP, Python, R,

Ruby, Scala, Clojure i Scheme. Postoje razne inačice razvojne okoline pa je često nalazimo pod različitim nazivama npr. *Eclipse ADT* za Adu, *Eclipse CDT* za C/C++, *Eclipse JDT* za Javu *i Eclipse PDT* za PHP. Sastoji se od nekoliko važnijih dijelova kao što su: editor izvornog koda (*source code editor*) i program za ispravljanje pogrešaka (*debugger*). Postoje razne dodatne opcije i mogućnosti kojima se može nadograditi kao npr. *Remote System Explorer* koji je posebno koristan kod sustava koji rade pod *embedded* linux OS jer omogućava pristup *file system*-u sustava preko *etherneta* i razne druge mogućnosti. U projektu je korištena verzija *Eclipse Ganymede IDE for C/C++ Developers*. Prevodilac (*compiler*) je posebno instaliran za ARM procesore i uključen u *Eclipse* razvojno okruženje tako da je na Windows *host* računalu moguće prevoditi aplikacije koje trebaju raditi pod linux OS na određenom ARM procesoru. Takav prevodilac se još naziva *cross compiler* i koristi se za prevođenje aplikacija za ugradbene sustave.

4.1.2. Model u SIMULINK-u

Glavni dio programske podrške za ARM procesorsku jezgru se realizira pomoću SIMULINK modela. Uz korištenje standardnih SIMULINK blokova moguće je dodati vlastite blokove koji izvršavaju nekakav program ili dio programa pisan u nekom od programskih jezika (najčešće C ili C++).

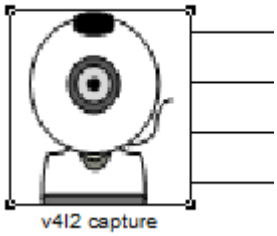


Sl 4.1 Realizirani model u SIMULINK-u

Konkretni model čita sliku sa web kamere te pročitanu sliku šalje na obradu. Kod obrade se traže pojdine zvijezde koje se tada izdvajaju te se izdvojene regije šalju DSP jezgri na aproksimaciju parametara *gaussian*-a. Radi usporedbe je ostavljena mogućnost isključenja DSP jezgre iz procesa obrade, pa se parametri na isti način mogu računati i na samoj ARM jezgri. Na temelju izračunatih parametara *gaussian*-a na trenutnoj i prethodnoj slici se tada vrši određivanje smjera i iznosa pomaka zvijezda na slikama, odnosno pomak trenutne slike u odnosu na prethodnu.

U *terminal* aplikaciji na Beagleboard *Desktop*-u se ispisuju parametri *gaussian*-a za svaku nađenu zvijezdu, a u SIMULINK sučelju se prikazuje srednji smjer i kut pomaka cijele slike u odnosu na prethodnu, kao i broj zvijezda na temelju kojih se pomak računao i koje su uspješno prepoznate i uparene na objim slikama. Pročitana slika se također prikazuje na Desktop-u Beagleboard sustava.

4.1.3. Blok *v4l2 capture*



Sl 4.2 Blok za čitanje slike sa kamere

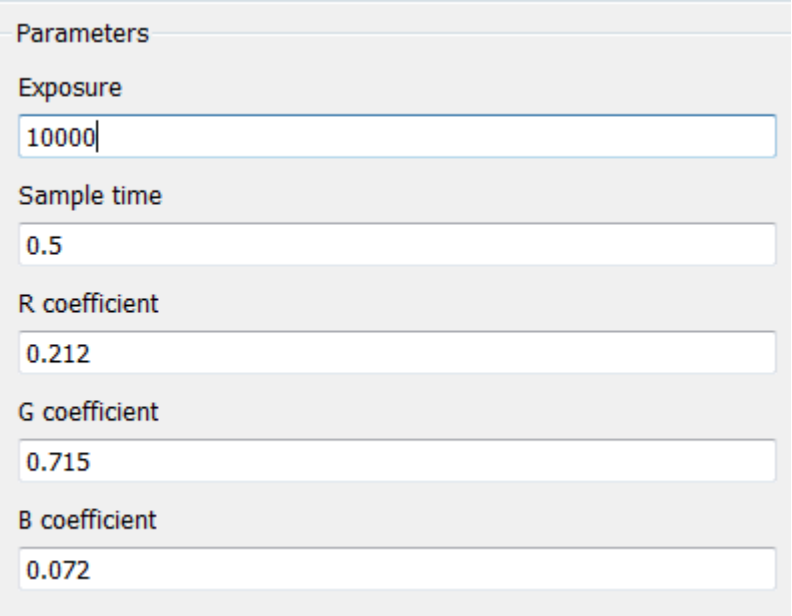
Za čitanje slike sa kamere izrađen je poseban blok u SIMULINK-u. Blok koristi v4l2 linux API za dohvaćanje slike sa kamere, a programski kod je pisan u C-u. v4l2 omogućava također i mijenjanje parametara kamere, kao što su razlučivost, ekspozicija i slično. Blok osim što čita sliku sa kamere, pročitanu sliku pretvara u *grayscale*. Blok ima predefiniranu razlučivost slike koju čita sa kamera, kao i format zapisa pojedinih piksela.

Kao ulazne parametre blok uzima trajanje eskpozicije, koeficijente za pretvorbu slike u *grayscale* i vrijeme uzorkovanja (*sample time*) koje se koristi samo u svrhe *debug*-iranja. Točno vrijeme uzorkovanja, odnosno vrijeme koje je potrebno da prođe jedan ciklus koji se sastoji od hvatanja, obrade i prikaza slike ovisi o samoj slici, te broju i obliku zvijezda na slici. Iz tog razloga nije moguće zadati točno vrijeme uzorkovanja jer može znatno varirati od slike do slike. Svi parametri se mogu mijenjati po želji i za vrijeme rada sustava.

Koeficijenti koji se koriste za pretvorbu u *grayscale* se mogu po volji odabrati, a pretvorba se vrši tako da se svaka komponenta boje pojedinog piksela množi sa zadanim koeficijentom pa potom zbrojeni umnošci predstavljaju piksel u *grascale* formatu. Zbog karakteristika i osjetljivosti ljudskog oka preporuča se i uzeti su koeficijenti kao na slici Sl 4.3. Iako se parametri mogu mijenjati po volji treba paziti na preljev tijekom pretvorbe. Iz tog razloga treba voditi računa da ukupan zbroj koeficijenata bude manji od 1.

$$R_{coeff} + G_{coeff} + B_{coeff} < 1 \tag{37}$$

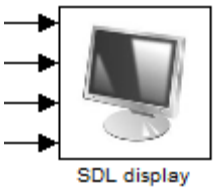
Ekspozicija se može mijenjati po želji, međutim C kod koji stoji iza bloka, ovisno o mogućnostima korištene kamere, ograniči ekspoziciju u nekom rasponu. Korištena web kamera ne može imati veću ekspoziciju od 10000.



Sl 4.3 Korišteni parametri *v4l2 capture* bloka

Izlazi iz bloka su 4 matrice piksela. Prve 3 predstavljaju sliku rastavljenu po pojedinim komponentama boja (crvena, zelena i plava). Četvrti izlaz daje matrica piksela slike u *grayscale* formatu.

4.1.4. Blok *SDL display*

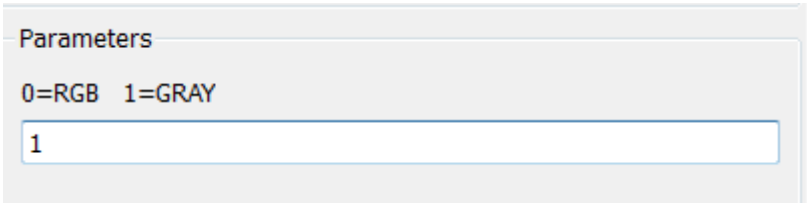


Sl 4.4 Blok za prikaz slike na *Desktop*-u

Za prikaz slike na *Desktop*-u Beagleboard sustava izrađen je poseban blok u SIMULINK-u. Blok koristi SDL (*Simple DirectMedia Layer*) *library* otvorenog koda koji predstavlja

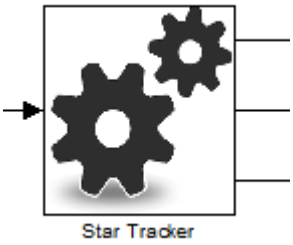
omotač (*wrapper*) oko karakterističnih funkcija korištenog operacijskog sustava. Pruža podršku za 2D operacije nad pikselima, rukovanje zvokom, pristup datotekama, posluživanje događaja (*event handling*), postavljanje brojača, stvaranje *thread*-ova te se često koristi u kombinaciji sa OpenGL-om kod raznih multimedijskih aplikacija i igara. Slično kao i kod v4l2 capture bloka, SDL display blok ima predefiniranu razlučivost slike koju očekuje na ulazima, kao i format zapisa pojedinih piksela.

Blok ima 4 ulaza od kojih 3 očekuju sliku rastavljenu po pojedinim komponentama boja (crvena, zelena i plava). Četvrti ulaz očekuje matricu piksela slike u *grayscale* formatu. Istovremeno može samo jedan način prikaza biti aktivan, tj u nekom trenutku je moguć prikaz slike ili u boji ili u *graycale*-u. Koji prikaz je aktivan se zadaje u parametrima bloka, te se isti može mijenjati tijekom rada sustava.



Sl 4.5 Parametri *SDL display* bloka

4.1.5. Blok *Star tracker*



Sl 4.6 Blok za obradu slike i estimaciju parametara zvijezda

Blok služi za obradu slike koja podrazumijeva traženje pojdinih regija zvijezde koje se tada izdvajaju te se nad njima vrši estimacija parametara *gaussian*-a. Nakon estimacije se na osnovu trenutnih parametara i parametara svih zvijezdi sa prethodne slike, koja je bila

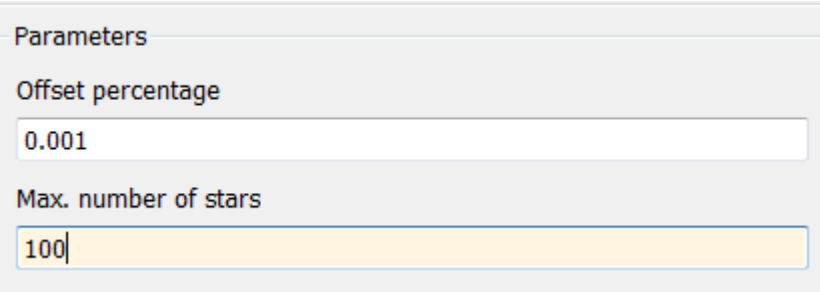
uhvaćena u prošlom ciklusu, određuje smjer i iznos pomaka trenutne zvijezde u odnosu na uparenu zvijezdu na prethodnoj slici.

Zvijezde se traže prema najsvijetlijem pikselu na slici. Kad je najsvijetliji piksel nađen algoritam određuje regiju zvijezde. Nađena regija se šalje DSP jezgri koja računa parametre *gaussian*-a za izdvojenu zvijezdu. Estimirane parametre DSP jezgra šalje natrag ARM jezgri. Na kraju se nađena regija sa zvijezdom briše (nulira) sa trenutne slike i traži se nova pa se postupak ponavlja za sljedeću zvijezdu. Obrada se smatra završenom dok na slici više nema piksela koji su po svojem iznosu iznad praga pozadine. Usporedbe radi je ostavljena mogućnost da se parametri *gaussian*-a računaju na samoj ARM jezgri bez korištenja DSP jezgre. Ako se koristi DSP jezgra, implementirano je da se za vrijeme računanja parametara za trenutnu zvijezdu na DSP jezgri, na ARM jezgri traži smjer i iznos pomaka prethodno nađene zvijezde na trenutnoj slici u odnosu na prethodnu sliku. U slučaju da se ne koristi DSP jezgra, određivanje pomaka se ne računa paralelno sa estimacijom parametara već nakon estimacije i prije traženja nove zvijezde.

Algoritam za određivanje pomaka prvo traži i uparuje zvijezdu sa istom na prethodnoj slici. Predefinirano je da pomak zvijzdezde na prethodnoj slici ne smije odstupati više od 50% u odnosu na pretpostavljeni pomak te da kut zakreta *gaussin*-a zvijezde na prethodnoj slici mora biti u granicama od ±15° u odnosu na traženu zvijezdu sa trenutne slike. Zvijezda sa prethodne slike koja zadovoljava navedene kriterije i najmanje odstupa od istih, uparuje se sa zvijezdom na trenutnoj slici. Samo uparene zvijezde ulaze u konačan izračun iznosa i smjera pomaka trenutne slike u odnosu na prethodnu.

Na svom ulazu blok očekuje matricu piksela *grayscale* slike. Razlučivost kao i format piksela su predefinirani. Izlazi bloka su srednji iznos pomaka trenutne slike u odnosu na prethodnu, smjer (kut) pomaka i broj zvijezdi kojima je nađen par na prethodnoj slici i koje su ulazile u proračun konačnog pomaka.

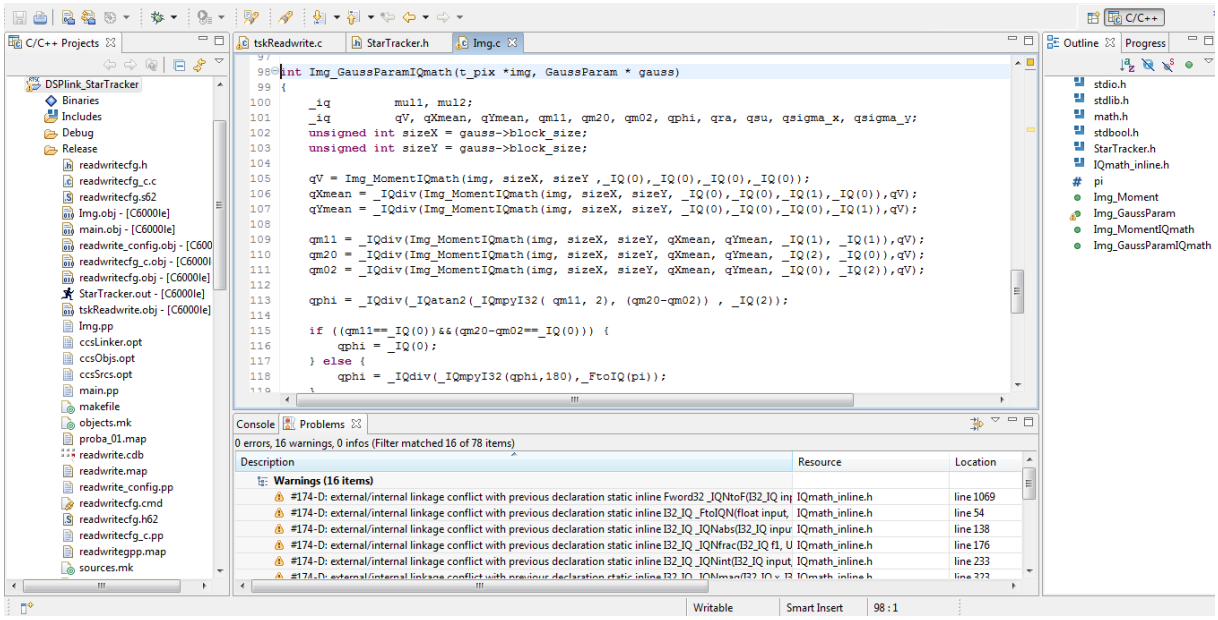
Ulazni parametri bloka za obradu su maksilmani broj zvijezdi koje se traže na slici i postotak najsvijetlijih piksela koji se interpretiraju kao dio zvijezde. Postotkom najsvijetlijih piksela se određuje prag pozadine.



Sl 4.7 Parametri *Star tracker* bloka

4.2. Programska podrška za DSP

Iako u SIMULINK-u postoje *DSPlink* blokovi za komunikaciju između DSP i ARM jezgre na Beagleboard sustavu, korištena verzija MATLAB-a nema podršku za generiranje koda za DSP jezgru prema modelu u SIMULINK-u. Iz tog razloga se za DSP jezgru mora aplikacija posebno pisati u nekom od okruženja specijalizirana za razvoj programske podrške za DSP jezgru na Beagleboard sustavu. Aplikacija za DSP jezgru je pisana u *Code Composer Studio*-u. DSP radi pod jednostavnim operacijskim sustavom zvan DSP BIOS, a za međuprocesorsku vezu se koristi DSP/BIOS Link programsko sučelje.



Sl 4.8 Podrška za DSP u *Code Composer Studio*-u

DSP se koristi isključivo za aproksimaciju i računanje parametara pojedinih *gaussian*-a. Ulazni parametri su izdvojena regija zvijezde sa uklonjenom pozadinom i veličina regije koju DSP dobiva od ARM jezgre. Rezultat obrade su parametri *gaussian*-a koji se vraćaju AMR jezgri. Za računanje parametara se koristi *library* IQmath verzije 2.01.04.00, koji je posebno optimiran za danu porodicu DSP procesora C64xx. IQmath *library* je zbirka vrlo optimiziranih matematičkih funkcija visoke preciznosti u C/C++ programskom jeziku kojima se omogućuje jednostavna pretvorba *floating-point* algoritama u frakcionalnu aritmetiku.

Za međuprocesorsku komunikaciju se koristi *DSP/BIOS Link driver* i *library* verzije 1.65.01.06.

DSP aplikacija se sastoji od jednog *task*-a koji se izvršava u 3 faze:

- Inicijalizacija
- Glavna petlja (izvršavanje)
- Deinicijalizacija i završetak izvršavanja

U fazi inicijalizacije se inicijaliziraju potrebni *DSPLink* moduli korišteni za međuprocesorsku komunikaciju, kao i semafor za sinkronizaciju *task*-a sa dolaznim porukama od ARM jezgre.

Faza izvođenja se sastoji od glavne petlje koja čeka poruke od ARM jezgre koje sadrže izdvojenu regiju zvijezde sa uklonjenom pozadinom i veličinu regije. Nad regijom se tada računaju momenti potrebni za aproskimaciju parametara *gaussian*-a. Izračunati parametri se tada šalju natrag ARM jezgri, a to su:

- *Xmean, Ymean* – težište *gaussin*-a
- *V* – volumen
- *A* – vrijednost *gaussian*-a u težištu
- *sigma_x, sigma_y* – standardna devijacija duž dužu i kraću os
- *phi* – kut zakreta *gaussian*-a

U posljednjoj fazi DSP zatvara sve inicijalizirane *DSPLink* module i oslobađa alociranu memoriju. Nakon toga program izlazi iz *task* funkcije čime se prekida izvršavanje *task*-a te više ne može biti pozvan.

Za računanje se koristi frakcionalna aritmetika, iako je ostavljena opcija za računanje parametara u *floating point* aritmetici dvostruke preciznosti. Korištena je frakcionalna aritmetika Q13, no ostavljeno je prostora za jednostavnu promjenu u neki drugi Qn format u

rasponu od Q0 do Q31. Prije vraćanja rezultati se pretvaraju nazad u *floating point* brojeve dvostruke preciznosti.

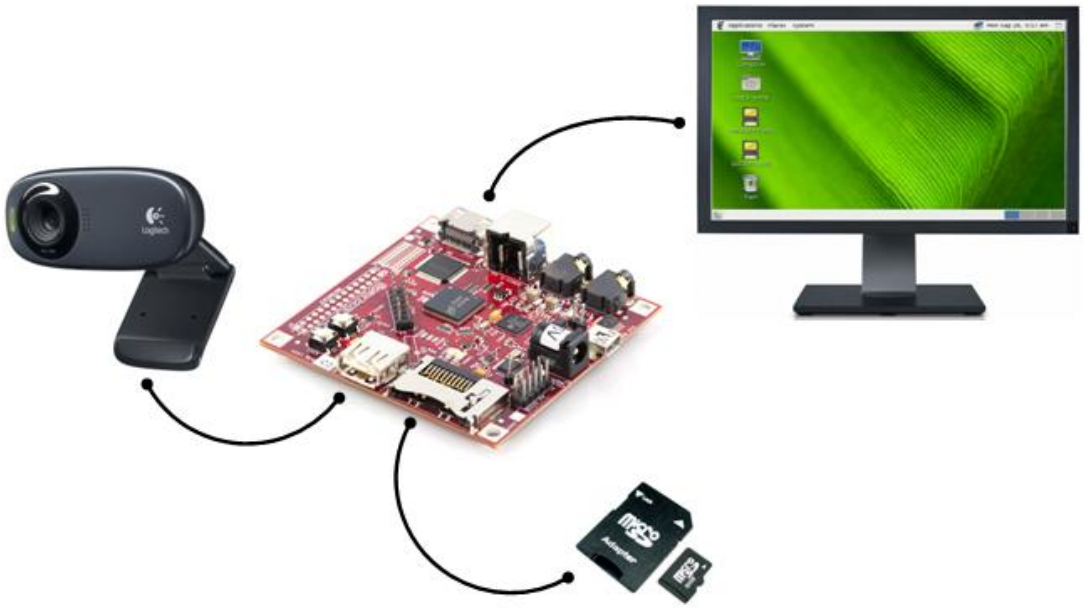
Kod konkretnog algoritma za računanje parametara *gaussian*-a, glavni problem frakcionalne aritmetike, osim što se unosi određena pogreška kod kvantizacije i računa, je činjenica da frakcije lako ulaze u zasićenje i dolazi do preljeva. Konkretno, problem se javlja kod računanja momenata. Ako je područje regije zvijezde dovoljno veliko i ako se u regiji nalazi velik broj piksela zvijezde, tada kod kumulativnog zbrajanja vrijednosti pojedinih piksela pomnoženih sa određenim faktorom lako dolazi do preljeva.

Primjerice, kod Q13 aritmetike, frakcija može biti u rasponu od -262144 do 262143.999 877 930. Teoretski, moguće je da svi piskeli u regiji zvijezde budu postavljeni na maksimalnu vrijednost koja je 255 kod 8 bitne reprezentacije vrijednosti piksela. Već kod računanja volumena ispod regije se lako dobije da regija ne smije biti veća od 32x32 piksela, inače teoretski postoji opasnost da će doći do preljeva. Kod astronomskih slika vjerojatnost je mala da baš svi pikseli regije budu postavljeni na najveću vrijednost. Pretpostavlja da se težište zvijezde nalazi u centru regije i da vrijednost piksela naglo trne prema rubovima. Zbog te činjenice je uzeto eksperimentalno da maksimalna površina regije bude 20000 piksela, što je 141x141 piksel u slučaju regije u obliku kvadrata (konkretno se samo kvadratne regije koriste). Ako je regija veća od 141x141 piksela, zvijezda se smatra prevelikom i parametri se ne računaju.

Dodatno je ostavljena mogućnost da parametri *gaussian*-a budu računati u *floating-point* aritmetici dvostruke preciznosti. Kod *floating-point* aritmetike ne dolazi do preljeva, ali se sporije se izvršava.

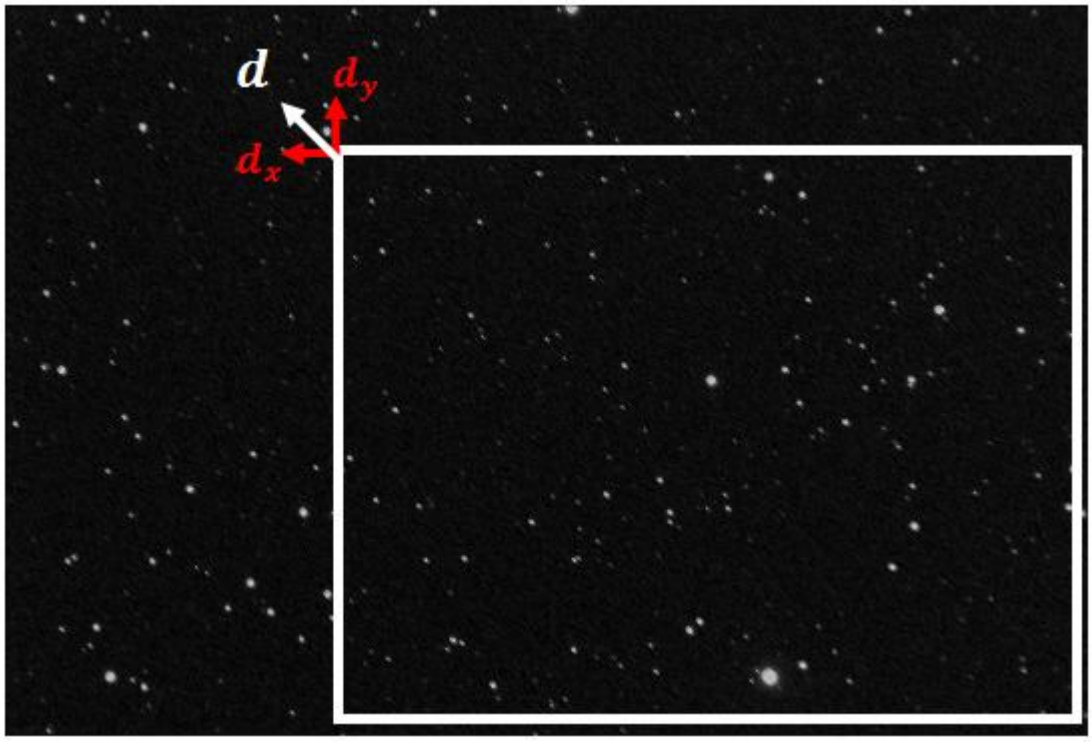
5. Simulacija i testiranje

Simulacijom se želi utvrditi koje su prednosti a koji nedostaci metode praćenja zvijezda i pomaka neba uz korištenje različitih načina određivanja parametara i pomaka. Simulacija se provodi sa i bez korištenja DSP jezgre za računanje parametara *gaussian*-a. Time se želi utvrditi koliko se brže obrada izvodi kad DSP i ARM jezgra rade paralelno u odnosu dok se sva obrada vrši samo na ARM jezgri. Kod korištenja DSP jezgre još se želi istražiti točnost u računanju sa različitim frakcijama u odnosu na isti algoritam u *floating-point* aritmetici dvostruke preciznosti također izvođen na DSP jezgri.



Sl 5.1 Sustav korišten za potrebe simulacije

Za potrebe simulacije napravljena je sintetička slika neba noću sa zvijezdama. Slika je generirana iz jedne fotografije koja je duplicirana i preklopljena preko originala uz mali pomak od jednog piksela u x i y smjeru. Time su zvijezde na slici izdužene i simulira se pomak neba uslijed duge ekspozicije kamere. Da bi se simuliralo više ciklusa hvatanja i obrade slike te pratio pomak, dodaje se prozor na dobivenu sliku koji se pomiće za isti iznos i smjer pomaka kao prije preklopljena slika. Za svaki pomak se generira nova slika iz prozora. Na taj način se smjer i iznos pomaka točno poklapaju sa pomakom i kutom zakreta *gaussian*-a pojedinih zvijezdi.



Sl 5.2 Postupak pomicanja prozora radi simulacije pomaka

Slika Sl 5.2 je već generirana preklapanjem svoje kopije preko originala, te se nad takvom slikom pomiče okvir u smjeru d_x i d_y te dobiva sekvenca slika vidljivih u okviru. Pomak d_x i d_y iznose točno jedan piksel, stoga je iznos rezultantnog pomaka jednak 1,41421 piksel i to je upravo vrijednost koja se traži kod simulacije. Budući da se pomicanjem okvira dobije dojam da se zvijezde kreću u suprotnom smjeru pod kutom od -45°, dobivena sekvenca slika se obrađuje suprotnim redoslijedom, tako da smjer, odnosno kut pomaka bude jednak 135° kao i kod pomaka okvira.

Za potrebe simulacije uzete su 3 uzastopne slike iz sekvence generiranih slika. Budući da se koriste 3 slike, simuliraju se 3 ciklusa obrade slike. Korišteni model, umjesto sa kamere, čita generirane slike i šalje na obradu. Parametri bloka za obradu podataka su:

- *Offset percentage = 0.001 (0.1%)*
- *Max. Number of stars = 100*

Ostali blokovi u modelu se mogu zanemariti budući da se slika ne čita sa kamere. Međutim simulira se vrijeme koje je potrebno za dohvat slike sa kamere. Dakle kamera je aktivna i dohvaća sliku sa maksimalnom ekspozicijom koja iznosi 10000 za danu web kameru, no uhvaćena slika ne odlazi na obradu, već se obrađuje prije generirana sekvenca slika. Slika koju dohvaća kamera je predefinirane razlučivosti od 800x600 piksela. Ista

razlučivost se koristi i kod bloka za prikaz slike na *Desktop*-u Beagleboard sustava te se upravo uhvaćena slika i prikazuje na *Desktop*-u. Time se simulira kompletan ciklus od dohvata, preko obrade i prikaza slike, sa jednom razlikom što se umjesto uhvaćene slike obrađuju prije generirane.

Na svakoj slici se traži do 100 zvijezdi te se za svaku nađenu zvijezdu računaju parametri *gaussian*-a. Nakon aproksimacije zvijezde *gauss*-olikom funkcijom traži se ekvivalentna zvijezda na prethodnoj slici. Ako ekvivalentna zvijezda postoji, određuje se pomak trenutne zvijezde u odnosu na istu na prethodnoj slici.

5.1. Rezultati simulacije

Ista simulacija je ponovljena 10 puta za slučaj sa i bez korištenja DSP jezgre kod izračuna parametra *gaussian*-a. Dodatno u slučaju sa DSP jezgrom provedene su simulacije uz korištenje *floating-point* i frakcionalne aritmetike za izračun parametara *gaussian*-a. Time se želi istražiti odnos točnosti i brzine izvođenja algoritma računanjem sa frakcijama u odnosu na isti algoritam u *floating-point* aritmetici dvostruke preciznosti izvođen na DSP jezgri. U Tablici Tablica 5.1 nalaze se usrednjeni rezultati vremena potrebnog za puni ciklus obrade pojedinih slika za svaki od 3 slučaja.

Usporedba srednjih vremena izvršavanja (*s*)			
Slika	**Bez DSP-a**	**DSP *floating-point* aritmetika**	**DSP frakcionalna aritmetika**
1	**28.465**	**28.384**	**28.305**
2	**25.673**	**25.657**	**25.588**
3	**26.583**	**26.395**	**26.338**

Tablica 5.1 Usporedbe srednjih vremena izvršavanja

Kod korištenja DSP jezgre za izračun parametara *gaussian*-a vrijeme izvršavanja je neznatno kraće nego u slučaju bez DSP jezgre. Jednim dijelom se to dešava iz razloga što

ARM jezgra treba čekati rezultate obrade od DSP jezgre prije nego krene u daljnje proračune tako da paralelizam u izvršavanju ne dolazi pretjerano do izražaja. Drugi razlog zašto se korištenjem DSP jezgre ne ubrza proces obrade je taj što se u tom slučaju troši dodatno vrijeme na komunikaciju i razmjenu podataka između jezgri. U slučaju kad se koristi frakcionalna aritmetika na DSP jezgri postoji određena pogreška kod izračuna parametara koja se uz korištenu Q13 aritmetiku primjećuje na četvrtoj decimali izračunatih parametara *gaussian*-a u odnosu na floating-point aritmetikom dvostruke preciznosti.

Zaključak

Sustav za praćenje radi dobro za proizvoljan broj zvijezdi u praćenom dijelu neba. Međutim broj nađenih zvijezdi na slici znatno ovisi o definiranom pragu pozadine, odnosno o vrijednostima piksela za koje će se isti interpretirati kao dio zvijezde. Što je više zvijezda na slici, proces prepoznavanja i određivanja parametara i pomaka traje duže. Osim toga postoji veća vjerojatnost da će zvijezde biti krivo uparene ili neke zvijezde neće biti uopće uparene sa zvijezdama na prethodnoj slici radi određivanja pomaka. Zvijezde koje nisu uparene ne ulaze u proračun pomaka. Prag pozadine znatno utječe i na parametre nađenih zvijezdi što opet može promijeniti konačni izračunati pomak

Korištenje DSP jezgre za izračun parametara *gaussian*-a neznatno se ubrza proces obrade budući da algoritam, odnosno ARM jezgra treba čekati rezultate obrade sa DSP-a prije nego može krenuti u slijedeću fazu izvršavanja. Dodatno se izvođenje može ubrzati korištenjem frakcionalne aritmetike na DSP jezgri. Korištena frakcionalna aritmetika Q13 unosi pogrešku u krajnji rezultat, tj. izračunati pomak, a uočava se na četvrtoj decimali rezultata. Nastala pogreška se može zanemariti budući da se radi o 10000-tom dijelu piksela, no i dalje se uz samo korištenje frakcionalne aritmetike neznatno ubrza sam proces obrade.

Također, u slučaju da se slika čita sa teleskopa, na konačni rezultat utječe i optički senzor koji se koristi. Konkretno kod astrofotografije se preporuča koristiti CCD senzore jer postižu bolje rezultate u uvjetima smanjene vidljivosti, tj. manjka svijetlosti, što je konkretno slučaj kod astrofotografije.

Literatura

[1] BeagleBoard.org, BeagleBoard System Reference Manual Rev C4, 2009

[2] Texas Instruments, OMAP3530/25 Applications Processor, 2008

[3] Texas Instruments, DSP/BIOS Link User Guide Ver 1.65, Copyright ©. 2003

[4] Texas Instruments, DSP/BIOS Link Installation Guide Ver 1.65, Copyright ©. 2003

[5] Texas Instruments, TMS320 DSP/BIOS v5.41 User's Guide, 2009

[6] Texas Instruments, TMS320C6000 DSP/BIOS 5.x Application Programming Interface (API) Reference Guide, 2009

[7] Texas Instruments, TMS320C64x+ IQmath Library User's Guide, 2008

[8] Interfacing Beagleboard with Simulink and Arduino http://mattbilsky.com/mediawiki/index.php?title=Interfacing_BeagleBoard_with_Simulink_and_Arduino, prosinac 2012.

[9] A. Kavianpour, N. Bagherzadeh, S. Shoari, *Finding Elliptical Shapes ina n Image Using a Pyramid Architecture*, University of California, Irvine, CA 92717, USA, 1994

Sustav za upravljanje automatiziranim postoljem teleskopa u svrhu praćenja zvijezda

Sažetak

U radu je opisano kako je korištenjem web kamere i Beagleboard razvojnog sustava moguće realizirati sustav za automatsko prepoznavanje i praćenje zvijezdi tijekom procesa oslikavanja. Razvojni sustav inicijalizira web kameru (model Logitech c310), postavlja određeno vrijeme ekspozicije za snimanje u uvjetima smanjene vidljivosti i čita slike uhvaćene kamerom. Na uhvaćenim slikama se traže svijetli objekti, odnosno zvijezde koje se aproksimiraju *gauss*-olikim funkcijama. Na osnovu estimiranih parametara funkcija na dvije uzastopne slike određuje se pomak pojedinih zvijezdi te ukupni srednji pomak trenutne slike u odnosu na prethodnu. Uhvaćena slika se prikazuje na radnoj površini Beagleboard sustava. Pokazan je način integracije vlastitog koda u SIMULINK, generiranje programske podrške prema modelu u SIMULIK-u, te upravljanje modelom iz istog za vrijeme rada sustava, odnosno simulacije. Na konkretnim slikama uspoređene su prednosti i nedostaci korištenja DSP jezgre i frakcionalne aritmetike kod izračuna parametara zvijezdi i pomaka istih te općenito prednosti korištenja DSP jezgre kao koprocesora kod obrade slike.

Ključne riječi: astrofotografija, praćenje zvijezda, vrijeme ekspozicije, *gaussian*, prag pozadine, obrada slike, CCD optički senzor, CMOS optički senzor, Beagleboard razvojni sustav, TMS320C64x/C64x+ DSP jezgra, SIMULINK model, v4l2 API, SDL biblioteka, međuprocesorska komunikacija, DSP/BIOS Link biblioteka, *floating-point* aritmetika, frakcionalna aritmetika

Controlling system for an automated telescope mount for the purpose of star tracking

Summary

The work describes how to implement a system to automatically detect and track stars using a webcam and the Beagleboard development system. The development system initializes the webcam (Logitech C310 model), sets a specific time exposure for image shooting in conditions of reduced visibility and reads images captured by the camera. Bright objects or stars are detected on the captured images and are accordingly approximated with gauss functions. Based on the estimated parameters of the gauss functions on two continues sequential images the shift of individual stars is determined and the overall mean shift of the current image relative to the previous. The Captured images are displayed on the Beagleboard Desktop. It is demonstrated how to integrate custom code into SIMULINK, generate code from SIMULINK model and controlling the model during runtime and simulation. The advantages and disadvantages of using the DSP core and factional arithmetics for parameter calculation of stars and their shifts are compared on concrete images. The benefits of using the DSP core as coprocessor for image processing are discussed in general.

Keywords: astrophotography, star tracking, exposure time, gaussian function, the background threshold, image processing, CCD optical sensor, CMOS optical sensor, Beagleboard development system, TMS320C64x/C64x+ DSP core, Simulink model, v4l2 API, SDL library, inteprocessor communication, DSP/BIOS Link Library, floating-point arithmetic, fractional arithmetic

Privitak

```
/*****************************************************************************
*       DSPLink.c
*****************************************************************************/

/*  ----------------------------------- DSP/BIOS Link                   */
#include <dsplink.h>

/*  ----------------------------------- DSP/BIOS LINK API               */
#include <proc.h>
#include <msgq.h>
#include <pool.h>

/*  ----------------------------------- OS Specific Headers             */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <malloc.h>
#include <math.h>

/*  ----------------------------------- APP HEADER                      */
//#include "BB_Init.h"

/* ============================================================================*
*       PROC parametri
*  ============================================================================*/
// ID DSP procesora
#define DSP_ID              0
// broj argumenata koji se salje DSP aplikaciji
#define BROJ_ARG            1

/* ============================================================================*
*       MSGQ parametri i strukture
*  ============================================================================*/
// struktura MSGQ poruke
typedef struct MSG_Poruka_tag {
    MSGQ_MsgHeader msgHeader     ;
    t_pix          img[MAX_DSP_IMAGE_SIZE];
    GaussParam     gauss;
} MSG_Poruka ;
// velicina MSGQ strukture (poruke)
#define VELICINA_MSG_PORUKE  DSPLINK_ALIGN (sizeof (MSG_Poruka), DSPLINK_BUF_ALIGN)
// Poruka
MSG_Poruka * msg ;
// Ime MSGQ za GPP
STATIC Char8 GppMsgqIme [DSP_MAX_STRLEN] = "GPPMSGQ1" ;
// Ime MSGQ za DSP
STATIC Char8 DspMsgqIme [DSP_MAX_STRLEN] = "DSPMSGQ1" ;
// GPP MSGQ objekt
STATIC MSGQ_Queue GppMsgqObjekt = (Uint32) MSGQ_INVALIDMSGQ ;
// DSP MSGQ objekt
STATIC MSGQ_Queue DspMsgqObjekt = (Uint32) MSGQ_INVALIDMSGQ ;
// argumenti za MSGQ_transportOpen()
#define SAMPLEMQT_CTRLMSG_SIZE  ZCPYMQT_CTRLMSG_SIZE
STATIC ZCPYMQT_Attrs  mqtAttrs ;

/* ============================================================================*
*       POOL parametri i strukture
*  ============================================================================*/
```



```
// POOL ID
#define POOL_ID          0
//  Number of buffer pools.
#define POOL_NUM         2

STATIC Uint32 VelicineBufferaUPoolu [POOL_NUM] =
{
    VELICINA_MSG_PORUKE,    // MSG buffer
    SAMPLEMQT_CTRLMSG_SIZE  // CTRLMSG buffer
} ;

STATIC Uint32 BrojBufferaPoPoolu [POOL_NUM] =
{
    2,                      // broj MSG buffera
    2                       // broj CTRLMSG buffera
} ;

STATIC SMAPOOL_Attrs PoolAtributi =
{
    POOL_NUM,               // broj pool-ova
    VelicineBufferaUPoolu,  // velicine buffera u pool-u
    BrojBufferaPoPoolu,     // broj buffera u pool-u
    TRUE
} ;

/* ============================================================================*
*       inicijalizacija PROC, POOL i MSGQ modula
*  ============================================================================*/
int DSPLink_InitModules (char * dspExecutable, char * strBrojIteracija)
{
    DSP_STATUS       status = DSP_SOK;
    Char8 *          args [BROJ_ARG];
    MSGQ_LocateAttrs syncLocateAttrs;

    // inicijaliziraj PROC modul
    status = PROC_setup (NULL); // inicijaliziraj
    if (DSP_FAILED (status)){ printf("Greska kod PROC_setup (). Status: [0x%x]\n", (unsigned
    int) status); return -1;}

    status = PROC_attach (DSP_ID, NULL); // pridruzi DSP
    if (DSP_FAILED (status)){ printf("Greska kod PROC_attach (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // inicijaliziraj POOL modul
    status = POOL_open ( POOL_makePoolId(DSP_ID, POOL_ID), &PoolAtributi); // otvori novi POOL
    if (DSP_FAILED (status)){ printf("Greska kod POOL_open (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // inicijaliziraj MSGQ modul
    status = MSGQ_open (GppMsgqIme, &GppMsgqObjekt, NULL); // MSGQ za GPP
    if(DSP_FAILED (status)){ printf("Greska kod MSGQ_open (). Status: [0x%x]\n",(unsigned int
    ) status); return -1;}

    // ucitaj izvrsni kod u DSP
    args [0] = strBrojIteracija; // argument DSP aplikaciji
    status = PROC_load (DSP_ID, dspExecutable, BROJ_ARG, args); // ucitaj izvrsni kod
    if (DSP_FAILED (status)){ printf("Greska kod PROC_load (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}
```

```c
    // pokreni DSP
    status = PROC_start (DSP_ID);
    if (DSP_FAILED (status)){ printf("Greska kod PROC_start (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // Omoguci MSGQ transport poruka, pridruzi MSGQ pripadajuci POOL
    mqtAttrs.poolId = POOL_makePoolId(DSP_ID, POOL_ID);
    status = MSGQ_transportOpen (DSP_ID, &mqtAttrs);
    if (DSP_FAILED (status)){ printf("Greska kod MSGQ_transportOpen (). Status: [0x%x]\n",(
    unsigned int) status); return -1;}

    // Lociraj otvoreni MSGQ na DSP procesoru
    syncLocateAttrs.timeout = WAIT_FOREVER ;
    status = DSP_ENOTFOUND ;
    while ((status == DSP_ENOTFOUND) || (status == DSP_ENOTREADY))
    {
        status = MSGQ_locate (DspMsgqIme,
        &DspMsgqObjekt,
        &syncLocateAttrs) ;
        if ((status == DSP_ENOTFOUND) || (status == DSP_ENOTREADY)) // ako ga nije naslo,
        cekaj 100000 uSec
            usleep (100000);
        else if (DSP_FAILED (status)){ printf("Greska kod MSGQ_locate (). Status = [0x%x]\n"
        ,(unsigned int) status); return -1;}
    }

    // alociraj memoriju za MSGQ poruku
    status = MSGQ_alloc (POOL_makePoolId(DSP_ID, POOL_ID),
                         VELICINA_MSG_PORUKE,
                         (MSGQ_Msg *) &msg);
    if (DSP_FAILED (status)){ printf("Greska kod MSGQ_alloc (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    return 0;
}

/* ========================================================================*
*       salji DSP-u
*  ========================================================================*/
int DSPLink_SendToDSP (t_pix *img, GaussParam * gauss)
{
    DSP_STATUS     status   = DSP_SOK;
    int             x,y;
    unsigned int sizeX = gauss->block_size;
    unsigned int sizeY = gauss->block_size;
    unsigned int offsetX = gauss->offsetX;
    unsigned int offsetY = gauss->offsetY;

    // puni spremnik
    for( y = 0; y < sizeY; y++) {
        for( x = 0; x < sizeX; x++) {
            (msg->img)[x + y*sizeX] = img[offsetX + x + (offsetY + y)*IMAGE_WIDTH];
        }
    }
    (msg->gauss).offsetX = gauss->offsetX;
    (msg->gauss).offsetY = gauss->offsetY;
    (msg->gauss).block_size = gauss->block_size;
```



```c
    // salji DSP-u
    do {
        status = MSGQ_put (DspMsgqObjekt, (MSGQ_Msg) msg); // šalji poruku
        if (DSP_FAILED (status)){ printf("Greska kod MSGQ_put (). Status: [0x%x]\n",(unsigned
         int) status); }
    } while (DSP_FAILED (status));

    return 0;
}

/* ========================================================================*
*       citaj sa DSP-a
*  ========================================================================*/
int DSPLink_ReadFromDSP (t_pix *img, GaussParam *gauss)
{
    DSP_STATUS     status   = DSP_SOK;
    Uint32          i;

    // cekaj dok DSP ne posalje poruku i citaj podatke
    status = MSGQ_get (GppMsgqObjekt, WAIT_NONE, (MSGQ_Msg *) &msg);
    if (DSP_SUCCEEDED (status)) {
        // puni ulazni spremnik
        gauss->Xmean = (msg->gauss).Xmean;
        gauss->Ymean = (msg->gauss).Ymean;
        gauss->V = (msg->gauss).V;
        gauss->A = (msg->gauss).A;
        gauss->sigma_x = (msg->gauss).sigma_x;
        gauss->sigma_y = (msg->gauss).sigma_y;
        gauss->sigma_omjer = (msg->gauss).sigma_omjer;
        gauss->phi = (msg->gauss).phi;
    } else {
        // DSP jos nije stigao obraditi
        return -1;
    }
    return 0;
}


/* ========================================================================*
*       zatvaranje PROC, POOL i MSGQ modula
*  ========================================================================*/
int DSPLink_DeinitModules()
{
    DSP_STATUS status    = DSP_SOK;

    // oslobodi memoriju alociranu za MSGQ poruku
    status = MSGQ_free ((MSGQ_Msg) msg);

    if (DSP_FAILED (status)){ printf("Greska kod MSGQ_free (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}
    // oslobodi MSGQ DSP objekt
    status = MSGQ_release (DspMsgqObjekt);
    if (DSP_FAILED (status)){ printf("Greska kod MSGQ_release (). Status: [0x%x]\n",(unsigned
     int) status); return -1;}

    // onemoguci MSGQ transport poruka
    status = MSGQ_transportClose (DSP_ID);
    if (DSP_FAILED (status)){ printf("Greska kod MSGQ_transportClose (). Status: [0x%x]\n",(
```

```
    unsigned int) status); return -1;}

    // zaustavi DSP procesor
    status = PROC_stop (DSP_ID);
    if (DSP_FAILED (status)){ printf("Greska kod PROC_stop (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // zatvori MSGQ modul
    status = MSGQ_close (GppMsgqObjekt);
    if (DSP_FAILED (status)){ printf("Greska kod MSGQ_close (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // zatvori POOL_modul
    status = POOL_close (POOL_makePoolId(DSP_ID, POOL_ID));
    if (DSP_FAILED (status)){ printf("Greska kod POOL_close (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // odijeliti DSP
    status = PROC_detach (DSP_ID);
    if (DSP_FAILED (status)){ printf("Greska kod PROC_detach (). Status: [0x%x]\n",(unsigned
    int) status); return -1;}

    // zatvori PROC_modul
    status = PROC_destroy();
    if (DSP_FAILED (status)){ printf("Greska kod PROC_destroy (). Status: [0x%x]\n",(unsigned
     int) status); return -1;}

    return 0;
}
```

```
/*****************************************************************************
*       Cam.c
*****************************************************************************/

/*  ---------------------------------- OS Specific Headers                */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include "libv4l2.h"

/*  ---------------------------------- APP HEADER                         */
#include "BB_Init.h"

#define CLEAR(x) memset(&(x), 0, sizeof(x))

struct v4l2_format              fmt;
unsigned int                    n_buffers;
struct buffer                   *buffers = NULL;
int                             fd = -1;

struct buffer {
        void   *start;
        size_t length;
};

int Cam_Ioctl(int fh, int request, void *arg)
{
        int                         r;
        do {
                r = v4l2_ioctl(fh, request, arg);
        } while (r == -1 && ((errno == EINTR) || (errno == EAGAIN)));

        if (r == -1) {
                fprintf(stderr, "error %d, %s\n", errno, strerror(errno));
                return r;
        }
        return 0;
}

int Cam_Open(char *dev_name)
{
        // NONBLOCK znači da ne blokira nego javi error
        fd = v4l2_open(dev_name, O_RDWR | O_NONBLOCK, 0);
        //fd = v4l2_open(dev_name, O_RDWR, 0);
        if (fd < 0) {
                printf("Greska kod v4l2_open ()\n"); return -1;
        }
        return 0;
}

int Cam_Close()
```



```
{
        int                         r;
        r = v4l2_close(fd);
        if(r == -1) { printf("Greska kod v4l2_close ()\n"); return -1;}
        return 0;
}

int Cam_Start()
{
        unsigned int                i;
        int                         r;
        struct v4l2_requestbuffers  req;
        enum v4l2_buf_type          type;
        struct v4l2_buffer          buf;

        // inicijaliziraj buffere u device memory
        CLEAR(req);
        req.count = 2;
        req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_MMAP;
        r = Cam_Ioctl(fd, VIDIOC_REQBUFS, &req);
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

        if (buffers == NULL) {
                buffers = calloc(req.count, sizeof(*buffers));
        }
        for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
                CLEAR(buf);

                buf.type        = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory      = V4L2_MEMORY_MMAP;
                buf.index       = n_buffers;

                r = Cam_Ioctl(fd, VIDIOC_QUERYBUF, &buf);
                if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

                buffers[n_buffers].length = buf.length;
                buffers[n_buffers].start = v4l2_mmap(NULL, buf.length,
                              PROT_READ | PROT_WRITE, MAP_SHARED,
                              fd, buf.m.offset);

                if (MAP_FAILED == buffers[n_buffers].start) {
                        printf("Greska kod v4l2_mmap ()\n"); return -1;
                }
        }

        // postavi nove buffere u red
        for (i = 0; i < n_buffers; ++i) {
                CLEAR(buf);
                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;
                buf.index = i;
                r = Cam_Ioctl(fd, VIDIOC_QBUF, &buf);
                if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}
        }
        // enable output hardware and buffers
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        r = Cam_Ioctl(fd, VIDIOC_STREAMON, &type);
```

```
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}
        return 0;
}

int Cam_Stop()
{
        unsigned int                    i;
        int                             r;
        enum v4l2_buf_type              type;
        // disable output hardware and buffers
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        r = Cam_Ioctl(fd, VIDIOC_STREAMOFF, &type);
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}
        // unmap buffers
        for (i = 0; i < n_buffers; ++i) {
                if (v4l2_munmap(buffers[i].start, buffers[i].length) == -1) {
                        printf("Greska kod v4l2_munmap ()\n");
                }
        }
        return 0;
}

int Cam_ImgFormat(unsigned int width, unsigned int height)
{
        int                             r;
        CLEAR(fmt);
        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        fmt.fmt.pix.width       = width;
        fmt.fmt.pix.height      = height;
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24; // RGB
        fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
        r = Cam_Ioctl(fd, VIDIOC_S_FMT, &fmt);
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

        if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24) {
                printf("Libv4l didn't accept RGB24 format. Can't proceed.\n"); return -1;
        }
        printf("Driver is sending image at %dx%d\n",
               fmt.fmt.pix.width, fmt.fmt.pix.height);
        return 0;
}

int Cam_SetExp(unsigned int exposure)
{
        int                             r;
        struct v4l2_queryctrl           queryctrl;
        struct v4l2_control             control;
        // AUTO_EXPOSURE
        memset (&queryctrl, 0, sizeof (queryctrl));
        queryctrl.id = V4L2_CID_EXPOSURE_AUTO;
        r = Cam_Ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl);
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

        if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
                perror("V4L2_CID_EXPOSURE_AUTO is permanently disabled!\n");
                printf("Can't disable auto-exposure! Auto-exposure enabled!\n");
        } else {
                // disable AUTO_EXPOSURE
```



```
                memset (&control, 0, sizeof (control));
                control.id = V4L2_CID_EXPOSURE_AUTO;
                control.value = V4L2_EXPOSURE_MANUAL;
                r = Cam_Ioctl(fd, VIDIOC_S_CTRL, &control);
                if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}
                // read status AUTO_EXPOSURE
                r = Cam_Ioctl(fd, VIDIOC_G_CTRL, &control);
                if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

                if (control.value == V4L2_EXPOSURE_AUTO) {
                        printf("Auto-exposure enabled!\n");
                } else {
                        printf("Auto-exposure disabled!\n");

                        // test if changing exposure is supported
                        queryctrl.id = V4L2_CID_EXPOSURE_ABSOLUTE;
                        r = Cam_Ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl);
                        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

                        if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
                                perror ("V4L2_CID_EXPOSURE_ABSOLUTE is permanently
                                disabled!\n");
                                printf ("Exposure cannot be changed!\n");
                        } else {
                                /*
                                printf ("VALID EXPOSURE VALUES:\n"
                                        "     min = %d\n"
                                        "     max = %d\n"
                                        "     step = %d\n"
                                        "     default = %d\n",
                                       queryctrl.minimum, queryctrl.maximum,
                                       queryctrl.step, queryctrl.default_value );
                                */
                                // set ABSOLUTE_EXPOSURE
                                memset (&control, 0, sizeof (control));
                                control.id = V4L2_CID_EXPOSURE_ABSOLUTE;

                                if (exposure > queryctrl.maximum) control.value = queryctrl.
                                maximum;
                                else if (exposure < queryctrl.minimum) control.value =
                                queryctrl.minimum;
                                else control.value = exposure;
                                r = Cam_Ioctl(fd, VIDIOC_S_CTRL, &control);
                                if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

                                // read status ABSOLUTE_EXPOSURE
                                r = Cam_Ioctl(fd, VIDIOC_G_CTRL, &control);
                                if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}
                                printf("Exposure is set to %d\n", control.value);
                        }
                }
        }
        return 0;
}

int Cam_Init(char *dev_name, unsigned int width, unsigned int height, unsigned int exposure)
{
        int                             r;
```

```
        // open video device
        r = Cam_Open(dev_name);
        if(r == -1) { printf("Greska kod Cam_Open ()\n"); return -1;}
        // set exposure
        r = Cam_SetExp(exposure);
        if(r == -1) { printf("Greska kod Cam_SetExp ()\n"); return -1;}
        // set image format
        r = Cam_ImgFormat(width, height);
        if(r == -1) { printf("Greska kod Cam_ImgFormat ()\n"); return -1;}
        // start STREAM
        r = Cam_Start();
        if(r == -1) { printf("Greska kod Cam_Start ()\n"); return -1;}

        return 0;
}

int Cam_Deinit()
{
        int                     r;
        r = Cam_Stop();
        if(r == -1) { printf("Greska kod Cam_Stop ()\n"); return -1;}
        r = Cam_Close();
        if(r == -1) { printf("Greska kod Cam_Close ()\n"); return -1;}

        return 0;
}

int Cam_Frame(t_pix *R, t_pix *G, t_pix *B, t_pix *gray, double Rcoef, double Gcoef, double
Bcoef)
{
        int                     r;
        struct v4l2_buffer      buf;
        unsigned int            pix_cnt, x, y;
        char                    *p_pix;

        // read ready buffer
        CLEAR(buf);
        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_MMAP;
        r = Cam_Ioctl(fd, VIDIOC_DQBUF, &buf);
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}

        // copy data from video device
        for(pix_cnt = 0 ; pix_cnt < fmt.fmt.pix.width*fmt.fmt.pix.height; pix_cnt++) {
                p_pix = (char*)buffers[buf.index].start;
#ifdef  IMG_TRANSPOSE   // frame(HEIGHT, WIDTH)
                x = pix_cnt % fmt.fmt.pix.width;
                y = (int)(pix_cnt / fmt.fmt.pix.width);
                if (gray != NULL)
                    // convert rgb (uint8) to grayscale
                    gray[x*fmt.fmt.pix.height + y] = (t_pix)(Rcoef*p_pix[3*pix_cnt]+Gcoef*
                    p_pix[3*pix_cnt+1]+Bcoef*p_pix[3*pix_cnt+2]);
                if ((R != NULL)&&(G != NULL)&&(B != NULL)) {
                    R[x*fmt.fmt.pix.height + y] = (t_pix)(p_pix[3*pix_cnt]);
                    G[x*fmt.fmt.pix.height + y] = (t_pix)(p_pix[3*pix_cnt + 1]);
                    B[x*fmt.fmt.pix.height + y] = (t_pix)(p_pix[3*pix_cnt + 2]);
                }
#else   // frame(WIDTH, HEIGHT)
```



```
                if (gray != NULL)
                    // convert rgb (uint8) to grayscale
                    gray[pix_cnt] = (t_pix)(Rcoef*p_pix[3*pix_cnt]+Gcoef*p_pix[3*pix_cnt+1]+
                    Bcoef*p_pix[3*pix_cnt+2]);
                if ((R != NULL)&&(G != NULL)&&(B != NULL)) {
                    R[pix_cnt] = (t_pix)p_pix[3*pix_cnt];
                    G[pix_cnt] = (t_pix)p_pix[3*pix_cnt + 1];
                    B[pix_cnt] = (t_pix)p_pix[3*pix_cnt + 2];
                }
#endif /* IMG_TRANSPOSE */
        }

        // postavi buffer nazad u red
        r = Cam_Ioctl(fd, VIDIOC_QBUF, &buf);
        if(r == -1) { printf("Greska kod Cam_Ioctl ()\n"); return -1;}
        return 0;
}
```

```c
/*****************************************************************************
*       Disp.c
*****************************************************************************/

/*  ---------------------------------- OS Specific Headers              */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "../SDL/include/SDL.h"

#define SDL_lib
#define SDL_FpsToPeriod(FPS)    1/FPS

// Display parameters
SDL_Surface     *screen;
Uint32          time = 0;
Uint32          Disp_Width, Disp_Height, Disp_Bpp;
//
Uint32          SDL_fps = 25;

int Disp_PutPixel(SDL_Surface *surface, Sint16 pos_x, Sint16 pos_y, Uint32 color)
{
    Uint32 bpp = surface->format->BytesPerPixel;
    // Here p is the address to the pixel we want to set
    Uint8 *p = (Uint8 *)surface->pixels + pos_y * surface->pitch + pos_x * bpp;

    switch(bpp) {
    case 1:
        *p = color;
        break;
    case 2:
        *(Uint16 *)p = color;
        break;
    case 3:
        if(SDL_BYTEORDER == SDL_BIG_ENDIAN) {
            p[0] = (color >> 16) & 0xff;
            p[1] = (color >> 8) & 0xff;
            p[2] = color & 0xff;
        } else {
            p[0] = color & 0xff;
            p[1] = (color >> 8) & 0xff;
            p[2] = (color >> 16) & 0xff;
        }
        break;
    case 4:
        *(Uint32 *)p = color;
        break;
    }
    return 0;
}

// Currently only the quit event
int Disp_EventHandler(double *quit_event)
{
    SDL_Event       event;
    *quit_event = 0;
    // Event handler
```



```c
    while (SDL_PollEvent(&event)) {
        switch(event.type){  /* Process the appropiate event type */
            case SDL_QUIT:  /* Handle a QUIT event */
                *quit_event = 1;
                break;
            default: /* Report an unhandled event */
                //printf("Unknown SDL event!\n");
                break;
          }
    }
    return 0;
}


int Disp_RegFPS(double fps)
{
    double      sample_time;
    // Regulate Sample time
    sample_time = ((double)(SDL_GetTicks() - time))/1000;
    if (sample_time < SDL_FpsToPeriod(fps)) {
        // Pause execution
        SDL_Delay( (Uint32)((SDL_FpsToPeriod(fps) - sample_time)*1000) );
    } else {
        printf("Sample_time = %.3f \n", sample_time);
        time = SDL_GetTicks();
        //return -1;
        return 0;
    }
    time = SDL_GetTicks();
    return 0;
}

int Disp_Frame(t_pix *R, t_pix *G, t_pix *B, t_pix *gray)
{
    int             r;
    Uint32          pix_cnt, x, y;
#ifdef USE_SDL
    // Lock the screen for direct access to the pixels
    if ( SDL_MUSTLOCK(screen) ) {
        r = SDL_LockSurface(screen);
        if (r == -1) { printf("Greska kod SDL_LockSurface ()\n"); return -1;}
    }
    for (pix_cnt = 0; pix_cnt < Disp_Width*Disp_Height; pix_cnt++) {
        // Draw pixel on screen
#ifdef  IMG_TRANSPOSE   // frame(HEIGHT, WIDTH)
        x = pix_cnt/Disp_Height;
        y = pix_cnt%Disp_Height;
#else   // frame(WIDTH, HEIGHT)
        x = pix_cnt%Disp_Width;
        y = pix_cnt/Disp_Width;
#endif /* IMG_TRANSPOSE */
        if((R != NULL)&&(G != NULL)&&(B != NULL)) {
            r = Disp_PutPixel(screen, x, y,
                        SDL_MapRGB(screen->format, (Uint8)R[pix_cnt], (Uint8)G[pix_cnt], (
                        Uint8)B[pix_cnt]));
            if (r == -1) { printf("Greska kod Disp_PutPixel ()\n"); return -1;}
        } else if (gray != NULL) {
            r = Disp_PutPixel(screen, x, y,
```

```
                        SDL_MapRGB(screen->format, (Uint8)gray[pix_cnt], (Uint8)gray[pix_cnt
                        ], (Uint8)gray[pix_cnt]));
            if (r == -1) { printf("Greska kod Disp_PutPixel ()\n"); return -1;}
        }
    }
    // Unlock the screen
    if ( SDL_MUSTLOCK(screen) ) { SDL_UnlockSurface(screen);}

    // Swaps screen buffers
    r = SDL_Flip(screen);
    if (r == -1) { printf("Greska kod SDL_Flip ()\n"); return -1;}
#endif /* USE_SDL */
    return 0;
}

int Disp_Init(Uint32 width, Uint32 height, Uint32 bpp)
{
    int             r;
    // Initialize global variables
    Disp_Width  = width;
    Disp_Height = height;
    Disp_Bpp    = bpp;
    // Initializes SDL
    r = SDL_Init(SDL_INIT_VIDEO);
    if (r == -1) { printf("Greska kod SDL_Init ()\n"); return -1;}
#ifdef USE_SDL
    // Set up a video mode with the specified width, height and bits-per-pixel
    screen = SDL_SetVideoMode(Disp_Width, Disp_Height, Disp_Bpp, SDL_SWSURFACE);
    if (screen == NULL) { printf("Greska kod SDL_SetVideoMode ()\n"); return -1;}
#endif /* USE_SDL */
    return 0;
}

int Disp_Deinit()
{
    // Free surface
    SDL_FreeSurface(screen);
    // Shut down SDL
    SDL_Quit();
    return 0;
}
```

```c
/*****************************************************************************
*       Img.c
*****************************************************************************/

/*  ---------------------------------- OS Specific Headers             */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

#include "test_02_0.h"
#include "test_02_1.h"
#include "test_02_2.h"

/*  ---------------------------------- APP HEADER                      */
// sortirani pikseli slike
SortedPix   img_sort[IMAGE_WIDTH*IMAGE_HEIGHT];
// parametri svake zvijezde
GaussParam  starNew[MAX_STAR_NUM];
// parametri svake zvijezde
GaussParam  starOld[MAX_STAR_NUM];
int     starOld_cnt = 0;
// binarna slika slika
t_pix   img_buff[IMAGE_WIDTH*IMAGE_HEIGHT] = {0};

// DEBUG
int pic = 0;

/*
-------------------------------------------------------------------------------------------------
-----------------------------------------------  */
#define     pi      3.14159265358979323846

double Img_Moment(t_pix *img, unsigned int sizeX, unsigned int sizeY, int offsetX, int
offsetY, double Xmean, double Ymean, double orderX, double orderY)
{
    double      r = 0;
    int         x,y;
    for( y = 0; y < sizeY; y++) {
        for( x = 0; x < sizeX; x++) {
            r += (pow((x-Xmean),orderX) * pow((y-Ymean),orderY) * (double)(img[offsetX + x +
            (offsetY + y)*IMAGE_WIDTH]));
        }
    }
    return r;
}

int Img_GaussParam(t_pix *img, GaussParam * gauss)
{
    double      m11, m20, m02, m20phi, m02phi, phi, ra, su;
    unsigned int sizeX = gauss->block_size;
    unsigned int sizeY = gauss->block_size;
    int offsetX = gauss->offsetX;
    int offsetY = gauss->offsetY;

    gauss->V = Img_Moment(img, sizeX, sizeY, offsetX, offsetY,0,0,0,0);
    gauss->Xmean = Img_Moment(img, sizeX, sizeY, offsetX, offsetY,0,0,1,0)/(gauss->V);
    gauss->Ymean = Img_Moment(img, sizeX, sizeY, offsetX, offsetY,0,0,0,1)/(gauss->V);
```



```c
    m11 = Img_Moment(img, sizeX, sizeY, offsetX, offsetY, gauss->Xmean, gauss->Ymean, 1, 1)/(
    gauss->V);
    m20 = Img_Moment(img, sizeX, sizeY, offsetX, offsetY, gauss->Xmean, gauss->Ymean, 2, 0)/(
    gauss->V);
    m02 = Img_Moment(img, sizeX, sizeY, offsetX, offsetY, gauss->Xmean, gauss->Ymean, 0, 2)/(
    gauss->V);

    phi = atan2((2*m11),(m20-m02))/2;

    if ((m11==0)&&(m20-m02==0)) {
        phi = 0;
    } else {
        phi = -180*phi/pi;
    }

    gauss->phi = phi;
    // prvi naÃ¯in
    //m20phi = m20*pow(cos(phi),2) + m02*pow(sin(phi),2) + m11*sin(2*phi);
    //m02phi = m20*pow(sin(phi),2) + m02*pow(cos(phi),2) - m11*sin(2*phi);
    //gauss->sigma_x = sqrt(m20phi);
    //gauss->sigma_y = sqrt(m02phi);

    // Drugi naÃ¯in
    ra = sqrt( pow((m20-m02),2) + 4*pow(m11,2) );
    su = m20+m02;
    gauss->sigma_x = sqrt((su+ra)/2);
    gauss->sigma_y = sqrt((su-ra)/2);

    if ((gauss->sigma_x == 0)||(gauss->sigma_y == 0)) {
        gauss->A = 0;
    } else {
        gauss->A = (gauss->V)/(2*pi*(gauss->sigma_x)*(gauss->sigma_y));
    }

    gauss->Xmean = gauss->Xmean + gauss->offsetX;
    gauss->Ymean = gauss->Ymean + gauss->offsetY;

    return 0;
}

int Img_Cmpfunc(const void * a, const void * b)
{
    if ( ((SortedPix*)b)->value > ((SortedPix*)a)->value ) return 1;
    if ( ((SortedPix*)b)->value < ((SortedPix*)a)->value ) return -1;
    if ( ((SortedPix*)b)->value == ((SortedPix*)a)->value ) return 0;
}

int Img_IsolateStar(t_pix *img, unsigned int posX, unsigned int posY, int *offsetX, int *
offsetY, unsigned int *block_size, t_pix pix_offset)
{
    int     r = 0;
    int     i;
    int     clear_cnt = 0;
    int     dir = 0; // 0 pozitivan smjer, 1 negativan smjer
    int     not_clear = 0;
    int     step = 1;
    int     pos_x = posX;
```

```
    int     pos_y = posY;

    while(1) {
        // x
        for(i = 0; i < step; i++) {
            if(!dir) {
                pos_x++;
            } else {
                pos_x--;
            }
            if(((pos_y < IMAGE_HEIGHT)&&(pos_y >= 0))&&((pos_x < IMAGE_WIDTH)&&(pos_x >= 0)))
             {
                if (img[pos_x + pos_y*IMAGE_WIDTH] > pix_offset) {
                    not_clear = 1;
                }
            } else {
                r = -1;
            }
        }

        if(!not_clear) {
            clear_cnt++;
            if((clear_cnt == 4)||(r == -1)) {
                if(!dir) {
                    *offsetX = pos_x - step;
                    *offsetY = pos_y;
                } else {
                    *offsetX = pos_x; // probati dodati +1
                    *offsetY = pos_y - step + 1;
                }
                break;
            }
        } else {
            not_clear= 0;
            clear_cnt = 0;
        }

        // y
        for(i = 0; i < step; i++) {
            if(!dir) {
                pos_y++;
            } else {
                pos_y--;
            }
            if(((pos_y < IMAGE_HEIGHT)&&(pos_y >= 0))&&((pos_x < IMAGE_WIDTH)&&(pos_x >= 0)))
             {
                if (img[pos_x + pos_y*IMAGE_WIDTH] > pix_offset) {
                    not_clear = 1;
                }
            } else {
                r = -1;
            }
        }

        if(!not_clear) {
            clear_cnt++;
            if((clear_cnt == 4)||(r == -1)) {
                if(!dir) {
```



```
                    *offsetX = pos_x - step;
                    *offsetY = pos_y - step;
                } else {
                    *offsetX = pos_x;
                    *offsetY = pos_y;
                }
                break;
            }
        } else {
            not_clear= 0;
            clear_cnt = 0;
        }

        // uveÄaj step i promijeni smjer
        step++;
        if(!dir) {
            dir = 1;
        } else {
            dir = 0;
        }
    }

    if (*offsetX < 0) *offsetX = 0;
    if (*offsetY < 0) *offsetY = 0;

    *block_size = step;
    if (r == -1) {
        if (*block_size + *offsetX > IMAGE_WIDTH) {
            *block_size = IMAGE_WIDTH - *offsetX;
        }
        if (*block_size + *offsetY > IMAGE_HEIGHT) {
            *block_size = IMAGE_HEIGHT - *offsetY;
        }
    }
    return r;
}

int Img_ClearArea(t_pix *img, int offsetX, int offsetY, unsigned int sizeX, unsigned int
sizeY, t_pix pix_offset)
{
    int     x, y, pos_x, pos_y;
    if(img == NULL) return -1;
    if(offsetX >= IMAGE_WIDTH) return -1;
    if(offsetY >= IMAGE_HEIGHT) return -1;
    if((double)pix_offset < 0) return -1;
    // obrisi dio slike sa zvijezdom
    for(x = 0; x < sizeX; x++) {
        for(y = 0; y < sizeY; y++) {
            pos_x = offsetX + x;
            pos_y = offsetY + y;
            if(pix_offset == 0) // brisi sve piksele
            {
                if(img[pos_x + pos_y*IMAGE_WIDTH] != 0) {
                    img[pos_x + pos_y*IMAGE_WIDTH] = 0;
                }
            } else { // brisi piksele ispod offseta
                if(img[pos_x + pos_y*IMAGE_WIDTH] <= pix_offset) {
                    img[pos_x + pos_y*IMAGE_WIDTH] = 0;
```

```
                }
            }
        }
    }
    return 0;
}

int Img_SortPix(t_pix *img, SortedPix *img_sort)
{
    int     x, y;
    if(img == NULL) return -1;
    if(img_sort == NULL) return -1;
    // pripremi piksele za sortiranje
    for(x = 0; x < IMAGE_WIDTH; x++) {
        for(y = 0; y < IMAGE_HEIGHT; y++) {
            img_sort[x + y*IMAGE_WIDTH].value = img[x + y*IMAGE_WIDTH];
            img_sort[x + y*IMAGE_WIDTH].posX = (unsigned int)x;
            img_sort[x + y*IMAGE_WIDTH].posY = (unsigned int)y;
        }
    }
    // sortiraj
    qsort(img_sort, IMAGE_WIDTH*IMAGE_HEIGHT, sizeof(SortedPix), Img_Cmpfunc);
    return 0;
}

int Img_GetMoveDist(GaussParam  *starCurent, GaussParam *starArray, int starArray_cnt)
{
    int     i;
    double  Xdist, Ydist, phi1, phi2;
    double  ExpectedDist;
    double  DistStarCurrent;
    double  DistStarBestFit = MAX_MOVE_DIST;
    if(starCurent == NULL) return -1;
    if(starArray == NULL) return -1;
    ExpectedDist = 5*(starCurent->sigma_x) + 5*(starCurent->sigma_y);
    // nadji najblizu zvijezdu na staroj slici
    for(i = 0; i < starArray_cnt; i++) {
        Xdist = starCurent->Xmean - starArray[i].Xmean;
        Ydist = starCurent->Ymean - starArray[i].Ymean;
        // kut se mora poklapati
        phi1 = atan2(Ydist, Xdist);
        if ((Ydist==0)&&(Xdist==0)) {
            phi1 = 0;
            phi2 = 0;
        } else {
            phi1 = -180*phi1/pi;
            phi2 = phi1 - 180;
            if (phi2 < -180)    phi2 += 360; // da kut bude unutar [-180Â° , 180Â°]
        }
        // ako je razlika manja od MAX_ANG_DIFF, onda je moguÃ¡e da je to zvijezda koja se
        traÅ¾i
        if ((MAX_ANG_DIFF > abs(phi1 - starCurent->phi))||
            (MAX_ANG_DIFF > abs(phi2 - starCurent->phi))) {
            // naÄ'i udaljenost
            DistStarCurrent = sqrt(pow(Xdist,2)+pow(Ydist,2));
            // provjeri udaljenost
            if(abs(ExpectedDist-DistStarCurrent) < abs(ExpectedDist-DistStarBestFit)) {
                DistStarBestFit = DistStarCurrent;
```



```
                // spremi smjer pomaka
                // starCurent->movePhi = (MAX_ANG_DIFF > abs(phi1 - starCurent->phi)) ? phi1
                : phi2; // prema kutu gaussiana
                starCurent->movePhi = phi1; // pravi smjer kretanja
            }
        }
    }
    // ako je pomak blizak pretpostavljenom pomaku
    if(abs(ExpectedDist-DistStarBestFit)/ExpectedDist < MAX_DIST_TOL) {
        // spremi iznos pomaka
        starCurent->moveDist = DistStarBestFit;
        // korigiraj kut gaussiana (time se nista ne gubi, samo se po potrebi zarotira za
        180Â°)
        starCurent->phi = (MAX_ANG_DIFF < abs(starCurent->movePhi - starCurent->phi)) ?
        starCurent->phi-180 : starCurent->phi;
        if (starCurent->phi < -180) starCurent->phi += 360;
        return 0;
    }
    return -1;
}

int Img_Proc(t_pix *img, double percent, double star_num, double *dist_mean, double *ang_mean)
{
    int     r;
    int     i;
    int     posX, posY, offsetX, offsetY, block_size;
    int     new_star = 0;
    int     starNew_cnt = 0;
    int     nove_zvijezde = 0;
    int     prepoznate_zvijezde = 0;
    int     proc_cnt = 0;
    t_pix   pix_offset;

    *dist_mean = 0;
    *ang_mean = 0;

    // DEBUG
    if (pic == 0) {
        img = &test_0[0];
        pic++;
    } else if (pic == 1) {
        img = &test_1[0];
        pic++;
    } else if (pic == 2) {
        img = &test_2[0];
        pic++;
    } else return 0;

    printf("<---------- Slika %d. ---------->\n", pic);

    // dalje radi sa img_buff, img na kraju obradi
    memcpy ( &img_buff[0], img, sizeof(t_pix)*IMAGE_WIDTH*IMAGE_HEIGHT );

    // MAX_STAR_NUM zvijezdi max
    if (star_num > (double)MAX_STAR_NUM) star_num = (double)MAX_STAR_NUM;

    // sortiraj piksele
    r = Img_SortPix(img_buff, img_sort);
```

```
    if(r == -1) { printf("Greska kod Img_SortPix ()\n"); return -1;}

    // naÄ'i offset
    if(percent > 1) percent = 1;
    else if(percent < 0) percent = 0;
    pix_offset = img_sort[(int)floor(percent*IMAGE_WIDTH*IMAGE_HEIGHT)].value;
    //pix_offset = 26;

    while((img_sort[0].value > pix_offset)&&(starNew_cnt < star_num)) {
        // izdvoji zvijezdu
        r = Img_IsolateStar(img_buff, img_sort[0].posX, img_sort[0].posY, &offsetX, &offsetY,
         &block_size, pix_offset);
        if((r == 0)&&(block_size < MAX_STAR_BLOCK_SIZE)) {
            //printf("NIJE RUBNA!!**\n");
            // obrisi sve pixele ispod pix_offset na dijelu slike sa zvijezdom
            r = Img_ClearArea(img_buff, offsetX, offsetY, block_size, block_size, pix_offset);
            if(r == -1) { printf("Greska kod Img_ClearArea ()\n"); return -1;}

            /* ============================================================================*
            *       raÄ'unaj parametre izdvojene zvijezde
            *  ============================================================================*/
            starNew[starNew_cnt].offsetX = offsetX;
            starNew[starNew_cnt].offsetY = offsetY;
            starNew[starNew_cnt].block_size = block_size;
            posX = img_sort[0].posX;
            posY = img_sort[0].posY;

#if defined USE_DSP
            proc_cnt++;
            r = DSPLink_SendToDSP(img_buff, &starNew[starNew_cnt]);
            if(r == -1) { printf("Greska kod DSPLink_SendToDSP ()\n"); return -1;}
#else
            r = Img_GaussParam(img_buff, &starNew[starNew_cnt]);
            if(r == -1) { printf("Greska kod Img_GaussParam ()\n"); return -1;}
#endif /* USE_DSP */

            while(new_star) { // racunaj za prethodno nadjenu zvijezdu
                    // odredi smjer i iznos pomaka
                    r = Img_GetMoveDist(&starNew[starNew_cnt-new_star], &starOld[0],
                    starOld_cnt);
                    if(r == -1) {
                        starNew[starNew_cnt-new_star].moveDist = -1;
                        nove_zvijezde++;
                    } else {
                        prepoznate_zvijezde++;
                        *dist_mean += starNew[starNew_cnt-new_star].moveDist;
                        *ang_mean += starNew[starNew_cnt-new_star].phi;
                    }
                    starNew[starNew_cnt-new_star].sigma_omjer = starNew[starNew_cnt-new_star
                    ].sigma_x/starNew[starNew_cnt-new_star].sigma_y;
                    printf("%d. = %.4f %.4f %.4f %.4f %.4f %.4f %.4f\n",   starNew_cnt-
                    new_star,
                                                                          starNew[
starNew_cnt-new_star].Xmean,
                                                                          starNew[
starNew_cnt-new_star].Ymean,

//starNew[starNew_cnt-new_star].V,
```



```
//starNew[starNew_cnt-new_star].A,
                                                                          starNew[
starNew_cnt-new_star].sigma_x,
                                                                          starNew[
starNew_cnt-new_star].sigma_y,
                                                                          starNew[
starNew_cnt-new_star].phi,
                                                                          starNew[
starNew_cnt-new_star].movePhi,
                                                                          starNew[
starNew_cnt-new_star].moveDist,
                                                                          5*(starNew[
starNew_cnt-new_star].sigma_x) - 5*(starNew[starNew_cnt-new_star].sigma_y));
                    //printf("%d. = %.4f %.4f %.4f %d %d %d %d %d\n", starNew_cnt-new_star,
                    starNew[starNew_cnt-new_star].Xmean,
                    starNew[starNew_cnt-new_star].Ymean, starNew[starNew_cnt-new_star].V,
                    posX, posY, starNew[starNew_cnt-new_star].offsetX,
                    starNew[starNew_cnt-new_star].offsetY,
                    starNew[starNew_cnt-new_star].block_size);
                    new_star--;
            }

            // obrisi sve piksele na dijelu slike sa zvijezdom
            r = Img_ClearArea(img_buff, offsetX, offsetY, block_size, block_size, 0);
            if(r == -1) { printf("Greska kod Img_ClearArea ()\n"); return -1;}
            // sortiraj piksele
            r = Img_SortPix(img_buff, img_sort);
            if(r == -1) { printf("Greska kod Img_SortPix ()\n"); return -1;}
            //printf("img_sort[0].value=%d > %d ; starNew_cnt=%d < %f; new_star=%d\n",
            img_sort[0].value, pix_offset, starNew_cnt, star_num, new_star);

    #ifndef USE_DSP
            if ((starNew[starNew_cnt].A!=0)&&(starNew[starNew_cnt].V > 200)) {
                //printf("%d. = %.4f %.4f %.4f %d %d %d %d %d %d\n", starNew_cnt,
                starNew[starNew_cnt].Xmean, starNew[starNew_cnt].Ymean,
                starNew[starNew_cnt].V, posX, posY, starNew[starNew_cnt].offsetX,
                starNew[starNew_cnt].offsetY, starNew[starNew_cnt].block_size, pix_offset);
                starNew_cnt++;
                new_star++;
                //printf("img_sort[0].value=%d > %d ; starNew_cnt=%d < %f; new_star=%d\n",
                img_sort[0].value, pix_offset, starNew_cnt, star_num, new_star);
            }
    #endif /* USE_DSP */
        } else {
            //printf("**RUBNA!!**\n");
            // obrisi sve piksele na dijelu slike sa zvijezdom
            r = Img_ClearArea(img_buff, offsetX, offsetY, block_size, block_size, 0);
            if(r == -1) { printf("Greska kod Img_ClearArea ()\n"); return -1;}
            // sortiraj piksele
            r = Img_SortPix(img_buff, img_sort);
            if(r == -1) { printf("Greska kod Img_SortPix ()\n"); return -1;}
            //printf("img_sort[0].value=%d > %d ; starNew_cnt=%d < %f; new_star=%d\n",
            img_sort[0].value, pix_offset, starNew_cnt, star_num, new_star);
        }
    #if defined USE_DSP
        //printf("TEST!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
        r = DSPLink_ReadFromDSP(NULL, &starNew[starNew_cnt]); // ignoriraj prvi parametar
```

```
    if(r != -1) { // obrada gotova
        proc_cnt--;
        //printf("%d. = %.4f %.4f %.4f %.4f %.4f %.4f %.4f\n", starNew_cnt,
        starNew[starNew_cnt].Xmean, starNew[starNew_cnt].Ymean, starNew[starNew_cnt].V,
        starNew[starNew_cnt].A, starNew[starNew_cnt].sigma_x,
        starNew[starNew_cnt].sigma_y, starNew[starNew_cnt].phi);
        if ((starNew[starNew_cnt].A!=0)&&(starNew[starNew_cnt].V > 200)) {
            //printf("%d. = %.4f %.4f %.4f %d %d %d %d %d %d\n", starNew_cnt,
            starNew[starNew_cnt].Xmean, starNew[starNew_cnt].Ymean,
            starNew[starNew_cnt].V, posX, posY, starNew[starNew_cnt].offsetX,
            starNew[starNew_cnt].offsetY, starNew[starNew_cnt].block_size, pix_offset);
            starNew_cnt++;
            new_star++;
            //printf("img_sort[0].value=%d > %d ; starNew_cnt=%d < %f; new_star=%d\n",
            img_sort[0].value, pix_offset, starNew_cnt, star_num, new_star);
        }
    }
#endif /* USE_DSP */
}
// u slučaju da su ostale obradjene zvijezde koje nisu uzete u obzir
do {
    //printf("VANI!!img_sort[0].value=%d > %d ; starNew_cnt=%d < %f; new_star=%d\n",
    img_sort[0].value, pix_offset, starNew_cnt, star_num, new_star);
    // racunaj jos za zadnju nadjenu zvijezdu, ako je ima
    while(new_star) {
        // odredi smjer i iznos pomaka
        r = Img_GetMoveDist(&starNew[starNew_cnt-new_star], &starOld[0], starOld_cnt);
        if(r == -1) {
            starNew[starNew_cnt-new_star].moveDist = -1;
            nove_zvijezde++;
        } else {
            prepoznate_zvijezde++;
            *dist_mean += starNew[starNew_cnt-new_star].moveDist;
            *ang_mean += starNew[starNew_cnt-new_star].phi;
        }
        starNew[starNew_cnt-new_star].sigma_omjer = starNew[starNew_cnt-new_star].sigma_x
        /starNew[starNew_cnt-new_star].sigma_y;
        printf("%d. = %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f\n",   starNew_cnt-new_star,
                                                                    starNew[starNew_cnt-
                                                                    new_star].Xmean,
                                                                    starNew[starNew_cnt-
                                                                    new_star].Ymean,

                                                                    //starNew[starNew_cnt-
                                                                    new_star].V,

                                                                    //starNew[starNew_cnt-
                                                                    new_star].A,
                                                                    starNew[starNew_cnt-
                                                                    new_star].sigma_x,
                                                                    starNew[starNew_cnt-
                                                                    new_star].sigma_y,
                                                                    starNew[starNew_cnt-
                                                                    new_star].phi,
                                                                    starNew[starNew_cnt-
                                                                    new_star].movePhi,
                                                                    starNew[starNew_cnt-
                                                                    new_star].moveDist,
```



```
                                                                    5*(starNew[
                                                                    starNew_cnt-new_star
                                                                    ].sigma_x) - 5*(
                                                                    starNew[starNew_cnt-
                                                                    new_star].sigma_y));
        //printf("%d. = %.4f %.4f %.4f %d %d %d %d %d\n", starNew_cnt-new_star,
        starNew[starNew_cnt-new_star].Xmean, starNew[starNew_cnt-new_star].Ymean,
        starNew[starNew_cnt-new_star].V, posX, posY,
        starNew[starNew_cnt-new_star].offsetX, starNew[starNew_cnt-new_star].offsetY,
        starNew[starNew_cnt-new_star].block_size);
        new_star--;
    }
#if defined USE_DSP
    r = DSPLink_ReadFromDSP(NULL, &starNew[starNew_cnt]); // ignoriraj prvi parametar
    if(r != -1) {
        proc_cnt--;
        //printf("%d. = %.4f %.4f %.4f %.4f %.4f %.4f %.4f\n", starNew_cnt,
        starNew[starNew_cnt].Xmean, starNew[starNew_cnt].Ymean, starNew[starNew_cnt].V,
        starNew[starNew_cnt].A, starNew[starNew_cnt].sigma_x,
        starNew[starNew_cnt].sigma_y, starNew[starNew_cnt].phi);
        if ((starNew[starNew_cnt].A!=0)&&(starNew[starNew_cnt].V > 200)) {
            //printf("%d. = %.4f %.4f %.4f %d %d %d %d %d %d\n", starNew_cnt,
            starNew[starNew_cnt].Xmean, starNew[starNew_cnt].Ymean,
            starNew[starNew_cnt].V, posX, posY, starNew[starNew_cnt].offsetX,
            starNew[starNew_cnt].offsetY, starNew[starNew_cnt].block_size, pix_offset);
            starNew_cnt++;
            new_star++;
            //printf("img_sort[0].value=%d > %d ; starNew_cnt=%d < %f; new_star=%d\n",
            img_sort[0].value, pix_offset, starNew_cnt, star_num, new_star);
        }
    }
#endif /* USE_DSP */
} while((new_star)||(proc_cnt));
// Prebaci starNew u starOld
starOld_cnt = starNew_cnt;
memcpy ( &starOld[0], &starNew[0], starNew_cnt*sizeof(GaussParam) );

*dist_mean /= (double)prepoznate_zvijezde;
*ang_mean /= (double)prepoznate_zvijezde;

// obrisi sve pixele ispod pix_offset na cijeloj slici
r = Img_ClearArea(img, 0, 0, IMAGE_WIDTH, IMAGE_HEIGHT, pix_offset);
if(r == -1) { printf("Greska kod Img_BinarizeImg ()\n"); return -1;}
printf("pix_offset=%d\nprepoznate_zvijezde=%d\nnove_zvijezde=%d\npomak=%.4f\nkut=%.4f\n",
 pix_offset, prepoznate_zvijezde, nove_zvijezde, *dist_mean, *ang_mean);
// return starNew_cnt;
return prepoznate_zvijezde;
}
```

```c
/*****************************************************************************
*       tskReadwrite.c (DSP)
*****************************************************************************/

/*  ----------------------------------- DSP/BIOS Headers            */
#include <std.h>
#include <log.h>
#include <swi.h>
#include <sys.h>
#include <sio.h>
#include <tsk.h>
#include <msgq.h>
#include <pool.h>

/*  ----------------------------------- DSP/BIOS LINK Headers       */
#include <failure.h>
#include <dsplink.h>
#include <platform.h>
#include <hal_cache.h>

/*  ----------------------------------- Sample Headers              */
#include "tskReadwrite.h"
#include "readwrite.h"
#include "readwrite_config.h"
#include <stdint.h>
#include <math.h>

/*  ----------------------------------- APP HEADER                      */
#include "StarTracker.h"

#define FILEID  FID_APP_C

// LOG objekt potreban kod LOG_printf
extern LOG_Obj trace ;

// broj iteracija (definirana u main.c)
extern Uint32 BrojIteracija ;

/* ========================================================================*
*       struktura MSGQ poruke
*  ========================================================================*/
typedef struct MSG_Poruka_tag {
    MSGQ_MsgHeader msgHeader;
    Uint8          img[MAX_DSP_IMAGE_SIZE];
    GaussParam     gauss;
} MSG_Poruka ;

/* ========================================================================*
*       inicijalizacija MSGQ modula
*  ========================================================================*/
Int Init_MSGQ (TransferInfo ** infoPtr)
{
    Int                     status    = SYS_OK ;
    MSGQ_Attrs              msgqAttrs = MSGQ_ATTRS ;
    MSGQ_LocateAttrs        syncLocateAttrs ;
    TransferInfo *  info      = NULL ;

    //alociraj TransferInfo strukturu
```



```c
    *infoPtr = MEM_calloc (DSPLINK_SEGID, sizeof (TransferInfo), DSPLINK_BUF_ALIGN);
    if (*infoPtr == NULL)
    {
        status = SYS_EALLOC ;
        LOG_printf (&trace, "Greska kod alociranja TransferInfo strukture. Status = 0x%x\n",
        status); return status;
    }
    info = *infoPtr ;

    // inicijaliziraj SEM objekt
    SEM_new (&(info->notifyDspSemObj), 0) ;

    // inicijaliziraj MSGQ modul
    msgqAttrs.notifyHandle = &(info->notifyDspSemObj) ; // postavi SEM handle
    msgqAttrs.pend         = (MSGQ_Pend) SEM_pendBinary ;
    msgqAttrs.post         = (MSGQ_Post) SEM_post ; // pozovi TASK svaki put kad dodje nova
    MSGQ poruka

    // otvori MSGQ objekt za DSP
    status = MSGQ_open (DSP_MSGQNAME, &info->DspMsgqObjekt, &msgqAttrs) ;
    if (status != SYS_OK){ LOG_printf (&trace, "Greska kod MSGQ_open. Status = 0x%x\n",
    status); return status;}

    // Lociraj otvoreni MSGQ na GPP procesoru
    status = SYS_ENOTFOUND ;
    while ((status == SYS_ENOTFOUND) || (status == SYS_ENODEV))
    {
        syncLocateAttrs.timeout = SYS_FOREVER ;
        status = MSGQ_locate (GPP_MSGQNAME,
        &info->GppMsgqObjekt,
        &syncLocateAttrs) ;
        if ((status == SYS_ENOTFOUND) || (status == SYS_ENODEV)) // ako ga nije naslo, cekaj
        1000 perioda takta
            TSK_sleep (1000) ;
        else if (status != SYS_OK) {LOG_printf (&trace, "Greska kod MSGQ_locate (). Status =
        [0x%x]\n", status); return status;}
    }

    return status ;
}

/* ========================================================================*
*       izvodjenje programa (glavna petlja)
*  ========================================================================*/
Int Izvodjenje (TransferInfo * info)
{
    Int   status  = SYS_OK;
    //Uint32          i, j;
    MSG_Poruka *    msg;

    status = SEM_pend(&(info->notifyDspSemObj), SYS_FOREVER); // stavi TASK na cekanje

    while (status == TRUE)
    {
        // GPP je poslao poruku, citaj poruku
        status = MSGQ_get (info->DspMsgqObjekt, (MSGQ_Msg *) &msg, SYS_FOREVER) ;
        if (status != SYS_OK){ LOG_printf (&trace, "Greska kod MSGQ_locate. Status = 0x%x\n",
         status); return status;}
```

```
#ifdef USE_IQMATH
        Img_GaussParamIQmath(&(msg->img)[0], &(msg->gauss));
#else
        Img_GaussParam(&(msg->img)[0], &(msg->gauss));
#endif

        // salji poruku GPP da je obrada gotova
        status = MSGQ_put (info->GppMsgqObjekt, (MSGQ_Msg) msg) ;
        if (status != SYS_OK){ LOG_printf (&trace, "Greska kod MSGQ_put. Status = 0x%x\n",
        status); return status;}

        status = SEM_pend(&(info->notifyDspSemObj), SYS_FOREVER); // inace stavi TASK na
        cekanje
    }
    return status ;
}

/* ========================================================================*
*       zatvaranje MSGQ modula
*  ========================================================================*/
Int Close_MSGQ (TransferInfo * info)
{
    Int         status     = SYS_OK ;
    Bool        freeStatus = FALSE ;

    // zatvori MSGQ modul
    if (info->DspMsgqObjekt != NULL) {
        status = MSGQ_close (info->DspMsgqObjekt) ;
        if (status != SYS_OK){ LOG_printf (&trace, "Greska kod MSGQ_close. Status = 0x%x\n",
        status); return status;}
    }

    // oslobodi memoriju alociranu za TransferInfo strukturu
    freeStatus = MEM_free(DSPLINK_SEGID, info, sizeof (TransferInfo));
    if (freeStatus != TRUE) { LOG_printf (&trace, "Greska kod MEM_free. Status = 0x%x\n",
    status); return status;}

    return status;
}
```

```c
/*****************************************************************************
*       Img.c (DSP)
*****************************************************************************/

/*  ----------------------------------- OS Specific Headers             */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

/*  ----------------------------------- APP HEADER                      */
#include "StarTracker.h"

/*  ----------------------------------- IQ math HEADER                  */
#include <IQmath_inline.h>

#define     pi      3.14159265358979323846

double Img_Moment(t_pix *img, unsigned int sizeX, unsigned int sizeY, double Xmean, double
Ymean, double orderX, double orderY)
{
    double      r = 0;
    int         x,y;
    for( y = 0; y < sizeY; y++) {
        for( x = 0; x < sizeX; x++) {
            r += (pow((x-Xmean),orderX) * pow((y-Ymean),orderY) * (double)(img[x + y*sizeX]));
        }
    }
    return r;
}

int Img_GaussParam(t_pix *img, GaussParam * gauss)
{
    double      m11, m20, m02, m20phi, m02phi, phi, ra, su;
    unsigned int sizeX = gauss->block_size;
    unsigned int sizeY = gauss->block_size;

    gauss->V = Img_Moment(img, sizeX, sizeY ,0,0,0,0);
    gauss->Xmean = Img_Moment(img, sizeX, sizeY, 0,0,1,0)/(gauss->V);
    gauss->Ymean = Img_Moment(img, sizeX, sizeY, 0,0,0,1)/(gauss->V);

    m11 = Img_Moment(img, sizeX, sizeY, gauss->Xmean, gauss->Ymean, 1, 1)/(gauss->V);
    m20 = Img_Moment(img, sizeX, sizeY, gauss->Xmean, gauss->Ymean, 2, 0)/(gauss->V);
    m02 = Img_Moment(img, sizeX, sizeY, gauss->Xmean, gauss->Ymean, 0, 2)/(gauss->V);

    phi = atan2((2*m11),(m20-m02))/2;

    if ((m11==0)&&(m20-m02==0)) {
        phi = 0;
    } else {
        phi = -180*phi/pi;
    }

    gauss->phi = phi;
    // prvi način
    //m20phi = m20*pow(cos(phi),2) + m02*pow(sin(phi),2) + m11*sin(2*phi);
    //m02phi = m20*pow(sin(phi),2) + m02*pow(cos(phi),2) - m11*sin(2*phi);
    //gauss->sigma_x = sqrt(m20phi);
```



```c
    //gauss->sigma_y = sqrt(m02phi);

    // Drugi način
    ra = sqrt( pow((m20-m02),2) + 4*pow(m11,2) );
    su = m20+m02;
    gauss->sigma_x = sqrt((su+ra)/2);
    gauss->sigma_y = sqrt((su-ra)/2);

    if ((gauss->sigma_x == 0)||(gauss->sigma_y == 0)) {
        gauss->A = 0;
    } else {
        gauss->A = (gauss->V)/(2*pi*(gauss->sigma_x)*(gauss->sigma_y));
    }

    gauss->Xmean = gauss->Xmean + gauss->offsetX;
    gauss->Ymean = gauss->Ymean + gauss->offsetY;

    return 0;
}

_iq Img_MomentIQmath(t_pix *img, unsigned int sizeX, unsigned int sizeY,_iq Xmean, _iq Ymean,
_iq orderX, _iq orderY)
{
    _iq         r = 0;
    _iq         mul1, pow1, pow2;
    int         x,y;
    for( y = 0; y < sizeY; y++) {
        for( x = 0; x < sizeX; x++) {
            if(_IQtoF(orderX) == 0) {   pow1 = _IQ(1);
            } else if(_IQtoF(orderX) == 1) { pow1 = _IQ(x)-Xmean;
            } else if(_IQtoF(orderX) == 2) { pow1 = _IQrsmpy(_IQ(x)-Xmean),(_IQ(x)-Xmean));
            }

            if(_IQtoF(orderY) == 0) {   pow2 = _IQ(1);
            } else if(_IQtoF(orderY) == 1) { pow2 = _IQ(y)-Ymean;
            } else if(_IQtoF(orderY) == 2) { pow2 = _IQrsmpy(_IQ(y)-Ymean),(_IQ(y)-Ymean));
            }
            //pow1 = _IQpow(_IQ(x)-Xmean, orderX);
            //pow2 = _IQpow(_IQ(y)-Ymean, orderY);
            mul1 = _IQrsmpy(pow1,pow2);
            r += _IQrsmpy( mul1, _FtoIQ((float)(img[x + y*sizeX])));
        }
    }
    return r;
}

int Img_GaussParamIQmath(t_pix *img, GaussParam * gauss)
{
    _iq         mul1, mul2;
    _iq         qV, qXmean, qYmean, qm11, qm20, qm02, qphi, qra, qsu, qsigma_x, qsigma_y;
    unsigned int sizeX = gauss->block_size;
    unsigned int sizeY = gauss->block_size;

    qV = Img_MomentIQmath(img, sizeX, sizeY ,_IQ(0),_IQ(0),_IQ(0),_IQ(0));
    qXmean = _IQdiv(Img_MomentIQmath(img, sizeX, sizeY, _IQ(0),_IQ(0),_IQ(1),_IQ(0)),qV);
    qYmean = _IQdiv(Img_MomentIQmath(img, sizeX, sizeY, _IQ(0),_IQ(0),_IQ(0),_IQ(1)),qV);

    qm11 = _IQdiv(Img_MomentIQmath(img, sizeX, sizeY, qXmean, qYmean, _IQ(1), _IQ(1)),qV);
```

```
    qm20 = _IQdiv(Img_MomentIQmath(img, sizeX, sizeY, qXmean, qYmean, _IQ(2), _IQ(0)),qV);
    qm02 = _IQdiv(Img_MomentIQmath(img, sizeX, sizeY, qXmean, qYmean, _IQ(0), _IQ(2)),qV);

    qphi = _IQdiv(_IQatan2(_IQmpyI32( qm11, 2), (qm20-qm02)) , _IQ(2));

    if ((qm11==_IQ(0))&&(qm20-qm02==_IQ(0))) {
        qphi = _IQ(0);
    } else {
        qphi = _IQdiv(_IQmpyI32(qphi,180),_FtoIQ(pi));
    }

    gauss->phi = -(_IQtoF(qphi));

    // Drugi način
    qra = _IQsqrt( _IQrsmpy((qm20-qm02),(qm20-qm02)) + _IQmpyI32(_IQrsmpy(qm11,qm11),4) );
    qsu = qm20+qm02;
    qsigma_x = _IQsqrt(_IQdiv(qsu+qra,_IQ(2)));
    qsigma_y = _IQsqrt(_IQdiv(qsu-qra,_IQ(2)));

    if ((_IQtoF(qsigma_x) == 0)||(_IQtoF(qsigma_y) == 0)) {
        gauss->A = 0;
    } else {
        mul1 = _IQmpyI32(_FtoIQ(pi),2);
        mul2 = _IQrsmpy(qsigma_x,qsigma_y);
        gauss->A = _IQtoF(_IQdiv(qV, _IQrsmpy(mul1,mul2)));
    }

    gauss->sigma_x = _IQtoF(qsigma_x);
    gauss->sigma_y = _IQtoF(qsigma_y);
    gauss->V = _IQtoF(qV);

    gauss->Xmean = _IQtoF(qXmean);
    gauss->Ymean = _IQtoF(qYmean);

    gauss->Xmean = gauss->Xmean + gauss->offsetX;
    gauss->Ymean = gauss->Ymean + gauss->offsetY;

    return 0;
}
```

```
/*****************************************************************************
*       main.c (DSP)
*****************************************************************************/

/*  ----------------------------------- DSP/BIOS Headers            */
#include <std.h>
#include <log.h>
#include <swi.h>
#include <sys.h>
#include <sio.h>
#include <tsk.h>
#include <gio.h>
#include <msgq.h>
#include <pool.h>

/*  ----------------------------------- DSP/BIOS LINK Headers       */
#include <dsplink.h>
#include <failure.h>

/*  ----------------------------------- Sample Headers              */
#include "readwrite.h"
#include "tskReadwrite.h"
#include "readwrite_config.h"

#define FILEID              FID_APP_C

// LOG objekt potreban za LOG_printf
extern LOG_Obj trace;

// broj iteracija
Uint32 BrojIteracija;

// pretvorba stringa u long int
extern long atol(const char *str);

// prototip TSK handle funkcije
static Int TSK_handle ();

/* ========================================================================*
*       glavna funkcija (main)
*  ========================================================================*/
Void main(Int argc, Char *argv[])
{
    Int  status = SYS_OK ;
    TSK_Handle     tskTask ;

    // inicijaliziraj DSP/BIOS Link
    DSPLINK_init () ;

    // broj iteracija poslan sa GPP preko argumenta
    BrojIteracija = atol (argv[0]) ;

     tskTask = TSK_create(TSK_handle, NULL, 0);
     if (tskTask == NULL) LOG_printf (&trace, "Greska kod TSK_create. Status = 0x%x\n",
     status);

    return;
}
```



```
/* ========================================================================*
*       TSK handle funkcija
*  ========================================================================*/
static Int TSK_handle()
{
    Int    status = SYS_OK;
    TransferInfo * info;

   // inicijalizacija
    status = Init_MSGQ (&info);
    if (status != SYS_OK) LOG_printf (&trace, "Greska kod TSKRDWR_create. Status = 0x%x\n",
    status);

   // izvodjenje programa (glavna petlja)
   status = Izvodjenje (info);
    if (status != SYS_OK) LOG_printf (&trace, "Greska kod TSKRDWR_execute. Status = 0x%x\n",
    status);

    // zatvaranje MSGQ modula
    status = Close_MSGQ (info);
    if (status != SYS_OK) LOG_printf (&trace, "Greska kod TSKRDWR_delete. Status = 0x%x\n",
    status);

    return status;
}
```

```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    clean_noise.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

function img_out = clean_noise(img,offset)
    [rows,cols]=size(img);

    for x = 1:rows
        for y = 1:cols
            if img(x,y) <= offset
                img_out(x,y) = 0.0;
            else
                img_out(x,y) = img(x,y);
            end;
        end;
    end;
end
```



```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    moment.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

function m = moment(img,X,Y,nx,ny)
    [rows,cols]=size(img);
    m = 0;
    for y = 1:rows
        for x = 1:cols
            m = m + ((x-X)^nx)*((y-Y)^ny)*img(y,x);
        end;
    end;
    %m = m/sum(img(:));
end
```

```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   gauss_param.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

function [ Xmean Ymean A sigma_x sigma_y phi O V ] = gauss_param( img )

    %V = moment(img,Xmean,Ymean,0,0);
    V = sum(img(:));
    Xmean = moment(img,0,0,1,0)/V;
    Ymean = moment(img,0,0,0,1)/V;

    m11 = moment(img,Xmean,Ymean,1,1)/V;
    m20 = moment(img,Xmean,Ymean,2,0)/V;
    m02 = moment(img,Xmean,Ymean,0,2)/V;

    phi = atan2((2*m11),(m20-m02))/2;

    if ((m11==0)&&(m20-m02==0))
        phi = 0;
    end

    % Prvi način
    m20phi= m20*cos(phi)^2 + m02*sin(phi)^2 + m11*sin(2*phi);
    m02phi= m20*sin(phi)^2 + m02*cos(phi)^2 - m11*sin(2*phi);
    sigma_x = sqrt(m20phi);
    sigma_y = sqrt(m02phi);

    % Drugi način, koji automatski veću sigmu stavi u sigma_x
    %ra=sqrt( (m20-m02)^2 + 4*m11^2 );
    %su=(m20+m02);
    %sigma_x=sqrt((su+ra)/2);
    %sigma_y=sqrt((su-ra)/2);

    phi = -phi;
    O = 180*phi/pi;

    if ((sigma_x==0)||(sigma_y==0))
        A = 0;
    else
        A = V/(2*pi*sigma_x*sigma_y);
    end

end
```



```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   move_dist.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

function [ ExpectedDist   ] = move_dist( img )

    ExpectedDist = 6*(starCurent->sigma_x) - 6*(starCurent->sigma_y);

end
```

```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   isolate_star_small.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

function [ X Y block_size error ] = isolate_star_small(img,x,y,pix_offset)
    [rows,cols] = size(img);
    error = 0;
    clear_cnt = 0;
    dir = 0; % 0 pozitivan smjer, 1 negativan smjer
    clear_flag = 0;
    step = 1;
    pos_x = x;
    pos_y = y;
    while 1
        % x
        for i = 1:step
            if ~dir
                pos_x = pos_x + 1;
            else
                pos_x = pos_x - 1;
            end
            if ((pos_y <= rows)&&(pos_y > 0))&&((pos_x <= cols)&&(pos_x > 0))
                if (img(pos_y,pos_x) > pix_offset)
                    clear_flag = 1;
                end
            else
                error = 1;
            end
        end

        if ~clear_flag
            clear_cnt = clear_cnt + 1;
            if (clear_cnt == 4)||(error == 1)
                if ~dir
                    X = pos_x - step;
                    Y = pos_y;
                else
                    X = pos_x;
                    Y = pos_y - step + 1;
                end
                break;
            end
        else
            clear_flag = 0;
            clear_cnt = 0;
        end
        % y
        for i = 1:step
            if ~dir
                pos_y = pos_y + 1;
            else
                pos_y = pos_y - 1;
            end
            if ((pos_y <= rows)&&(pos_y > 0))&&((pos_x <= cols)&&(pos_x > 0))
                if (img(pos_y,pos_x) > pix_offset)
                    clear_flag = 1;
                end
            else
```



```
                error = 1;
            end
        end

        if ~clear_flag
            clear_cnt = clear_cnt + 1;
            if (clear_cnt == 4)||(error == 1)
                if ~dir
                    X = pos_x - step;
                    Y = pos_y - step;
                else
                    X = pos_x;
                    Y = pos_y;
                end
                break;
            end
        else
            clear_flag = 0;
            clear_cnt = 0;
        end
        % uvećaj step i promijeni smjer
        step = step + 1;
        if(~dir)
            dir = 1;
        else
            dir = 0;
        end
    end

    if (X < 1) X = 1; end;
    if (Y < 1) Y = 1; end;

    block_size = step;
    if (error == 1)
        if (block_size + X > cols + 1)
            block_size = cols + 1 - X;
        end
        if (block_size + Y > rows + 1)
            block_size = rows + 1 - Y;
        end
    end

end
```

```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   gauss2d.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

function [ img ] = gauss2d(img, Xmean, Ymean, A, sigma_x, sigma_y, theta )
    % u kartezijevom koordinatnom sustavu theta ide u negativnom smjeru.
    % na slici (y os ide u suprotnom smjeru) se to vidi kao rotacija u
    % pozitivnom smjeru.
    [rows,cols]=size(img);
    for y = 1:rows
        for x = 1:cols
            a = cos(theta)^2/2/sigma_x^2 + sin(theta)^2/2/sigma_y^2;
            b = -sin(2*theta)/4/sigma_x^2 + sin(2*theta)/4/sigma_y^2 ;
            c = sin(theta)^2/2/sigma_x^2 + cos(theta)^2/2/sigma_y^2;
            img(y,x) = A*exp( - (a*(x-Xmean).^2 + 2*b*(x-Xmean).*(y-Ymean) + c*(y-Ymean).^2));
        end;
    end;
end
```



```
%{ %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   star_tracker.m
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

close all;
clear all;

img_num = 50;
% zadaj offset kao postotak
percent = 0.99;
MAX_ANG_DIFF = 15;
MAX_DIST_TOL = 0.5;
MAX_MOVE_DIST = 10000;

Xmean       = zeros(400,50);
Ymean       = zeros(400,50);
sigma_X     = zeros(400,50);
sigma_Y     = zeros(400,50);
O           = zeros(400,50);
A           = zeros(400,50);
V           = zeros(400,50);
Dist        = zeros(400,50);
sigma_omjer = zeros(400,50);

for sekv = 1:img_num
    filename = strcat ('sekv_', num2str (sekv), '.png');
    img = double(imread(filename));

    [rows,cols]=size(img);
    img_star_clean  = zeros(rows,cols);
    img_gauss       = zeros(rows,cols);

    pixS = sort(img(:));
    pix_offset(sekv) = pixS(floor(length(pixS)*percent));

    % kopiraj da originalna slika bude očuvana
    img_clean = img;

    % očišćena slika "prije" i "poslije"
    img_clean_before = img_clean;
    img_clean_affter = img_clean;

    cnt(sekv) = 1;
    max_pix = max(img_clean(:));
    while ((max_pix > pix_offset(sekv))&&(cnt(sekv) <= 400)) % dok postoje svijetli pikseli
    (zvijezde) na ociscenoj slici
        [row,col] = find(img_clean == max_pix,1);
        % izoliraj zvijezdu
        [ offset_x offset_y block_size rubna] = isolate_star_small(img_clean, col, row,
        pix_offset(sekv));
        img_star_clean(1:block_size,1:block_size) = clean_noise(img_clean(offset_y:offset_y+
        block_size-1, offset_x:offset_x+block_size-1),pix_offset(sekv));
        % makni trenutno najsvijetliji pixel i pripadajuću zvijezdu sa ociscene slike
        img_clean(offset_y:offset_y+block_size-1,offset_x:offset_x+block_size-1) = 0;
        % nadi najsvijetliji piksel
        max_pix = max(img_clean(:));

        if (rubna == 0) % ako zvijezda nije na rubu slike
```

```
% nadi parametre gaussiana
[ x_mean y_mean a sigma_x sigma_y phi o v ] = gauss_param( img_star_clean(1:
block_size,1:block_size) );

% ako su parametri gaussiana dobri i volumen je dovoljno velik (dovoljno velika
zvijezda)
if (a~=0)&&(v > 200)
    % generiraj gauss ako su dobri parametri
    theta = -phi;
    img_gauss(1:block_size,1:block_size) = gauss2d(img_gauss(1:block_size,1:
    block_size), x_mean, y_mean, a, sigma_x, sigma_y, theta);
    % oduzmi generirani gaussian (zvijezdu) od originalne slike
    img_clean_affter(offset_y:offset_y+block_size-1, offset_x:offset_x+block_size
    -1) = abs(img(offset_y:offset_y+block_size-1, offset_x:offset_x+block_size-1)
     - img_gauss(1:block_size,1:block_size));
    % pretvori lokalne koordinate u globalne i spremi parametre zvijezde
    Xmean(cnt(sekv),sekv) = offset_x + x_mean - 1;
    Ymean(cnt(sekv),sekv) = offset_y + y_mean - 1;
    sigma_X(cnt(sekv),sekv) = sigma_x;
    sigma_Y(cnt(sekv),sekv) = sigma_y;
    sigma_omjer(cnt(sekv),sekv) = sigma_x/sigma_y;
    V(cnt(sekv),sekv) = v;
    O(cnt(sekv),sekv) = o;
    A(cnt(sekv),sekv) = a;
    %% move distance
    Dist(cnt(sekv),sekv) = MAX_MOVE_DIST;
    ExpectedDist(cnt(sekv),sekv) = 5*sigma_x + 5*sigma_y;
    if sekv > 1
        for i = 1:cnt(sekv-1)
            Xdist = Xmean(cnt(sekv),sekv) - Xmean(i,sekv-1);
            Ydist = Ymean(cnt(sekv),sekv) - Ymean(i,sekv-1);
            % oderdi kut, oba smjera
            phi1 = atan2(Ydist, Xdist);
            if (Ydist==0)&&(Xdist==0)
                phi1 = 0;
                phi2 = 0;
            else
                phi1 = -180*phi1/pi;
                phi2 = phi1 - 180;
                % dovedi u granice [-180, 180]
                if phi2 < -180
                    phi2 = phi2 + 360;
                end
            end
            % provjeri dal se zvijezda nalazi unutar tolerancija kuta
            if (MAX_ANG_DIFF > abs(phi1 - O(cnt(sekv),sekv)))||(MAX_ANG_DIFF >
            abs(phi2 - O(cnt(sekv),sekv)))
                % racunaj udaljenost
                dist = sqrt(Xdist^2 + Ydist^2);

                % spremi najmanju udaljenost koja je u granicama tolerancije
                if (abs(ExpectedDist(cnt(sekv),sekv) - dist)/ExpectedDist(cnt(
                sekv),sekv) < MAX_DIST_TOL)&&(abs(ExpectedDist(cnt(sekv),sekv) -
                dist) < abs(ExpectedDist(cnt(sekv),sekv) - Dist(cnt(sekv),sekv)))
                    Dist(cnt(sekv),sekv) = dist;
                    % korigiraj kut gaussiana (time se nista ne gubi, samo se po
                    potrebi zarotira za 180°)
                    if (MAX_ANG_DIFF < abs(phi1 - O(cnt(sekv),sekv)))
                        O(cnt(sekv),sekv) = O(cnt(sekv),sekv) - 180;
                        if (O(cnt(sekv),sekv) < -180)
                            O(cnt(sekv),sekv) = O(cnt(sekv),sekv) + 360;
                        end
                    end
                end
            end
        end
        % ako zvijezda na prijasnjoj slici nije nadjena stavi da se nezna udaljenost
        if  Dist(cnt(sekv),sekv) == MAX_MOVE_DIST
            Dist(cnt(sekv),sekv) = NaN;
        end
        %% samo zvijezde za koje su parametri gaussiana dobri i volumen je dovoljno
        velik se uzimaju u obzir
        cnt(sekv) = cnt(sekv) + 1;
    else
        % inače oduzmi originalnu zvijezdu od originalne slike
        img_clean_affter(offset_y:offset_y+block_size-1,offset_x:offset_x+block_size-
        1) = img(offset_y:offset_y+block_size-1,offset_x:offset_x+block_size-1) -
        img_star_clean(1:block_size,1:block_size);
    end

else
    % ako je zvijezda na rubu slike, ignoriraj je i makni
    img_clean_affter(offset_y:offset_y+block_size-1,offset_x:offset_x+block_size-1) =
     img(offset_y:offset_y+block_size-1,offset_x:offset_x+block_size-1) -
    img_star_clean(1:block_size,1:block_size);
end

img_clean_affter = clean_noise(img_clean_affter,pix_offset(sekv));
%E(sekv) = sum(sum(img_clean_affter(:)));
%figure, imagesc(img_clean_affter), colormap(gray), title('stars');
%display(sekv), display(E(sekv))
end

%% statistika

%sekv = 2;
%[Xmean(1:cnt(sekv)-1,sekv)-1 Ymean(1:cnt(sekv)-1,sekv)-1 sigma_X(1:cnt(sekv)-1,sekv)
sigma_Y(1:cnt(sekv)-1,sekv) O(1:cnt(sekv)-1,sekv) ExpectedDist(1:cnt(sekv)-1,sekv)
Dist(1:cnt(sekv)-1,sekv)] % slika broj sekv, sve zvijezde
%[Xmean(1:cnt(sekv+1)-1,sekv+1)-1 Ymean(1:cnt(sekv+1)-1,sekv+1)-1
sigma_X(1:cnt(sekv+1)-1,sekv+1) sigma_Y(1:cnt(sekv+1)-1,sekv+1) O(1:cnt(sekv+1)-1,sekv+1)
ExpectedDist(1:cnt(sekv+1)-1,sekv+1) Dist(1:cnt(sekv+1)-1,sekv+1)] % slika broj sekv+1, sve
zvijezde

% nadji srednji kut na nosnovu svih kuteva i na osnovu kuteva od zvijezde za koje je naden
pomak
for slika = 2:img_num
    display(slika);
    % ispitaj koji su NaN (za koje nije naden pomak)
    x = find(isnan(Dist(1:cnt(slika)-1,slika)));
    nove_zvijezde = length(x) % broj zvijezdi za koje se nezna pomak
    prepoznate_zvijezde = cnt(slika)-1-length(x) % broj zvijezdi za koje je naden pomak
    srenji_kut = mean(O(find(not(isnan(Dist(1:cnt(slika)-1,slika))),slika)) % srenji kut
    srednja_pomak = mean(Dist(find(not(isnan(Dist(1:cnt(slika)-1,slika))),slika)) % srednji
```

```
    pomak
    srednji_ocekivani_pomak = mean(ExpectedDist(find(not(isnan(Dist(1:cnt(slika)-1,slika)))),
    slika)) % srednji očekivani pomak
end
```