SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 511

AUTOMATSKO UPRAVLJANJE TELESKOPOM U SVRHU PRAĆENJA POGLEDA

Nikola Petrak

Zagreb, veljača 2013.

Diplomski zadatak :

Automatsko upravljanje teleskopom u svrhu praćenja pogleda

U okviru diplomskog rada potrebno je razviti sustav za automatsko upravljanje teleskopom na ekvatorijalnoj montaži po osi desnog uzdizanja i po osi deklinacije. Upravljanje je potrebno ostvariti na temelju promjene odnosno pomaka slike dobivene sustavom uslikavanja koji se nalazi u fokusu teleskopa. Sustav uslikavanja temeljiti na Web kameri s USB priključkom, a sustav upravljanja realizirati pomoću ugradbenog računalnog sustava Beagleboard. Provjeriti točnost rada sustava određivanjem pogreške vođenja. Definirati način kalibracije radi određivanja kuta zakreta pogleda u odnosu na osi vođenja. Za detaljnije informacije kontaktirati mentora.

rad posvećujem mojoj Sunčici

Zahvaljujem se svima koji su doprinijeli izradi ovog diplomskog rada,
posebno prof.dr.sc Davoru Petrinoviću na savijetima i mentorstvu
te Marijanu Kuriju na pomoći oko izrade tiskane pločice

Sadržaj:

1. Uvod

Sustav za automatsko praćenje pogleda teleskopa omogućuje usmjeravanje i fiksiranje optičkog sustava zemaljskog teleskopa na jednu točku na nebeskom svodu. Noćno nebo se prividno kreće od istoka na zapad brzinom $7.2921150*10^{-5}$ radijana po sekundi pa je ekvivalentan kompenzacijski pomak montaže teleskopa potreban.

U današnje vrijeme postoji nekoliko vrsta različitih montaža teleskopa, kao i teleskopa različitih veličina, masa, optičke razlučivosti i preciznosti. Većina ekvatorijalnih montaža jesu automatizirane no zbog nesavršenosti u konstrukciji, postavljanju i kalibraciji uređaja javljaju se pogreške vođenja koje znatno mogu utjecati na kvalitetu slike koju promatramo. Korištenjem ugradbenog sustava Beagleboard te najmodernije inačice programskog paketa Matlab navedene nesavršenosti će se nastojati zaobići. Program razvijen u Matlab okruženju sastoji se od nekoliko pod algoritama:

- Detektora zvijezda,
- blokovske usporedbe,
- izračuna zakreta kamere,
- izdavanje naredbi vođenja.

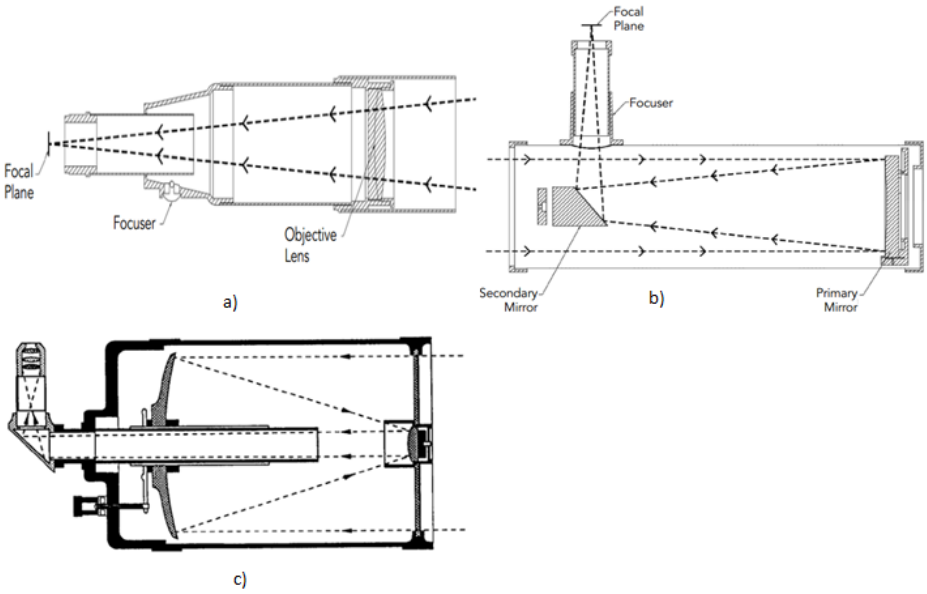
Svaki od pod algoritama treba se najviše moguće simplificirati i optimizirati za izvođenje na moćnom, ali ipak ograničenom ugradbenom sustavu. S toga je za potrebe testiranja algoritma konstruirana jedna inačica, a za potrebe izvođenja konstruirana je jednostavnija inačica pod algoritama.

Razvijeni sustav se koristi u astrofotografiji, gdje je, zbog velikog trajanja ekspozcije fotografija, potrebno dugo vremena držati optiku fiksiranu u jednu točku. Sam algoritam je široko primjenjiv i koristi se u automatskom navođenju svemirskih letjelica, projektila, raspoznavanju mikroskopskih čestica i sl.

2. Fizička izvedba

2.1. Teleskop

Teleskop grč. *tele+scopos* (dalek+ pogled) je naprava za promatranje udaljenih predmeta. Danas postoji više vrsta teleskopa, a klasificiraju se po valnim duljinama zračenja koje sakupljaju. To su radioteleskopi, ultraljubičasti teleskopi infracrveni teleskopi, teleskopi x-zračenja i optički teleskopi. Optički teleskop je instrument koji sabire i fokusira svjetlo. Način oblikovanja svjetlosnog snopa koji dolazi u okular teleskopa određuje njegovu vrstu. Refraktorni teleskopi primjerice koriste optičke leće. Prvi takvi teleskopi (1600. g.) koristili su jednostruku leću koja se ponaša kao prizma te dolazi do pojave kromatske aberacije odnosno razlaganja jednobojnog svjetla u više boja. Kako bi se zaobišla ta pojava postavljena je i druga leća koja ima drugačiji indeks loma kako bi se dvije različite valne duljine mogle fokusirati na isto mjesto. Druga vrsta teleskopa koja se odmah nakon toga pojavila je Newtonov teleskop ili reflektivni koji koristi jedno konkavno zrcalo kao primarnu napravu za fokus svjetla. Drugi dio za fokusiranje svjetla je koso zrcalo koje fokusira zraku na okular. Treća najpoznatija vrsta teleskopa, ona koja se koristi u ovom radu je Schmidt-Cassegrain koja koristi kombinaciju leća i zrcala te se na njega referencira kao katadioptarski teleskop. Naziv je dobiven jer koristi propusna korekcijska pločica sa sfernim primarnim i eliptičnim sekundarnim zrcalom Zahvaljujući prilično kompleksnoj kombinaciji leća moguće je postići veliko povećanje sa prilično malom teleskopskom cijevi. Ova vrsta teleskopa sastoji se od kolektorske ploče, sfernog primarnog zrcala i sekundarnog zrcala. Kada zrake uđu u optički sustav prevaljuju duljinu cijevi tri ili više puta ovisno o njegovoj konstrukciji.



Slika 1:vrste teleskopa a) refraktorni b) newtonov c) Schmidt-Cassegrain

Danas, većina amaterskih kao i velikih teleskopa za istraživanje koriste zrcalo kao primarni sakupljač svjetla i reflektorskog su tipa. Stakleni supstrat se koristi se kako bi držao optičku figuru dok se refelektivnost postiže tankim slojem aluminija koje se naparuje na zrcalo. Osnovne podvrste teleskopa prikazane su slikom 1.

2.2. Optička svojstva teleskopa

Teleskop ima nekoliko važnih svojstava u kojima se teleskopi, zbog svoje konstrukcije mogu razlikovati. To su žarišna duljina i f-omjer, veličina pogleda i pokrivanje neba te kutna rezolucija. Žarišna duljina je zapravo udaljenost od leće pa do mjesta projekcije. F-omjer se definira kao omjer žarišne duljine i promjera primarne leće ili zrcala. Žarišna duljina određuje veličinu slike, dok promjer određuje količinu svjetla koje upada na mjesto projekcije. Sustavi s malim F-omjerom imaju velike količine svjetla s obzirom na veličinu slike. Takvi

sustavi zovu se brzim sustavima, dok su sustavi s velikim F-omjerom brzi sustavi.

Kutovi među objektima na slici određeni su fokalnom duljinom sustava. Primjerice ako su neka dva objekta međusobno razmaknuti za kut α, gledano okomito na cijev teleskopa, tada se njihova međusobna udaljenost na slici računa formulom:

$$S = f\alpha \tag{1}$$

Kut α mjeri se u radijanima, dok se žarišna duljina mjeri u kutnim sekundama odnosno minutama po milimetru. Faktor konverzije radijana i kutnih sekundi je 20635.

U velikim optičkim sustavima i preciznim teleskopima žarišna duljina je višestruko veća od onih jednostavnijih, dodatnim zrcalima, pa govorimo o efektivnoj fokalnoj duljini, koja se razlikuje od fokalne duljine primarnog zrcala ili leće.

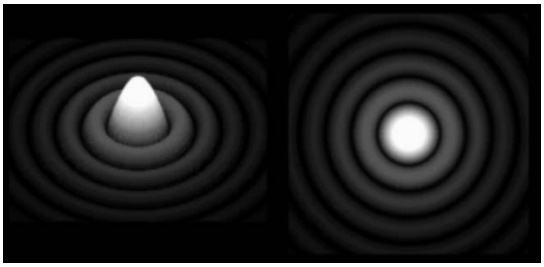
Vidno polje teleskopa (FOV- eng. *field of view*) je površina neba koje je obuhvaćeno slikom slikanom na teleskopu. Ono ovisi o fokalnoj duljini i površini detektora svjetla. Analogno kutnoj udaljenosti dva objekta na nebu, postoji i kutna razapetost površine na koju se u literaturi [1] referencira kao na čvrsti kut (eng. *solid angle*). Jedinica čvrstog kuta je u steradijanima odnosno u „kvadratnim radijanima“ ili „kvadratnim stupnjevima“. Unatoč brojnim postignućima u izradi optički moćnijih teleskopa i površinom velikih optičkih senzora vidno polje teleskopa je i dalje malo. Iz toga se razloga se slika na teleskopima lomi i fokusira na višestruke senzore.

Kutna rezolucija, odnosno razlikovanje objekata unutar vidnog polja teleskopa također je jedno od optičkih svojstava teleskopa koje je ograničeno svojstvima promatranog objekta. Naime zbog valne prirode svjetlosti svaki se točkasti izvor svjetlosti raspršuje i dolazi na metu u uzorku zrakastog diska. Zrakasti disk se sastoji od centralnog vrha koji je najsvjetliji, zatim od serije tamnih i svijetlih koncentričnih prstena. Kutna veličina zrakastog diska nekog izvora svjetlosti

ovisi o promjeru primarnog zrcala ili leće kao i o valnoj duljini svjetlosti koji upada na optiku teleskopa. Kutni promjer računa se sljedećom formulom:

$$\sigma = \frac{1.22 * \lambda}{D} \quad (2)$$

Gdje je D promjer zrcala, a λ valna duljina svjetlosti. Veći će teleskopi imati manji σ ali samim time i veću rezoluciju, a zvijezde neće biti toliko razmazane te će ih se moći više razlučiti. Slika 2 prikazuje amplitudu i oblik zrakastog diska.



Slika 2: PSF funkcija zrakastog diska

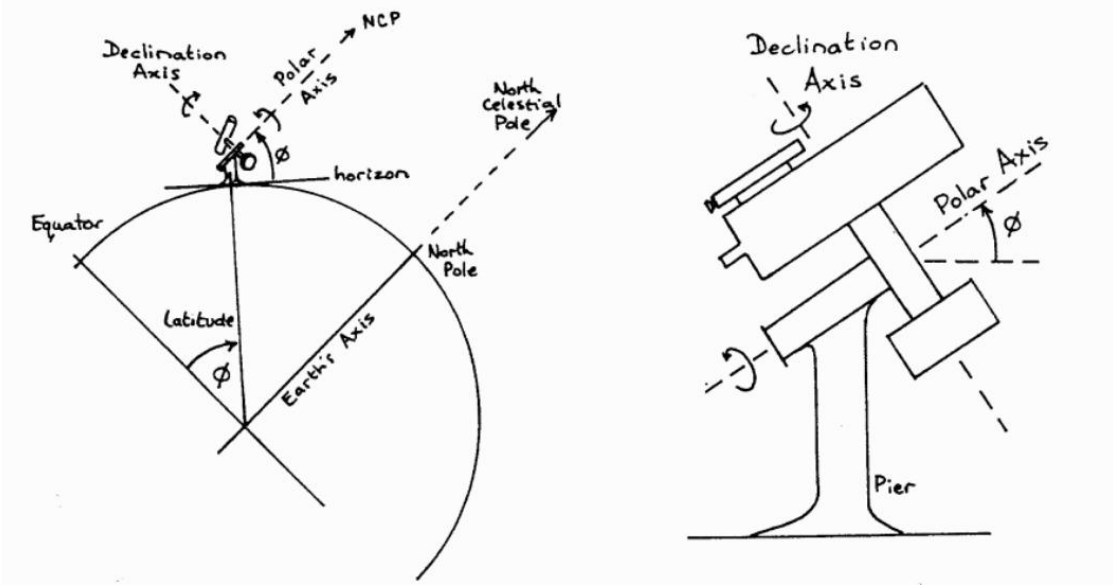
2.3. Montaža teleskopa

Kvaliteta slika nastalih iz teleskopa uvelike ovisi o njegovoj montaži. Naime kako je primarna namjena teleskopa povećanje oku jedva vidljivih objekata, tako se i jedva osjetljive smetnje izazvane vibracijama, trešnjom, nepravilnostima pri pomicanju teleskopske cijevi, povećavaju i postaju izvor pogrešaka i slika loše kvalitete.

Postoje dvije najpoznatije vrste montiranja teleskopa na stalak odnosno stativ. To su ekvatorijalna i alt-azimutna montaža. Ekvatorijalne montaže postavljaju se tako da je jedna (polarna) os pod kutom koji odgovara zemljopisnoj širini na kojoj se promatranje izvršava. To se izvodi tako da se teleskop usmjerava paralelno osi koja prolazi kroz sjeverni ili južni nebeski pol. Kako bi se kompenzirala rotacija zemlje teleskop i montaža pomiču se oko te osi, obično korištenjem motora koji pomiče montažu kutnom brzinom jednaku zemljinoj. Ova os naziva se polarna ili os desnog uzdizanja (dalje u radu RA). Oko ove osi teleskopska montaža ima pomične utege za protutežu, ako se radi o tzv.

Njemačkoj montaži, postavljene na šipci okomitoj na os, a omogućuju rotiranje teleskopa oko osi bez gubljenja ravnoteže.

Druga os koja se mora postaviti zove se deklinacijska os. (dalje u radu DEC) Ova os omogućava pomicanje teleskopa okomito na RA. Jednom kad je željeni nebeski objekt pronađen obje osi se mogu zaključati te bi teleskop automatski trebao pratiti objekt.

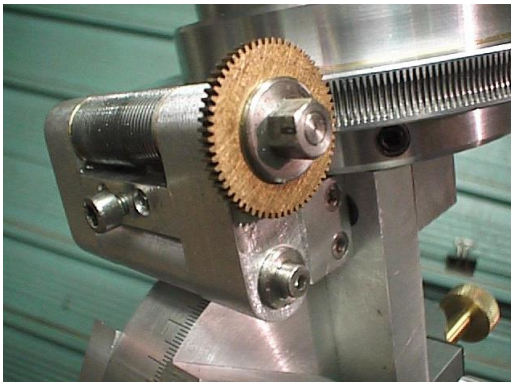


Slika 3: ekvatorijalna montaža

Automatizirane ekvatorijalne montaže imaju i deklinacijsku os motoriziranu kako bi se postavila u pravilan položaj i zaključala. Do eventualnih pogrešaka koje se ovim motorom mogu ispraviti dolazi ukoliko polarna os nije dobro poravnata pa objekt bježi po deklinacijskoj osi. Ovaj problem bi se trebao rješavati ponovnim baždarenjem položaja teleskopa. Motor RA osi okreće se konstantnom brzinom no moguće je ubrzavanje odnosno usporavanje teleskopa vanjskom pobudom na motor. Konstrukcija RA osi (zupčanik, pužni kotač i motor) se okrene za puni krug u 23,93447 sati. Pužni kotač i motor prikazan je slikom 3. Točnije jedan okret u 23 sata, 56 minuta i 4.091 sekundi odnosno 86164.091 sekundi za puni okret od 360 stupnjeva. Pogonska konstrukcija je izveden tako iz razloga što 24 sata obuhvaća samo vrtnju zemlje oko svoje osi, ali ne i oko Sunca, pa ako se

treba izračunati okretanje zemlje prema dalekim zvijezdama tada se mora uzeti i vrtnja oko sunca tijekom jedne godine.

Puni krug ima 360*3600 tj. 1296000 kutnih sekundi, dakle rotacija neba u jednoj vremenskoj sekundi prema siderealu iznosi 15.041 kutnih sekundi. Dok bi uz zanemarivanje rotacije oko sunca to bilo (360*3600)/(24*3600) = 15.000 kutnih sekundi.



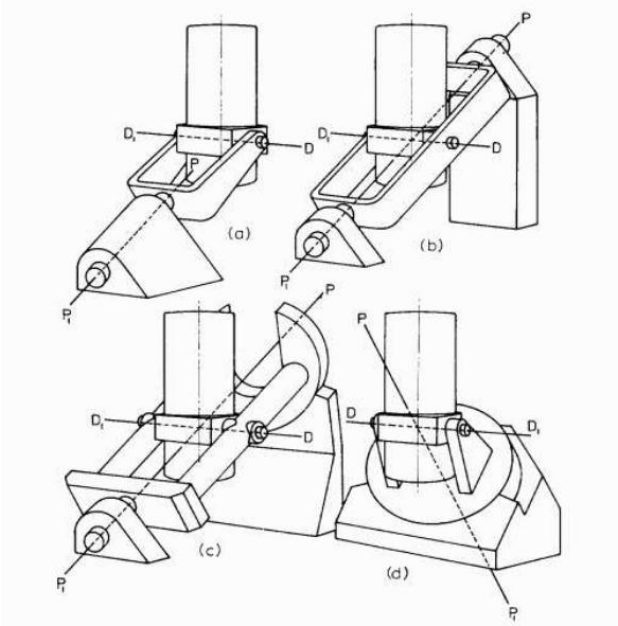
Slika 4: pužni mehanizam

Postoji nekoliko podvrsta ekvatorijalnih montaža koje se razlikuju po konstrukciji te su namijenjene teleskopima različitih veličina:

- Otvorena viljuška (eng. *open fork*) – na svome podnožju je priključen na sidereal koji se okreće oko RA osi. Sam teleskop je priključen na dvije točke rotacije na vrhu viljuške kako bi se mogao pokretati po deklinacijskoj osi, slika X – a). Postolje je za profesionalne teleskope veličine 0.5 – 2 m,
- engleska montaža – sastoji se od okvira koji sadrži ležišta za RA os na počecima kraju okvira. U središnjem dijelu okvira nalaze se dvije rotacijske točke na koje je priključen teleskop koji se onda može pomicati po obje osi. Prosječna veličina teleskopa koja se priključuje na montažu je 2,5 m. Nedostatak montaže je što se teleskop ne može usmjeriti u točku koja je blizu sjevernog ili južnog nebeskog pola,
- montaža u obliku potkove – slične je funkcionalnosti kao i engleska montaža ali zaobilazi nedostatak engleske montaže tako da se teleskop

pozicionira unutar dijela u obliku potkove koji se nalazi na vrhu okvira. Ovakvu montažu ima Haleov teleskop veličine 5,1 m,

- montaža polarnog diska –sličan montaži u obliku viljuške. Polarna os postavlja se okomito na disk koji se može rotirati i pomicati gore-dolje. Prednost ove montaže je precizno usmjeravanje teleskopa zbog veličine diska,
- Njemačka ekvatorijalna montaža – teleskop se nalazi na poluzi koja mora biti u ravnoteži kako bi mirno stajala te se stoga s druge strane šipke stavljaju utezi na određenoj udaljenosti ovisno o nagibu postavljenog teleskopa.



Slika 5: a)otvorena montaža, b)engleska montaža, c)potkovasta montaža d)montaža polarnog diska

Alt-azimutne montaže slične su ekvatorijalnoj montaži viljuške i imaju mogućnost horizontalnog i vertikalnog pomaka. Rotacija oko vertikalne osi mijenja azimut, (orijentacija kompasa), a rotacija oko horizontalne osi mijenja visinu (eng. *altitude*) tj. kut elevacije. Prednost im je podržavanje većih i težih konstrukcija.

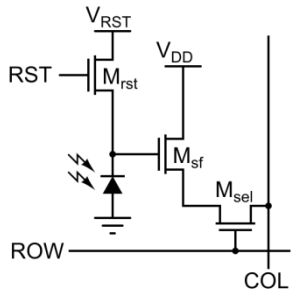
Težište teleskopa je u njegovom podnožju. Radi se o teleskopima čije su cijevi duljine od 6 m do 10 m. Superiorniji su od ekvatorijalnih montaža u tome što im je vertikalno opterećenje konstantno i simetrični, a rotacija cijevi teleskopa odvija se samo u jednoj ravnini, ovisno samo o kutu zenita. Nedostatak ove montaže je konstantna potreba za izračunavanjem visine i azimuta iz RA i Dec podataka o trenutnom položaju kako bi se dobile slike koje ne rotiraju. Bitni predstavnici ove skupine montaža su Dobsonova te Goto. Dobsonova montaža koristi ne mehaničke dijelove za konstrukciju te je jeftin za izgradnju, posebno je zanimljiv astronomima hobistima. Goto montaže koriste računalno pokretanje obaju osi teleskopa u svrhu praćenja nebeskih objekata.

2.4. Kamera i beagleboard sustav

2.4.1. Logitech c310

Korištena web kamera je Logitech c310. Radi se o kameri visoke rezolcije i konfigurabilne ekspozicije. Središnji dio sustava kamere je CMOS senzor, sa plastičnim lećama i poljem pogleda od 60°. Optička rezolucija je 1280x960 piksela. Maksimalna frekvencija slikanja je 30 sličica po sekundi.

CMOS senzor sastoji se od matrice jednostavnih poluvodičkih aktivnih senzora. Aktivni senzor je integrirani sklop baziran na foto osjetljivoj diodi, a sastoji se još i prijenosnog, odabirnog i tranzistora za brisanje stanja (eng.*reset transistor*). Dodatno na supstrat se stavlja i sloj silicija koji se naziva difuzijska regija koja povećava kapacitet pohrane naboja. Shema tro-tranzistorskog aktivnog piksela prikazana je na slici 6.



Slika 6: tro-tranzistorski piksel CMOS senzora

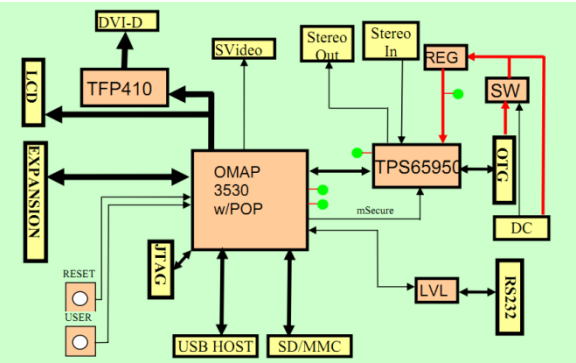
Piksel prima svjetlost s određenom kvantnom efikasnosti QE (eng.*quantum efficacy)*, odnosno samo se dio svjetlosti konvertira u slobodne elektrone. Nakon što je dovoljno elektrona aktiviralo fotodiodu, aktivira se M_{sf} odnosno prijenosni tranzistor i ćelija postaje aktivna. Prednost korištenja CMOS senzora naprema CCD senzorima, koji su alternativa, je jeftinija izrada te sprječavanje pojave prelijevanja svjetlosti. Pojava se javlja pri prevelikoj ekspoziciji, kada se zbog previše svjetlosti na jednom pikselu aktiviraju elektroni i u okolnim pikselima.

Nakon prihvata slike sa CMOS senzora, slika se šalje u procesor za obradu signala (eng. DSP *digital signal processor*). DSP na web kameri c310 ima ugrađene filtre za potiskivanje smetnji. Informaciju o kakvim je točno filtrima riječ proizvođač nije objavio.

2.4.2. Beagleboard C4

Pločica BeagleBoard razvijena je u suradnji tvrtki Texas Instruments i DigiKey. Radi se o ugradbenom sustavu malog formata i velike snage, modularnosti i nadogradivosti. Temeljena je na procesoru OMAP3530, koji je zapravo cijeli sustav na jednom čipu. Procesor sadrži superskalarnu jezgru Cortex8 ARM sa dodatnom TMS320C64x+ DSP jezgrom sa podrškom za openGL 2D/3D grafiku, HD video i brojnu USB periferiju. Na razvojnoj pločici Beagleboard je postavljeno nekoliko standardnih sučelja oko OMAP procesora koja omogućuju veliku proširivost funkcionalnosti pločice te tako omogućuju

iskorištavanje snage procesora. Slika 7 prikazuje blok shemu Beagleboard sustava. Na OMAP procesor spojena su dva integrirana sklopa TFP410, univerzalni grafički kontroler koji omogućuje prikaz slike na DVI (eng. *digital video interface*) sučelje i TPS65950 koji služi kao regulator izvora napajanja za sustav i USB, a sadrži i audio kodek za prijenos zvučnih zapisa sa periferija.

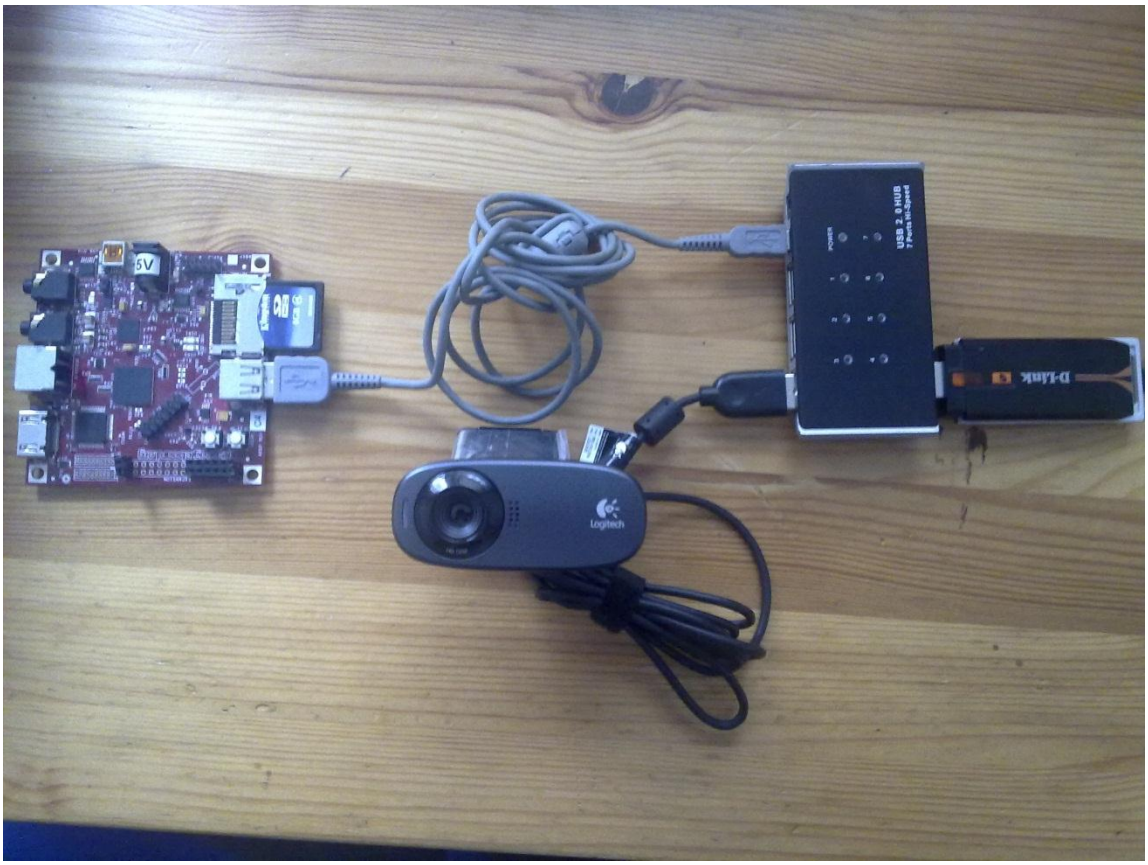


Slika 7: blok shema beagleboard sustava

Na konfiguraciju BeagleBoard pločice koja dolazi sa revizijom četiri u zbog potreba diplomskog rada bilo je potrebno iskoristiti priključak za nadogradnju (eng. *expansion slot*)

kako bi se na njega mogla postaviti pločica za upravljanje teleskopom te je trenutna konfiguracija beagleboard sustava prikazana slikom 8. Detalji o pločici za upravljanje nalaze se u zasebnom poglavlju.

Na pločicu se spajaju priključak za napajanje, HDMI kabel koji se spaja na monitor visoke rezolucije te USB replikator (eng. *hub*). Na USB replikator spojeni su miš, tipkovnica, mrežni ethernet adapter te Logitech C305 web kamera. Spoj je prikazan slikom 8.



Slika 8: modificirana Beagleboard pločica spojena na USB hub

U Beagleboard potrebno je utaknuti SD (eng. *secure digital*) karticu koja sadržava operacijski sustav. Na Beagleboard je moguće staviti niz operacijskih sustava i više različitih platformi. Istaknutije su Embedded linux, Angstrom linux, Ubuntu linux, Android i Microsoft Windows Mobile. U radu korišten je Ubuntu operacijski sustav, inačice 10.04.4 Lucid Lynx sa jezgrom 2.6.48. SD karticu potrebno je pripremiti prema uputama [2]. Nakon što je SD kartica pripravljena moguće je upaliti pločicu i pokrenuti program napisan u programskom jeziku Matlab. S novijom inačicom matlab-a 2012-a dolazi mogućnost automatskog instaliranja operacijskog sustava Ubuntu. Zbog podešavanja Beagleboardovih memorijskih sustava instalacija pomoću matlaba zahtijevala je izradu serijskog adaptera. Upotreba adaptera potrebna je samo pri prvom pokretanju

operacijskog sustava, tome je riječ u poglavljima o samom implementiranom algoritmu.

Na operacijski sustav dodani su upravljački programi za web kameru kao i programi za upravljanje postavkama putem komande linije. Slijedi niz naredbi kako instalirati program za upravljanje postavkama „v4l2-ctl“:

```
sudo add-apt-repository ppa:libv4l/ppa
sudo apt-get update
sudo apt-get install v4l-utils
```

Beagleboard pločica mora moći pristupiti Internetu, pa ukoliko navedene naredbe ne rade provjerite postavke mrežnog adaptera naredbom *„ifconfig“.*

Kako bi se Beagleboard sustav uspješno bežičnom mrežom spojio na računalo na kojem je instaliran Matlab, potrebno je podesiti takozvano ad-hoc spajanje između računala i sustava Beagleboard.

Koraci za to su sljedeći:

- u izborniku mreža na Ubuntu operacijskom sustavu kojem se pristupa desnim klikom miša na ikonu računala potrebno je odabrati *„Create a new network“,*
- Zatim je potrebno upisati ime mreže i vrstu zaštite pristupa. Zbog jednostavnosti povezivanja odabire se bez zaštite *none,*
- Nakon klika mišem na *Connect* na sustavu Beagleboard pojavit će se nova mreža pod nazivom koji smo zadali,
- Na *host* računalu u listi mreža potrebno je odabrati prethodno podešenu mrežu,
- Isto tako je potrebno postaviti ručno dodjeljivanje IPv4 adresa kako bi se Matlab mogao protokolom SSH (*eng. secure shell)* spojiti na sustav Beagleboard,
- IPv4 adrese u konkretnom slučaju su:
    - Beagleboard: 192.168.1.7/24

- *host* računalo: 192.168.1.15/24.

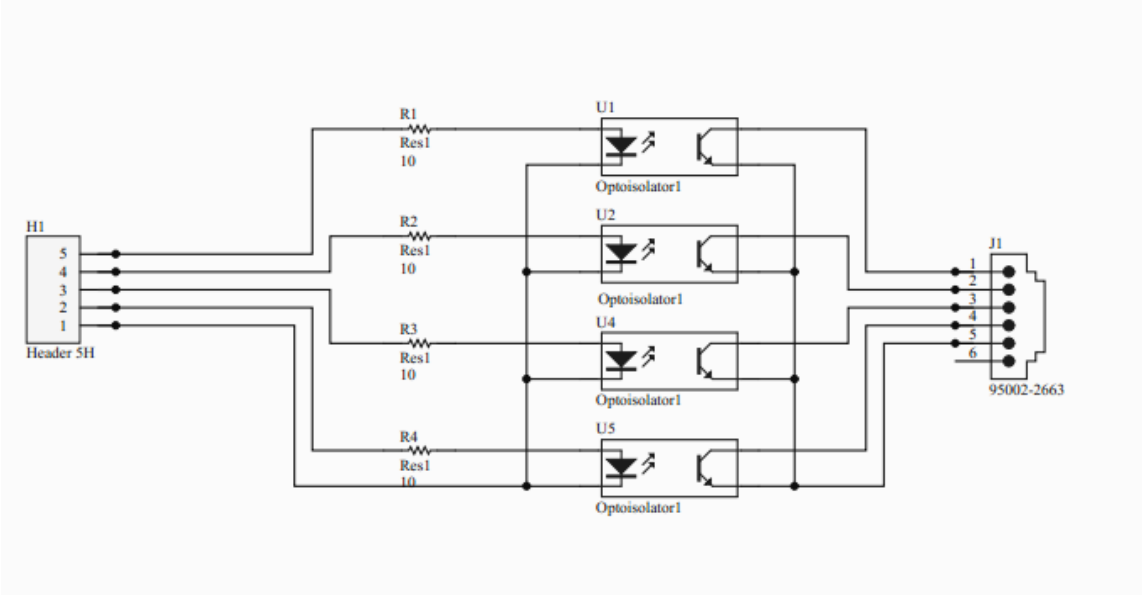
Napomena: ponekad je na Beagleboard sustavu potrebno ručno postaviti IPv4 adresu mreže. To se izvodi sljedećom naredbom:

```
sudo ifconfig nestat sh 192.168.1.7
```

2.4.3. Pločica za upravljanje i testno sklopovlje

Pločica za upravljanje teleskopom priključak je na standardni ulaz teleskopa koji su napravljeni u obliku RJ12 utičnice. Raspored funkcija svih pinova je sljedeći :

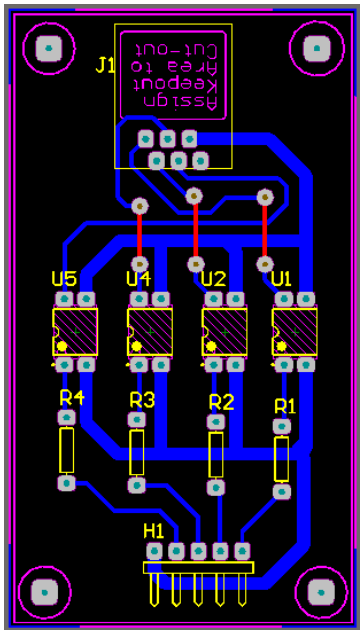
1. Pomak RA negativno,
2. pomak Dec negativno,
3. pomak Dec pozitivno,
4. pomak RA pozitivno,
5. masa,
6. nije spojen.



Slika 10: shema spoja

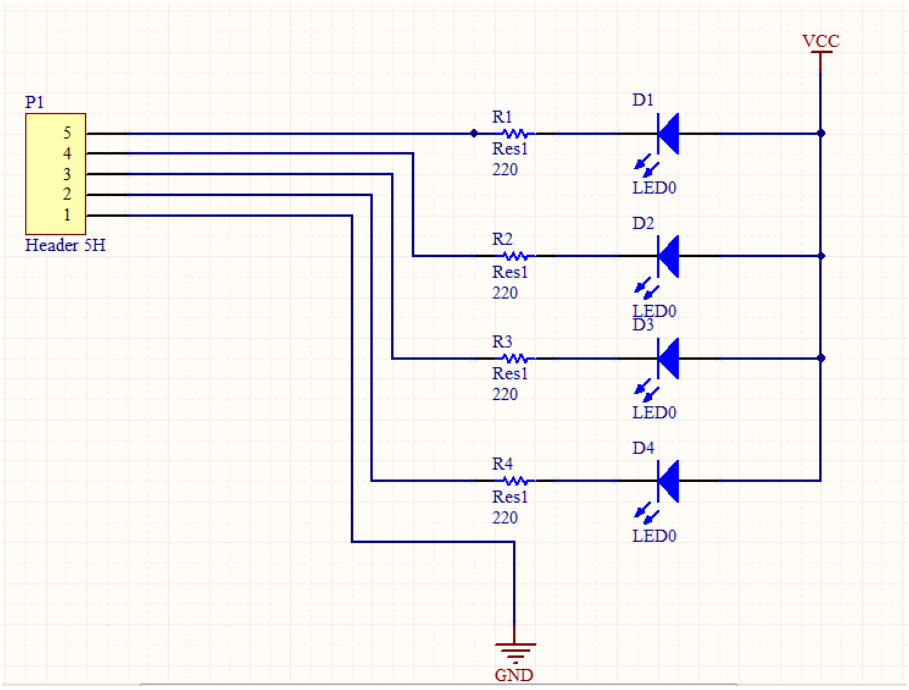
Prema zadanom rasporedu priključaka utičnice, u programu Altium Designer 2009, izrađena je tiskana pločica, prikazana na slikama 10 i 11, koja naponskim razinama s GPIO (eng. *general purpose input output*) priključnica razvojne pločice Beagleboard kratko spaja spojnice za upravljanje teleskopom. Korišteni su svjetlom okidani tranzistori kako bi se električki odvojio Beagleboard od teleskopa i time spriječilo moguće širenje kvarova i smetnji s jednog sustava na drugi. Korišteni optički prekidači tvrtke CEL, sa sljedećim karakteristikama:

- Visoki izolacijski napon (BV = 5000Vr.m.S)
- Visoki C-E napon(U=80V)
- Velika brzina okidanja (t_r=3μs, t_f=5 μs)



Slika 11: položajna shema tiskane pločice

Za testiranje napravljena je indikatorska shema sa 4 LED diode postavljene na zajedničku katodu. Ukoliko je pokrenut bilo koji od smjerova motora teleskopa pojavit će se svjetlo na jednoj od LED dioda. Shema je prikazana slikom 12.



Slika 12: shema testnog spoja

3. Teorijske pretpostavke

3.1. Uvod

Algoritam za praćenje zvijezda implementiran u ovom radu autonomno izvodi prepoznavanje zvijezda unutar polja pogleda teleskopa odnosno web kamere postavljene na pomoćni okular teleskopa. Na temelju prepoznatih zvijezda obavlja se korekcija orijentacije teleskopa u odnosu na nebesku sferu. Uspješnost algoritma uvelike ovisi o CCD odnosno CMOS senzoru i intenzitetu svjetlosti koje upada na senzor kao i o pozadinskim smetnjama uzrokovanim drugim izvorima svjetlosti ili lošim stanjem atmosfere. Detaljan matematički pogled na problematiku bit će izjašnjen u ovom poglavlju.

Kao što je navedeno, astronomska slika je zapravo digitalno izmjeren tok fotona integriran po vremenu t. Ukupan broj fotona koji pobuđuju senzor dan je integralom:

$$p_{x,y} = \int_0^t I(x,y)\,dt \tag{3}$$

Kvaliteta fotografija zvijezda ovisi o broju fotona koji pristignu sa zvijezda na senzor. Postizanje zadovoljavajućeg broja fotona ostvaruje se ili dobrom optikom ili dugom ekspozicijom kamere. Prosječan broj zvijezda unutar vidnog polja kamere računa se prema intenzitetu svjetlosti koje zvijezde koje promatramo emitiraju. Većina zvijezda vidljiva amaterskim teleskopima imaju površinsku temperaturu približno jednaku Sunčevoj [3] te se s toga računa spektralna osjetljivost za G2 spektralnu klasu zvijezda.

Po formuli za zračenje crnog tijela imamo iznos (2.96* 10^{14} W) izračenog zračenja po mm kvadratnom:

$$I(x,T) = \frac{2 \cdot \Pi \cdot n \cdot c^2}{x^5\left(e^{n \cdot C / x k_B T} - 1\right)} \quad (4)$$

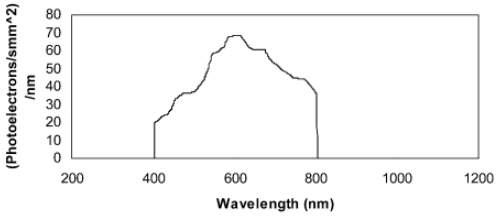
gdje je h = 6.626*10^{-34}Js, c brzina2.997*10^{8}m/s, k_b=1.38* 10^{-23} J/K. λ valna duljina i T temperatura u Kelvinima.

Kako su kod teleskopa minimalizirane refleksije, kromatska izobličenja te filtrirani dijelovi spektra koji se raspršuju unutar fokusne ravnine (crvena svijetlost). Pretpostavljeno je da je spektar svjetla koji upada u CMOS senzor kamere 400-800 nm. Iz formule 2 dobiva

se energija fotona određene valne duljine:

$$E = \frac{h \cdot c}{\lambda} \quad (5)$$

Možemo izračunati da je primjerice za teleskop koji ima zasun leće 3 cm, a kamera ekspoziciju od 200 ms, dobit će se 69235 fotona u jednoj fotografiji ukoliko je teleskop postavljen u idealnim uvjetima bez ikakvih smetnji. U žarištu gdje su fotoni fokusirani samo dio njih će senzor pretvoriti u električni signal odnosno u broj elektrona koji su izbačeni iz svojih ljusaka fotonskom ekscitacijom. Dio fotona koji ekscitiraju tzv. fotoelektrone naziva se apsolutna kvantna učinkovitost.



Slika 13: distribucija ekscitiranih fotoelektrona

Na slici X vidimo eksperimentalno dobivene [4] ekscitirane fotoelektrone distribuirane po valnim duljinama što bi značilo da kvaliteta detekcije zvijezde ovisi o valnoj duljini svjetla koju ona emitira.

3.2. Model zvjezdane slike na CMOS senzoru

Na CMOS senzor upada određena količina svjetla fokusirana pomoću optike. Tijekom hvatanja slike, fotoosjetljiva rešetka dijeli sliku na osnovne elemente slike, piksele. Slika se na pikselima formira pod određenim kutom. Slika se formira na visini *h* od optičke osi:

$$h = F * tan\theta \tag{6}$$

gdje je Ɵ kutna udaljenost od optičke osi, a F žarišna udaljenost. S toga senzor mora biti dobro postavljen na optiku. Pretpostavljajući da je središte senzora na optičkoj osi teleskopa tada bi veličina senzora trebala biti barem 2*h* odnosno dvostruko veća od visine slike, a tada senzor obuhvaća pogled dan formulom:

$$\sigma_{fw} = 2\arctan\left(\frac{d}{2F}\right) \text{[radian]} \tag{7}$$

za teleskop sa žarišnom duljinom F. Istu formulu možemo koristiti za izračun kutne veličine fotodetekcijskog središta piksela. Uzevši da je jedan piksel 17μm širok i 19μm visok može se izračunati da je kutno pokrivanje jednog piksela 0.000974° odnosno 3,5 kutnih sekundi.

Ovisno o broju piksela i fizičke širine detektora može se pronaći veličina piksela tako da se dvije veličine podijele :

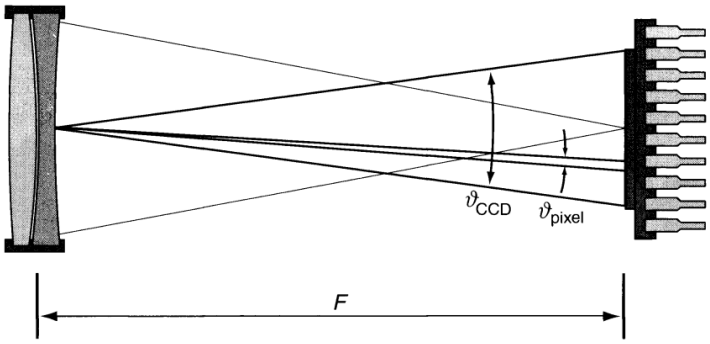
$$d_{piksel} = \frac{d_{CMOS}}{N} \tag{8}$$

Kako bi senzor kamere mogao razlučiti sliku, njegova veličina uzorkovanja mora biti dovoljno mala kako bi uhvatila funkciju točkastog raspršenja (eng. *point spread function*) svjetlosti koje prolazi kroz teleskop (idealno bi to bio zrakasti disk), karakteristične dimenzije za najmanjeg detalja na teleskopskoj slici. Prema Shannon-Nyquistovom teoremu frekvencija uzorkovanja mora biti barem dva puta veća od frekvencije signala kako bi rekonstrukcija bez aliasinga bila moguća. Primijenjeno na uzorkovanje slike, teorem govori kako veličina piksela mora biti najmanje pola promjera difrakcijskog (zrakastog) diska. Inače bi fine

strukture bile izgubljene. Slike koje su uzrokovane sa više od dva piksela po jezgri difrakcijskog diska sadrže redundantne informacije i osiguravaju manje šuma uzorkovanja. Veličina jezgre difrakcijskog diska koji ne mora nužno biti zrakastog oblika računa se pomoću sljedeće formule:

$$d_{ad} = 1.02\lambda\frac{F}{A} \tag{9}$$

gdje je A veličina otvora teleskopa, F žarišna daljina, a λ valna duljina svjetlosti koje uzrokuje difrakcijsku pojavu. Iz dane formule može se izračunati koja je minimalna veličina objekata koju dani seznor može razlučiti.



Slika 14: kut formiranja slike

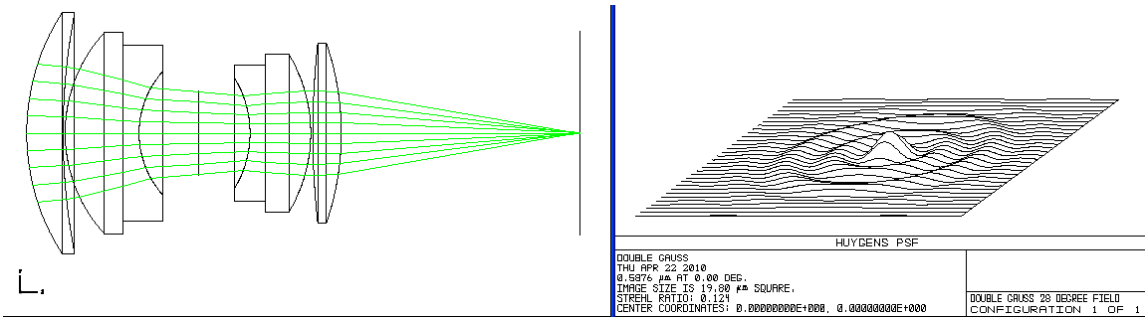
Kako bi algoritam uspješno detektirao razlučive zvijezde veće svjetline, potrebno je odrediti prag detekcije koji izravno ovisi o spomenutoj razlučivosti senzora. Kako se žarište slike sastoji od otprilike 10^6 piksela prema izvoru [4] optimalno je uzeti prag detekcije signala da bude median vrijednosti pozadinskih piksela + 5 puta standardne devijacije kako bi se izbjegla lažna detekcija. Zvijezda se detektira ako je najsvjetliji piksel iznad praga detekcije.

Prema izvoru [4] najsvjetliji piksel ovisi o obliku difrakcijskog diska kao i položaju zvijezde naprema drugim zvijezdama. Ako se primjerice nalazi uz sjajniju zvijezdu tada je moguće da se zvijezda iznad praga neće detektirati. Ukoliko zvijezda ima Gaussov PSF radijusa 0.5 piksela tada će 30 % signala biti sadržano unutar jednog piksela. No ako je radius Gaussovog PSF-a, slika 15,

jedan piksel, najsvjetliji će piksel imati samo 12.7% ukupnog signala. Prag je s toga dan formulom:

$$p = A_{piksel} + 5\sigma_{\text{piksel}} \frac{1}{\iint_0^1 \frac{1}{2\pi\sigma_{psf}} exp\left(\frac{-x^2+y^2}{2\sigma_{p_{sF}}}\right) dxdy} \tag{10}$$

Gdje je A srednja vrijednost piksela, σ_{piksel} standardna devijacija vrijednosti piksela u mračnom kadru, a σ_{psf} radijus PS funkcije dane u pikselima uz pretpostavljenu Gaussovu zvonoliku distribuciju.



Slika 15: Gaussova distribucija PSF-a

Prosječan broj zvijezda u vidnom polju uz minimalne smetnje, izvedene su iz zvjezdanog kataloga sa određenim magnitudama M [4].

$$N = 6.57 * \exp(1.08M) * \frac{1-\cos(\frac{A}{2})}{2} \tag{11}$$

U prosjeku detektor zvijezda prema katalogu [4] bit će 6.2 zvijezde.

3.3. Neželjeni signali na fotografiji -šum

Zvjezdane slike na CMOS ili CCD senzorima kao i svi signali i informacije imaju neželjene dijelove odnosno smetnje. Smetnje mogu biti uzrokovane vanjskim ili unutarnjim izvorima. Vanjski izvori su nepovoljni atmosferski uvjeti, vibracije montaže teleskopa, svjetlosno zagađenje, onečišćenje optike i sl. Unutarnji izvori, su tamna struja (eng. *dark current*), šum uzorkovanja, elektromagnetske

smetnje i sl. Kvaliteta svakog signala opisana je omjerom signal šum (eng. SNR-*sound to noise ratio*). Za broj fotona koji upada na žarište SNR se može izraziti formulom:

$$SNR = \frac{\bar{x}}{\sqrt{\bar{x}}}, \bar{x} = \frac{1}{n}\Sigma x_i \tag{12}$$

Gdje je u brojniku signal, a u nazivniku šum. Statistička nesigurnost u broju fotona, σ, naziva se šum sačme (eng. *shot* noise).

Dodatno, osim šuma detekcije fotona, CCD i CMOS uređaji generiraju neželjeni signal koji se naziva tamno strujanje (eng. *dark current*). Tamno strujanje nije nasumičan šum kao što je bio pri hvatanju fotona. Nastaje zbog nepravilnosti kristalne rešetke koja generira višak elektrona i statistički se ponaša po Poissonovoj razdiobi.

Kao nedostatak senzora unosi se i šum očitavanja (eng. *readout noise)*. Javlja se kao nasumična varijacija unutar pojačala na izlazima senzora. Nije ovisna o razini signala.

Unutar kamere javlja se i kvantizacijski šum koji se javlja pri konverziji analognog signala u digitalni.

Većina navedenih šumova uklanja se softverskim filtrima koji ne propuštaju pojaseve frekvencije u kojima se nalazi šum, koji su integrirani u kameru. Tamno strujanje se uklanja slikanjem na zatvorenu blendu te filtriranjem superpozicijom dva signala. Ako sliku propustimo bez filtara, u RAW formatu ukupan šum je jednak:

$$\sigma_{Raw} = \frac{1}{y}\sqrt{\sigma^2 + \sigma_d^2 + \sigma_{RON}^2} \tag{13}$$

Gdje je σ_{raw} ukupan šum, σ šum sačme, σ_d šum tamnog strujanja, σ_{RON}šum očitavanja slike.

Osim unutarnjih grešaka uzrokovanih nesavršenostima opreme za slikanje astronomskih slika, kao što smo naveli imamo i vanjske izvore šuma. Primaran izvor šuma kod astronomskih fotografija su atmosferski uvjeti. Izvor [5] navodi

omjer sličan SNR-u, a odnosi se na broj fotona pristiglih s promatranog objekta naprema broju fotona s ostalih izvora na nebu:

$$OSR = \frac{X_{objekt}}{X_{nebo}} \tag{14}$$

Ova veličina može varirati od 1000 za vrlo svijetle objekte do 0.001 za vrlo slabe. Za mračno ruralno nebo pretpostavlja se signal od 40 elektrona po pikselu unutar ekspozicije od 60 sekundi, OSR iznosi 6,3 za promatrani objekt. Za sub-urbano nebo očitava se 600 elektrona po pikselu, a OSR iznosi 0.42 dok u svjetlom urbanom nebu imamo 4000 elektrona po pikselu, a isti objekt ima OSR 0.06. Iako je urbano nebo 100 puta svjetlije od ruralnog, detekcija je ipak uspješna no puno manje smetnji ima ako se fotografije uzimaju s mjesta gdje atmosfera nije svjetlosno zagađena.

Drugi vanjski izvor šuma su dinamični uvjeti slikanja fotografija. Odnosno pomak teleskopa pri vođenju. Ni jedan sustav za vođenje nije savršeno izrađen s toga se javljaju vibracije koje uzrokuju razmazivanje slika zvijezda, a samim time onemogućuje pravilnu detekciju centroida zvijezde koji se više ne nalazi u jednoj točci nego je njegova energija distribuirana kroz više piksela. Takve zvijezde ponekad ne mogu proći prag detekcije te ih se ne klasificira kao zvijezde. Ovakva vrsta smetnje rješava se pomoću filtriranja. Filtar se sastoji od Malletovog algoritma za izračun koeficijenata valićne transformacije, izračuna varijance šuma, i obrada parametara skale, praga i invertiranja više-skalne dekompozicije kako bi se rekonstruirala originalna slika bez pomaka. Prema izvoru [6] ovaj postupak donosi značajne rezultate no isto tako ovisi o pozadinskom osvjetljenju.

3.4. Vođenje teleskopa

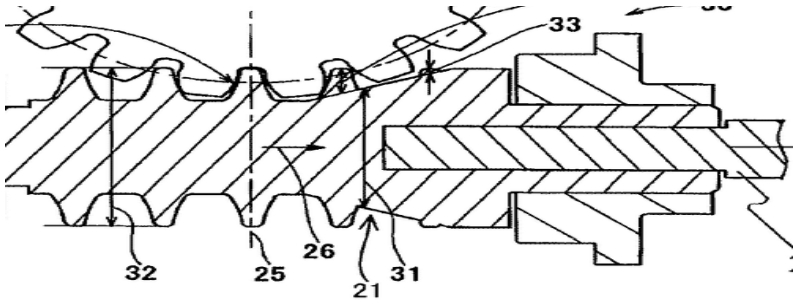
Kaže se da je teleskop jednako dobar kao i njegova montaža. Ukoliko je montaža defektivna, javljaju se pogreške u praćenju te uzrokuju razmazane fotografije. Tada je nužno uvesti automatsku korekciju praćenja koju će se ostvariti ovim radom.

Pogreške pri vođenju mogu rezultirati izduženim ili povećanim slikama zvijezda. Možemo ih podijeliti u tri kategorije: pogreške okretaja, periodične pogreške i nasumične pogreške. Pogreške okretaja uzrokovane su nejednolikom brzinom okretaja elektromotora na montaži teleskopa, periodičke pogreške javljaju se zbog nesavršenog poklapanja zupčanika i pužnog mehanizma i nasumične pogreške koje se javljaju u slučaju kvara montaže ili trešnje teleskopa uzrokovane vanjskim faktorima (slučajno zapinjanje).

- pogreške okretanja –ispravljaju se namještanjem brzine motora kako bi dobro pratio pokretanje nebeskog svoda. Ako se ne radi o računalno vođenim montažama sa sinkronim motorima moguće je dodati ispravljače pogona koji se pokreću sa 12V istosmjerne ili 220V izmjeničnog napona. Montaže koje su pokretane manjim DC pulsnim motorima ili koračnim (eng. *step*) motorima omogućuju korisniku ručno prilagođavanje brzine putem daljinskog upravljača. Obično ponuđene brzine su lunarna, solarna, sidereal ili varijabilna brzina. Ukoliko sidereal brzina (brzina astronomskog okretanja) nije dobro kalibrirana biti će potrebna ručna korekcija brzine.
- periodičke pogreške –znače da je prosječna brzina pogona ispravna ali pogon naizmjenično radi brzo i polako. Ukoliko su zupčanici i prijenos nesavršeno konstruirani, što je vrlo česta pojava u komercijalnih teleskopa, tada će se javiti ova vrsta pogreške. Moguće pogreške konstrukcije su:
    - Zupčanik motora nije u ravnini sa pužnim prijenosom,
    - pužni prijenos nije centriran na svojoj osovini,
    - pužni prijenos je svinut.

Po definiciji periodička pogreška se ponavlja u kvazisinusoidalnom ili sinusoidalnom uzorku. Period ciklusa je gotovo uvijek jednak vremenu koliko je potrebno da se pužni prijenos potpuno izvrti na zupčaniku montaže. Obično je to vrijeme od 8 minuta. Za ispravljanje ovakvih grešaka na montažama se obično primjenjuje sustav za uklanjanje periodičke pogreške (eng. PEC *periodic error correction*). Sustav adaptivno kroz 8 minuta vođenja nauči karakteristiku pužnog prijenosa te PID regulatorom ispravlja pogrešku. Ukoliko amplituda ove pogreške nije reda veličine tri ili četiri puta veća od osjetljivosti kamere tada se ovakve pogreške mogu zanemariti.

- Nasumične pogreške –javljaju se ukoliko pužni mehanizam nije dovoljno prislonjen na zupčanike montaže tada se javlja preskakanje mehanizma, ili se javlja zbog nagle promjene smjera vođenja kada se zbog inercije i velike mase teleskopa naprema zupčaniku, zupčanik nađe u praznom hodu odvojen od bilo koje dvije lopatice pužnog mehanizma slika 16.



Slika 16 prijanjanje pužnog mehanizma

Nakon što je uhvaćena jedna ili niz fotografija sa ispravljenim pogreškama navedenim u ovom i prethodna dva poglavlja moguće je vođenje teleskopa praćenjem objekta koji se nalazi u sredini vidnog polja. Računalom opremljene montaže imaju opciju automatskog praćenja pogleda. Postoji niz komercijalnih proizvoda koji obavljaju ovaj zadatak pomoću ugradbenog računala. Jedan takav primjer je Celestron NextGuide. Ima sljedeće specifikacije:

- Ekspozicija: 1,2,4,8,16,32,64,128,256,512,1024,2048,4096 ms
- Veličina senzora:5,59mm*4,68 mm, Sony CCD

- rezolucija:510*492 px
- napajanje 6-14V DC, 250mA
- mogućnost vođenja alt-azimut montaže i ekvatorijalne montaže,
- idealna žarišna daljina 400-1200mm

Automatski pratitelj pogleda ima nekoliko koraka rada:

- namještanje fokusa u kojem se slika izoštrava na najvišu moguću razinu
- zaključavanje zvijezde –traženje zvijezda i potom zaključavanje
- automatsko vođenje –praćenje označene zvijezde nebom.

Jednom kad je zvijezda detektirana i zaključana u sustavu za automatsko praćenje izvodi se transformacija koordinatnih sustava zvijezde i koordinatnih sustava kamere, a potom se može izvodi astrometrija i fotometrija odnosno uspoređivanje trenutnog vidnog polja teleskopa sa bazom podataka koja sadržava više od 40000 nebeskih objekata.

3.1. Koordinatni sustavi i preslikavanja

Jednom kad su zvijezde uslikane potrebno ih je smjestiti i pravi koordinatni sustav. U astronomiji postoji više nego nekoliko koordinatnih sustava prema kojima se obavlja navođenje. Prvi počeci astrometrije sežu do Ptolomejeva doba kad je načinjen prvi zabilježeni katalog zvijezda. Današnji katalozi zvijezda sadrže i do 100000 nebeskih objekata te imaju zabilježene parametre poput RA, Dec, skale, magnitude i druge. Preciznost astrometrijskog mjerenja, a samim time i nalaženja odgovarajućeg sustava u bazi zvijezda ovisi o preciznosti referentnog okvira unutar kojeg je izmjeren položaj nebeskih objekata. Kod amaterskih teleskopa sa CCD kamerom ta preciznost iznosi 0.2 kutne sekunde i bolje.

Kada se promatra samo dio neba putem teleskopa čini se da se promatra ravna ploča. Međutim to nije točno, jer se promatra dio nebeske sfere čija je

zakrivljenost toliko velika da se može smatrati da te zakrivljenosti nema, odnosno, matematičkim rječnikom uzima se tangencijalna ravnina okomita na normalu koja prolazi kroz središte optičke osi teleskopa.

Standardni koordinatni sustav uzimaju prethodno spomenuti koncept te se vidno polje teleskopa definira kao tangentu na nebesku sferu u točci $T(\alpha_{0,}\delta_0)$. X os poravnata je sa desnim uzdizanjem α, a Y os sa deklinacijom δ. Točka $T(\alpha_{0,}\delta_0)$ smatra se ishodištem koordinatnog sustava. Za dani objekt X i Y koordinate računaju se po sljedećim formulama:

$$x = \frac{\cos\delta \sin(\alpha - \alpha_U)}{\cos\delta_0 \cos\delta cos(\alpha - \alpha_0) + \sin\delta_0 \sin\delta} \tag{15}$$

$$Y = \frac{sin\delta_0 \cos\delta \cos(\alpha - \alpha_U) - cos\delta_0 sin\delta}{\cos\delta_0 \cos\delta cos(\alpha - \alpha_0) + \sin\delta_0 \sin\delta} \tag{16}$$

Desno uzdizanje i deklinacija mogu se računati po inverznim formulama:

$$\alpha = \alpha_0 + \arctan(\frac{X}{cos\delta_0 - Y sin\delta_0}) \tag{17}$$

$$\delta = \arcsin(\frac{\sin\delta_0 + Y cos\delta_0}{\sqrt{1 + X^2 + Y^2}}) \tag{18}$$

Slika zvijezda koja upada u žarište teleskopa odnosno u senzor ravna je površina umjesto zakrivljena sferna površina te je linearna udaljenost r jednaka iznosu žarišne daljine i tangensa kutne udaljenosti. Za jako male kutne udaljenosti vrijedi tanβ=β. Većina današnjih teleskopa ima dovoljnu optičku korekciju kako bi se prethodna aproksimacija bila točna za sve slučajeve.

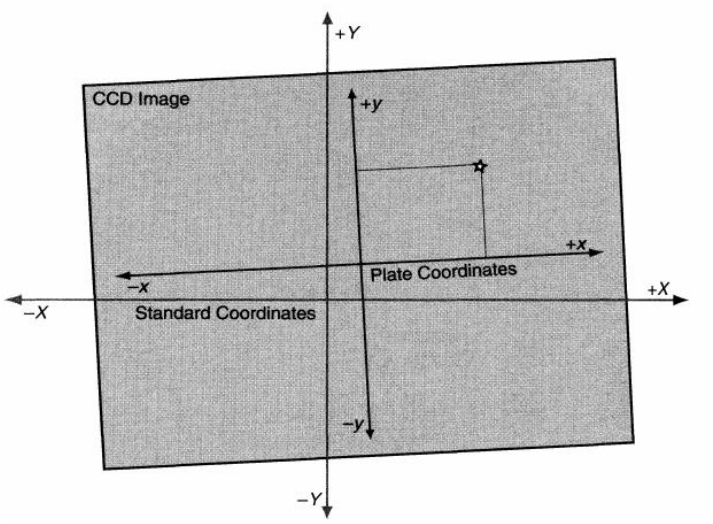
Uslikavanje na kameri, nikad nije savršeno poravnato sa optičkom osi teleskopa te se centar $T(\alpha_{0,}\delta_0)$ se ne može precizno odrediti, a i slika je većinom zakrenuta za određen broj stupnjeva. Ukoliko je središte slike pomaknuto od

optičkog središta za neki uređeni par x, y te ukoliko je pomaknut za kut ρ tada je odnos među koordinatama uslikanog sustava i centriranog sustava:

$$X = \frac{cos\rho}{\mathrm{F}}x - \frac{sin\rho}{\mathrm{F}}y - \frac{x_{odmak}}{F} \tag{19}$$

$$X = \frac{\sin\rho}{\mathrm{F}}x - \frac{cos\rho}{\mathrm{F}}y - \frac{y_{odmak}}{F} \tag{20}$$

Iako gotovo zanemarivo, senzor može biti nagnut u odnosu na optičku os teleskopa, no proizvođači opreme često empirijski proračunate parametre za kompenzaciju nagnutosti i zakrenutosti. Kako bi se ručno izračunali ove parametri koriste se standardne koordinate triju zvijezda koje daju tri parametarske linearne jednadžbe za izračun konstanata ravnine zakreta (eng. *plate constants*).



Linearne jednadžbe glase :

$$X_1 = ax_1 + by_1 + c \tag{21}$$

$$X_2 = ax_2 + by_2 + c \tag{22}$$

$$X_3 = ax_3 + by_3 + c \tag{23}$$

Standardnim postupkom računaju se tri nepoznanice, a, b i c. Nakon što su navedene konstante proračunate pozicija željenog objekta mogu se izračunati iz ravninskih (plate) koordinata. Prema formulama za α i δ računaju se desno uzdizanje i deklinacija za praćeni objekt.

4. Algoritam

Programsko rješenje problema automatskog praćenja zvijezda pisano je u programskom jeziku Matlab. Kao razvojno okruženje koristi se Matlab 2012a i programski paket Simulink koji dolazi s istom inačicom Matlaba. Riječ je o integriranom rješenju koje dozvoljava tekstualni i blokovski razvoj matematičkih i multimedijskih programa.

Algoritam se odvija u dva stupnja. Prvi stupanj služi za automatsku kalibraciju montaže dok drugi služi za centriranje zvijezde u sredinu kadra odnosno praćenje. Detaljan opis stupnjeva slijedi u narednim poglavljima.

4.1. Prvi stupanj: kalibracija

Kalibracija se odvija u nekoliko koraka, prva četiri koraka su zajednička za svaki algoritam pa ih se opisuje samo jednom:

1. Uslikavanje - (eng. *frame grabbing*) prvi korak algoritma, zajednički je za prvi i drugi stupanj algoritma. Programiran je u Simulink okruženju, te se može izvoditi iz tri različita izvora, ovisno o postavljenoj razini provjeravanja koda na greške (eng. *debugging*). Izvor uslikavanja mogu biti nizovi slikovnih datoteka koje su pohranjene na računalu, unutar bloka „From Multimedia File“, drugi izvor uslikavanja je web kamera priključena na prijenosnik ili osobno računalo. Drugi izvor se koristi pri naprednom uhodavanju algoritma. Treći izvor je web kamera spojena na razvojni sustav Beagleboard ,
2. prilagođavanje slike i konverzija – pomoću Simulink bloka „Color Space Conversion“ obavlja se pretvorba iz RGB (eng. *red green blue*) formata slike u crno-bijeli format. Potom slika prelazi iz jednobajtnog cjelobrojnog tipa podataka u dvostruku preciznost kako bi se izbjegla zasićenja pri manipulacijom slikovnim datotekama. Posljednji korak prilagođavanja slike je

prolazak slike kroz pojasno-propusni filter koji uklanja zvijezde koje se nejasno vide i sprečava lažnu detekciju,

3. spremanje slike u cirkularni spremnik izvodi se korištenjem permanentne memorije koja glumi FIFO (eng. *first-in-first-out*) spremnik. Nakon što se uslika dovoljno slika, ukoliko je tako postavljeno spremnik se briše odnosno puni s nulama.
4. izračun potrebnog broja točaka pomaka slike izražen u pikselima odnosno koliko je dugo potrebno držati komandu u aktivnom stanju da bi se postigao željeni pomak,
5. izračun centra svih zvijezda na trenutnoj slici, algoritam je zajednički za oba stupnja uz minorne preinake, jer se u ovom stupnju unutar strukture podataka za koordinate centra bilježi i pomak i smjer centra u pikselima.
6. komparacija dvaju slika- ukoliko se stupanj za kalibraciju pokrenuo više od jednom uspoređuju se slike posebno oblikovanim algoritmom za usporedbu blokova slika (eng. *block matching algorithm*)
7. detekcija zakreta koordinatnog sustava- računa se u odnosu na sekvencu prethodnih slika dobiva se kut delta koji se koristi u drugom stupnju kako bi se ispravio zakret kamere u odnosu na ravninu vođenja teleskopa.
8. slanje sljedeće naredbe na temelju trenutnog stanja programa

4.2. Drugi stupanj: vođenje

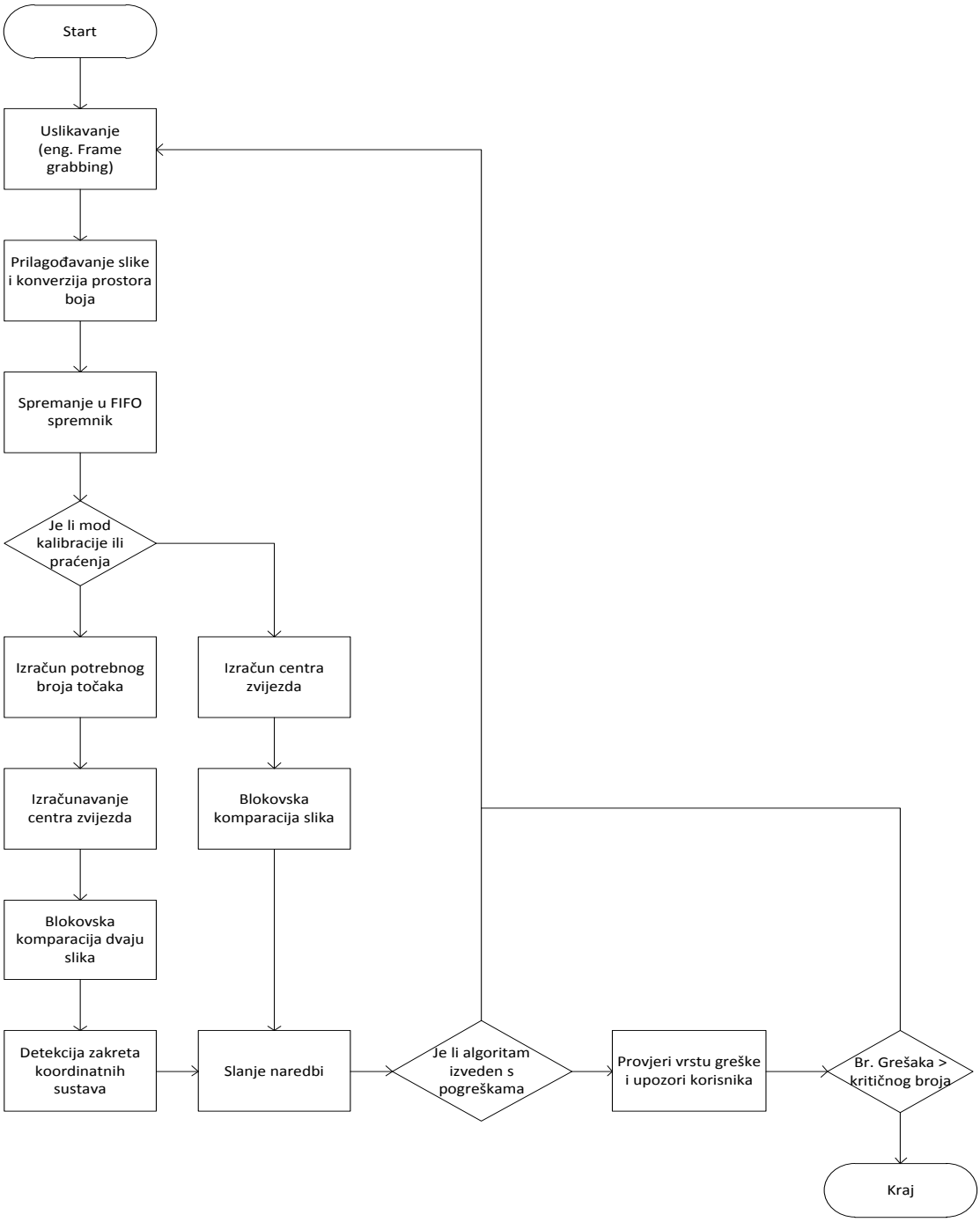
Stupanj vođenja po svojoj implementaciji je sličan stupnju za kalibraciju uz nekoliko ključnih razlika.

1. koraci od 1 – 4 zajednički su za oba stupnja algoritma
2. izračun centra zvijezda izvodi se bez posebnog spremanja koordinata
3. komparacija slika provodi se jednakim algoritmom kao i u stupnju za kalibraciju uz mogućnost pretraživanja unutar više slika odnosno vremenskih okvira

4. zadnji korak je izračunavanje sljedeće naredbe koja se treba poslati na Beagleboard pločicu. Naredba se izračunava na temelju zakreta koordinatnog sustava i trenutne pozicije zvijezde u odnosu na centar osi. Implementirana je mogućnost ispravka centriranja na temelju jedne osi, osi desnog uzdizanja, tako da po deklinacijskoj osi sustav nimalo ili samo malo klizi. Ukoliko je deklinacijska os dobro namještena kliženja ne bi trebalo biti.

Nakon dva stupnja algoritma dodan je još jedan zajednički stupanj koji sakuplja sve greške nastale pri provođenju programa, obrađuje ih i broji. Ukoliko je pređena određena količina grešaka algoritam će o tome obavijestiti korisnika, a ako je broj pogrešaka kritičan tada će se algoritam zaustaviti.

Shema izvođenja algoritma prikazana je na slici 17



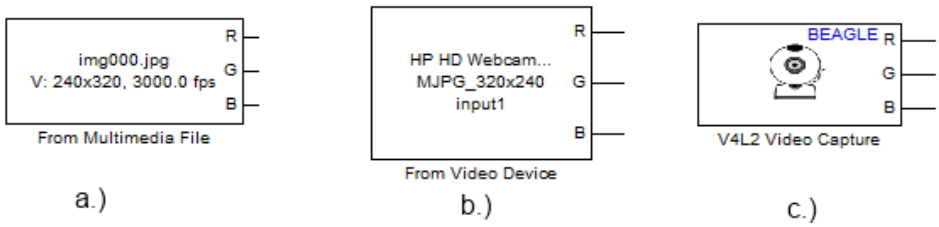
Slika 17 blokovski prikaz algoritma

U sljedećim poglavljima bit će detaljno opisan svaka komponenta algoritma, njihove posebnosti i varijable, povezanost sa sklopovljem i slično.

4.3. Uslikavanje (eng. Frame grabbing)

Uslikavanje je jedan od najvažnijih dijelova algoritma zbog toga što određuje frekvenciju izvođenja algoritma. Svaki Simulink blok od kojih je sustav za uslikavanje načinjen ima u postavkama postavljeno vrijeme uzorkovanja (eng. *sample time*) koji određuje brzinu izvođenja pojedinog elementa. Vrijeme uzorkovanja može poprimiti bilo koju pozitivnu realnu vrijednost osim nule, te -1 i beskonačno (inf). Ukoliko su postavljene dvije zdanje spomenute vrijednosti tada se vrijeme uzorkovanja nasljeđuje iz roditeljskog bloka odnosno simulinka ako je blok već roditeljski. Glavni elementi za uslikavanje, su:

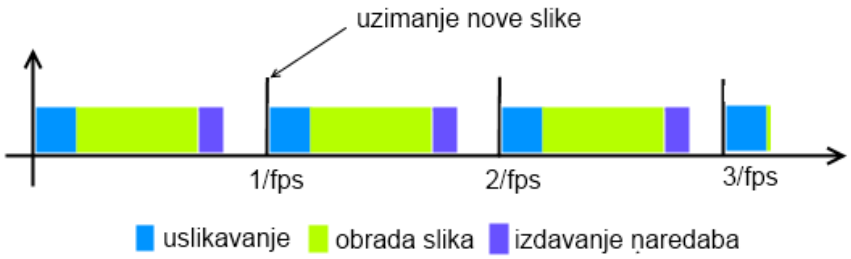
- „From Video Device“,
- „From Multimedia File“ te
- „V4L2 Video Capture“ i prikazani su slikom 18.



Slika 18: Elementi prihvata slike: a) iz multimedijske datoteke, b) web kamere na prijenosniku, c) kamere na beagleboard pločici.

Navedeni elementi ne mogu imati naslijeđeno vrijeme uzorkovanja, nego moraju imati realan broj što ne dozvoljava manualno okidanje, što bi rješenje problema diplomskog zadatka uvelike pojednostavnilo. Naime ako bi se koristio podsustav sa ručnim okidanjem (eng. *triggered subsystem*) moglo bi se povećati vrijeme obrade slike, a samim time bi se rasteretio ograničeni memorijski podsustav pločice Beagleboard. Pošto je ova mogućnost u počecima izrade rada zanemarena svim komponentama postavljeno je vrijeme uzorkovanja od 1/3000, a kontrola uslikavanja izvodi se preko upravljačkih programa za blokove „From Video Device“ i „V4L2 Video Capture“. Za „From Multimedia File“ koristi se samo

jedna slika jer taj blok služi isključivo za uhodavanje. Iz toga slijedi da se program mora izvesti, izuzev u slučaju „From Multimedia File“ unutar razmaka vremena uslikavanja. Uslikavanje je postavljeno za obje glavne komponenete na 5 okvira po sekundi (FPS eng. *frames per second)*. Izvođenje programa uključuje izvršavanje barem jednog stupnja programa, dijela za slanje naredbi i ispravljanja pogrešaka. Dijagram na slici 19 prikazuje vremenski tok izvođenja programa:



Slika 19: Vremenski tok izvođenja programa

Sav programski posao treba se obaviti u roku jednakom recipročnoj vrijednosti broja sličica u sekundi inače će doći do gubitka podataka. Simulink shema sustava za uslikavanje sadrži i elemente za transponiranje slika kako bi se svim izvorima uslikavanja prilagodila rezolucija, jer blokovi podržavaju isključivo predefinirane rezolucije slika, no neki od njih su pogrešno postavljeni.

Broj sličica u sekundi određuje se unutar upravljačkih programa, a za potrebe diplomskog rada opisat će se samo sučelje sa Beagleboard upravljačkim programima pošto je web kamera priključena na osobno računalo služila isključivo za ispitivanja i uhodavanje. Upravljački programi instalirani dodatno na sustav Beagleboard nazivaju se „V4l2-ctl“.

Naredbama koje se odašilju unutar Matlab koda prije pokretanja cijelog algoritma ili direktno na terminalu Beagleboard pločice mogu se postaviti parametri uslikavanja za web kameru. Naredbe su sljedeće:

```
v4l2-ctl -info
```

- prikazuje osnovne informacije o instaliranim uređajima

```
v4l2-ctl --list-ctrls
```

- prikazuje moguće opcije u obliku popisa s kojeg se može odabrati koje postavke želimo. Primjer ispisa naredbe je sljedeći:

```
brightness (int)   : min=0 max=255 step=1 default=128 value=128
contrast (int)   : min=0 max=255 step=1 default=128 value=128
gamma (int)   : min=1 max=6 step=1 default=4 value=4
auto_gain (bool) : default=1 value=1
```

Za upisivanje određene postavke koristi se naredba „v4l2-ctl –set-ctrl“, na primjer:

```
v4l2-ctl --set-ctrl brightness=200
v4l2-ctl --set-ctrl auto_gain=0
```

4.4. Prilagođavanje slike i konverzija prostora boja

Slika, jednom fotografirana na web kameri zapisuje se u dva moguća formata:

- YCbCr – prostor boja koji se inače koristi u video i slikovnim zapisima digitalne fotografije, cjelokupan spektar nije predstavljen nego se koristi samo dio boja koji ljudsko oko može zamijetiti
- RGB – (eng. *red green* blue) prostor boja od 16 milijuna mogućih kombinacija, sastoji se od crvene zelene i plave

Konverzija prostora boja je drugi naziv za pretvorbu iz jednog zapisa slikovnih datoteka u drugi. U ovom slučaju radi se o pretvorbi iz tro-kanalne slike sa razinama boja RGB u monokromatske razine, tj. u crno bijelu sliku. Slika se

zapisuje u obliku matrice koja je cjelobrojnog tipa bez predznaka. Nakon konverzije u crno-bijelu sliku slika mora proći kroz pojasno-propusni filtar kako bi se uklonio šum i spriječilo lažno detektiranje zvijezda. Radi se o filtru koji uklanja šum prihvata piksela koji je spomenut u teorijskom uvodu i dugovalne varijacije slike dok se čuva karakteristika veličina zvijezda. Ulazi u funkciju su sljedeći:

- Dvodimenzionalna slika koja se treba filtrirati,
- karakteristična skala buke izražena u pikselima. Aditivni šum ove veličine i manji bi trebao nestati, a određuje se empirijski ovisno o šumu koji ima okolina. Određeno je da je najbolja vrijednost za astronomske slike 5. Ukoliko se ova vrijednost postavi na nula odrađuje se samo uklanjanje pozadine,
- cjelobrojna duljina u pikselima koja je za nekoliko piksela veća od tipičnog objekta na slici. Može biti postavljena u 0, a tada se obavlja samo nisko propusna operacija zamućivanja bez pozadinskog oduzimanja definirana u l-objektu.
- prag filtracije – ukoliko nije postavljen tada se nakon konvolucije svaki negativni piksel postavlja na nulu. Ukoliko je postavljena neka pozitivna vrijednost tada se može ukloniti buka malog intenziteta i male čestice koje se mogu zanemariti. Ako je postavljen u –INF tada se ne provodi oduzimanje od praga na normalnim pikselima.

Algoritam se provodi tako da se prvo radi dvodimenzionalna konvolucija sa matricom koeficijenata (eng. *kernel*) Gaussove funkcije. Zatim se kreira nisko propusna slika tako da se konvoluira original sa pravokutnom funkcijom (eng. *boxcar*). Nakon toga se od Gaussove verzije slike oduzima nisko propusna slika nastala pravokutnom funkcijom kako bi se na kraju dobila slika propuštena i kroz nisko propusni filtar.

4.5. Detekcija zvijezde

Računanje centra zvijezde provodi se u oba stupnja algoritma, sa minornim razlikama u implementaciji. Proces izračunavanja zvijezde provodi se u nekoliko koraka. Prvi korak je izračun medijana svih piksela slike. Vrlo česta zabuna je da je medijan ukupna suma podjeljena sa brojem piksela no to nije točno. Radi se izračunu srednjeg člana sortirane liste piksela od najmanjeg prema najvećem. Potom se ugrađenom Matlabovom funkcijom računa maksimalni iznos vrijednosti piksela. Obje vrijednosti potrebne su za izračun praga detekcije zvijezde:

$$T = |M - m| * 0{,}4 \tag{24}$$

gdje je T prag, M maksimum slike i m medijan. Nakon izračunatog praga izrađuje se kopija slike koja je potrebna ukoliko postoji više od jednog objekta od interesa na slici odnosno zvijezde koja je dovoljno svijetla. Naime nakon što je zvijezda detektirana ona se s originalne slike briše kako bi se izbjegla ponovna detekcija. Prvo se detektiraju one najsvjetlije zvijezde pa sve tamnije do praga detekcije.

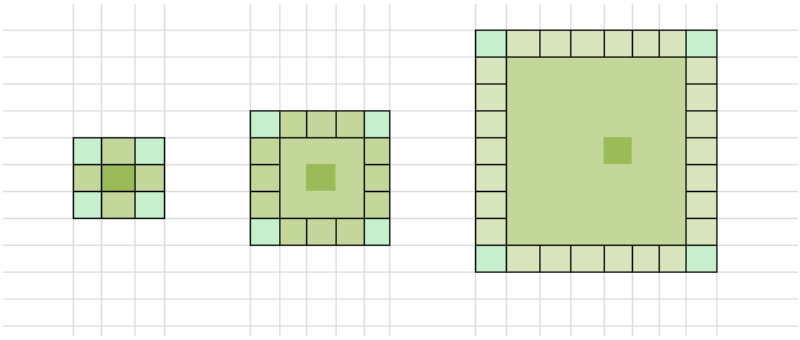
4.5.1. Pronalaženje regije interesa

U koraku algoritma koja pronalazi regiju interesa (eng. ROI – *region of interest*) koristi se ugrađena funkcija „*find*()“ koja za izlaz ima matricu koordinata koji zadovoljavaju neki logički kriterij. U algoritmu taj kriterij je da su maksimumi sve zvijezde čiji je sjaj do 70% svjetlosti najsjajnije zvijezde na slici. Jednom kad su sve sjajne zvijezde nađene, a maksimum broja zvijezda koje korisnik želi raspoznati na slici zadan je brojem N u algoritmu, koristi se funkcija „roi*()“*. Funkcija se vrti u petlji za onoliko koliko ima zvijezda, a za ulazne parametre ima sljedeće atribute:

- Slika – 2D matrica, nad kojom se obavlja pronalaženje centroida.
- x, y koordinate najsjajnije zvijezde,

- prag – prag detekcije zvijezda zaračunat formulom,
- razina ispravljanja programa- broj koji označava da li će se pomoćne funkcije za pronalaženje grešaka uključiti.

Funkcija nakon prihvata parametara postavlja zastavice detekcije za donji i gornji redak te lijevi i desni stupac na nulu. U petlji koja traje sve dok se ne prođe gotovo cijela slika, uzimaju se vektori redaka i vektori stupaca najbliži najsjajnijoj zvijezdi. U svakom se sljedećem koraku petlje vektori stupaca odnosno redaka proširuju te provjeravaju imaju li bilo koji piksel koji je veći od praga detekcije. Ukoliko imaju u sljedećem koraku se redak odnosno stupac povećava za jedan. Ukoliko je stupac odnosno redak u kojem su svi pikseli ispod praga pronađen tada se zastavica detekcije postavlja u jedinicu. Algoritam na kraju svakog koraka provjerava zastavice te ako su sve zastavice podignute tada se algoritam zaustavlja jer je pronađena kompletna regija od interesa. Ukoliko ROI nije na rubovima slike, dodaje se još jedan dodatni redak koji je ispod praga kako bi se osigurala bolja detekcija. Kada je ROI pronađen tada se isti briše sa originalne slike kako glavni algoritam ne bi ponovno tražio najsvjetlije zvijezde na već obrađenom području.



Slika 20: izračun regije interesa

Brisanje se ne obavlja punjenjem slike nulama nego, da bi se izbjegle nepravilnosti u daljnjoj detekciji, primjerice jako bliskih objekata, pronađena se regija od interesa popunjava bojom koja je 30% od praga izračunatog formulom 24.

4.5.2. Pronalaženje centra zvijezde

Nakon što je regija interesa izolirana potrebno je pronaći centar zvijezde. Kao što je u radu već spomenuto u savršeno fokusiranoj slici zvijezde su točkasti izvori svjetlosti, a zrakasti disk pada unutar jednog ili dva piksela. Gdje je vrh od spomenutih 27% energije, odnosno središte pozicionirano negdje unutar samog piksela. Ukoliko je optika imalo defokusirana, što je čest slučaj pri automatskoj detekciji s CCD kamerom zbog jeftine konstrukcije, zvijezda će zauzimati i nekoliko piksela, a pravi vrh će biti razmazan negdje u prostoru između njih. U ovom radu koristi se formula koja omogućuje izračun centra zvijezde na subpikselnoj razini. Ukoliko je zvijezda defokusirana tada se centar može i preciznije računati, jer ukoliko nije tada se izračun ograničava teoremom uzorkovanja. Riječ je o preciznosti koja je ograničena na samo 0.38 piksela po formuli:

$$\iint_{-0,5}^{0.5} \sqrt{X^2 + y^2}\,dx\,dy = 0.3825 \tag{26}$$

Jednom kad je pronađena regija interesa i pretpostavljajući da sadrži zvijezdu potrebno je izračunati srednju vrijednost piksela unutar ROI-a prema formuli:

$$B = \sum_{x=ROImin}^{ROImax} \sum_{y=ROImin}^{ROImax} vpiksel(x,y) \tag{27}$$

Koordinate centra računaju se po sljedećim formulama:

$$x_{centar} = \sum_{x=ROImin}^{ROImax} \sum_{y=ROImin}^{ROImax} x * \frac{vpiksel(x,y)}{B} \tag{28}$$

$$y_{centar} = \sum_{x=ROImin}^{ROImax} \sum_{y=ROImin}^{ROImax} y * \frac{vpiksel(x,y)}{B} \tag{29}$$

Funkcija koja obavlja dane operacije nad ROI-em je pod nazivom „*centeroid()*“. Funkcija prihvaća segment slike koji je ograničen koordinatama regije od interesa, a nakon provedenih izračuna vraća koordinate centra u formatu pomičnog zareza dvostruke preciznosti.

Preciznost ove vrste izračuna centra proporcionalna je korijenu broja fotoelektrona koji su generirani od zvijezde koju promatramo. Izrađena je studija [] koja opisuje kako različiti faktori pogrešaka utječu na izračun centroida. U istoj studiji dan je način kako kompenzirati greške no pošto su kompenzacije računski zahtjevne, nisu implementirane u ovom radu.

Kako bi se pomoglo što preciznijem izračunu centroida, sam izračun se ne obavlja nad svakom uhvaćenom slikom pojedinačno nego se računa nad sumom slika nad kojom pomak nije značajan ili nije postojeći. ROI je zapravo prosječna vrijednost nekoliko uzastopnih fotografija.

Postoji nekoliko načina obrade regije od interesa. Prethodna je najjednostavnija,a samim time i najbrža što nam je iznimno važno zbog uštede na memoriji i procesorskoj snazi ograničenog Beagleboard sustava.

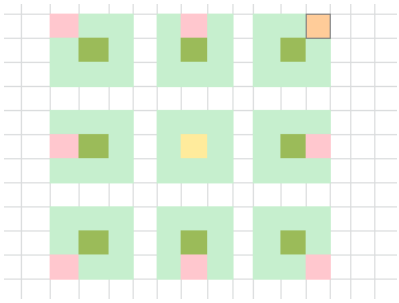
4.6. Blokovska komparacija dvaju slika

Ukoliko se stupanj algoritma izvodi drugi put ili više slike koje su u međuspremniku, provodi se algoritam blokovske komparacije dvaju slika. Radi se o algoritmu koji se standardno može naći u MPEG koderima videa koji omogućuju brzo prikazivanje slike. Naime kako se ne bi pri izradi videa kodirala svaka slika zasebno u potpunosti, kodira se samo dio slike koji se mijenja iz kadra u kadar. Ovaj postupak se primjenjuje zbog uštede na prostoru pri spremanju videa kao i brzini prikaza slike. Navedeni algoritam sastoji se od dijeljenja trenutne slike u manje blokove i uspoređivanje svakog dijela s prethodnom slikom. Napredniji algoritmi koriste pretraživanja slike kako bi

smanjili kompleksnost, i ne uspoređuju svaki dio slike sa prethodnom. Tada se koriste sljedeća pretraživanja:

- Logaritamsko,
- pretraživanje u tri koraka,
- pretraživanje u četiri koraka,
- i binarno pretraživanje.

U svrhu ovog diplomskog rada izrađen je poseban, modificirani, blokovski algoritam koji se razlikuje u tome što se slika ne pretražuje u potpunosti, iz razloga što se zvijezde iz kadra u kadar ne mogu previše pomaknuti. Pretražuje se samo okolni dio zvijezde koji je površine 3 puta veće nego regija od interesa. To jest pretražuje se gornji lijevi dio iznad regije interesa, veličine regije interesa pa sve do donjeg desnog dijela, isto veličine regije interesa, piksel po piksel, prikazano na slici 21.



Slika 21: pretraživanje slike za odgovarajućom regijom interesa

Ostatak slike se ne pretražuje te ukoliko je pomak vrlo velik bilježi se pogreška algoritma. Sama usporedba svakog bloka sa svakim provodi se prema opće prihvaćene formule za minimalnu apsolutnu razliku (eng. *MAD minimum absolute diference*). Nakon provedene obrade svih okvira slika za rezultat se uzima minimalna vrijednost, te je tamo trenutna slika potpuno preklopljena s originalom.

Formula za MAD dana je sljedećim izrazom:

$$MAD(x,c,y,t) = \sum_{x}^{pix_{max}-1} \sum_{y}^{pix_{max}-1} |d(x,y) - d(x+t,y+c)| \quad (30)$$

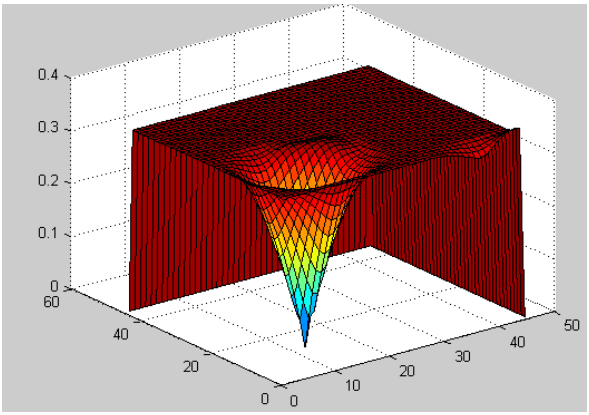
gdje je d(x,y) pozicija piksela referentne slike, a d(x+t,y+c) piksel na određenoj poziciji trenutne slike. Formula se izvršava za svaki okvir te se uzima minimum. Postoji još nekoliko kriterija za određivanje razlike među slikama kao što su kvadratna apsolutna razlika ,.-.. , no MAD se prema istraživanjima provedenim u [] pokazao najboljom i hardverski najmanje zahtjevnom opcijom.

Blokovsku komparaciju u ovom radu provodi funkcija *„usporedi_slike()“.* Spomenuta funkcija sama po sebi nema navedenu funkcionalnost nego poziva nekoliko podfunkcija kako što su „*izracunaj_kor_pomaka()*“ i „*MAD_funkcija()*“ tim redoslijedom. Funkcija obavlja logičku usporedbu nad rezultatima MAD funkcije te svrstava rezultat kao potvrdan ili ne. Ako je potvrdan tada se koordinate zvijezda poklapaju te im se ne mijenja marker, ukoliko nije algoritam se provodi još jednom na cijeloj slici, pomoću Matlabovog ugrađenog blokovskog komparatora koji koristi istu vrijednosnu funkciju. Matlabov blokovski algoritam se koristi isključivo ako se funkcija mogla izvršiti u vremenskom roku između uslikavanja.

Sljedeća funkcija koja se poziva je *„izracunaj_kor_pomaka()“* koja priprema slike za blokovsku obradu. Referentna slika se skraćuje na površinu tri puta veću od ROI-a, a kopija trenutne slike se smanjuje na veličinu ROI-a zvijezde koja je na redu za obradu.

„MAD_funkcija()“ izvršava obradu u četiri FOR petlje. Dvije petlje prolaze kroz sve okvire koji su mogući, a ograničeni su veličinom referentne slike, piksel po piksel. Unutarnje petlje obavljaju sumaciju apsolutnih vrijednosti prema formuli za MAD funkciju. Nakon sumacije, pronalazi se minimum vrijednosti za sve blokove koje su spremljene u polje. Dodatno, stvoreno je polje koje daje razliku

između dvaju slika tamo gdje je razlika najmanja tako da se, ukoliko zvijezda nije jednakog sjaja u svakoj od slika tada se računa apsolutna vrijednost razlike u postocima da se može numerički potvrditi da je slika jednaka prethodnoj (slika22).



Slika 22: graf polja sumacija

Po izlazu iz funkcije dobivaju se koordinate pomaka za određenu zvijezdu te se iz njih potvrđuje o kojoj se zvijezdi radi, a isto tako ove koordinate mogu se koristiti u predviđanju gdje će se zvijezda nalaziti u sljedećem kadru.

4.7. Detekcija zakreta koordinatnog sustava

Detekcija zakreta koordinatnog sustava odvija se na dva načina, kroz cijeli program. Prvi način je izračunom zakreta prema položaju zvijezda u dva susjedna kadra, dok je drugi način, točniji, izračun pomoću pomaka na temelju izdanih naredbi za uhodavanje teleskopa. Prvi način ostvaren je u funkciji „izracunaj_zakret()“. Funkcija prima koordinate svih trenutnih okvira koji su uslikani, pokazivač na trenutnu poziciju te broj zvijezda koji je detektiran u prethodnim koracima algoritma. Za svaku zvijezdu posebno se računa iznos kuta između dva susjedna okvira prema formuli:

$$\Delta = \arctan(\frac{x_1 - x_2}{y_1 - y_2}) \tag{31}$$

Za sve zvijezde kut zakreta bi trebao biti približno jednak te se računa prosječna vrijednost svih kutova.

Drugi račun zakreta kuta računa se kroz tijelo glavne funkcije odnosno u dijelu prvog stupnja. U naredbi sa slučajevima (eng. *switch case*) gdje se računa i pomak u pikselima u odnosu na izdanu naredbu za Bagleboard. Kutni se pomak bilježi ukoliko je izdana naredba u jednom smjeru no pomak koji se javio je za neki kut pomaknut od originalnog kuta. Tada se računa projekcija pomaka na os te se sljedeća naredba izdaje za kosinus kuta koji je izračunat.

4.8. Slanje naredbi na Beagleboard sustav

Na Beagleboard sustavu pokrenut je Ubuntu operacijski sustav. Sustav omogućuje pristup izlazno-ulaznim priključnicama (GPIO) pisanjem u datoteku. Postavljanje naponskih razina na priključnice Beagleboard sustava obavlja se putem konzolnih naredbi, točnije putem naredbe *echo*. Spomenutom naredbom obavljaju se sljedeći koraci za korištenje GPIO izlaza:

- inicijaliziranje GPIO priključnice za korištenje
- postavljanje GPIO priključnice kao izlazne
- pisanje vrijednosti signala.

Prvi korak obavlja se sljedećim naredbama:

```
cmd01 = c_string(['echo ', gpioPin, '> /sys/class/gpio/export']);
        coder.ceval('system', coder.rref(cmd01));
```

c_string naredba radi konkatenaciju nad izrazom navedenim u zagradama, dok coder.ceval referencira cijeli znakovni niz na konzolu sustava.

Naredbe za sljedeća dva koraka su iste, jedina razlika je u znakovnom nizu koji se postavlja na sustav. Znakovni niz za postavljanje priključnica na izlazne je sljedeći:

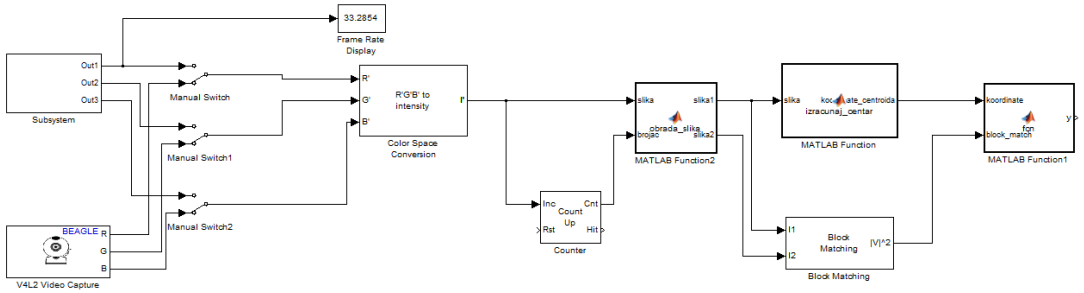
```
cmd11 = c_string(['echo out> /sys/class/gpio/gpio', gpioPin,
'/direction']);
```

Dok je za postavljanje priključnica u neko stanje:

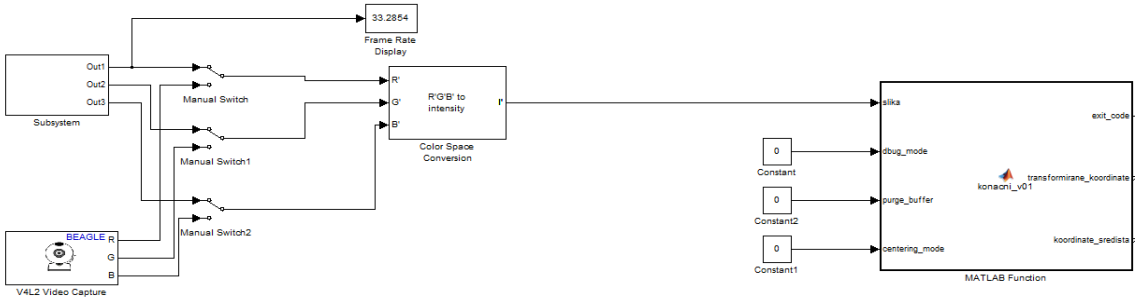
```
cmd11 = c_string(['echo out> /sys/class/gpio/gpio', gpioPin,
'/value']);
```

4.9. Simulink i matlab implementacija algoritma

Za potrebe diplomskog rada izrađena su dva po funkcionalnosti jednaka algoritma, no po implementaciji različita. Naime za testiranje korišteno je osobno računalo na kojem nije bilo potrebe koristiti permanentne varijable te se ta inačica algoritma izvodi iznimno brzo, dok je drugu inačicu bilo potrebno posebno modificirati i prepisati u simulink model kako bi cijela stvar funkcionirala na Beagleboard sustavu. Slika 23. prikazuje simulink inačicu algoritma, dok druga prikazuje matlab inačicu algoritma ali pod simulink sustavom.



Slika 10: simulink shema



Slika 11 matlab shema

4.10. Grafičko sučelje i komunikacija s Beagleboard pločicom

Za potrebe ovog rada razvijeno je grafičko sučelje koje prikazuje trenutan položaj zvijezda u odnosu na središte koordinatnog sustava. Implementacija sučelja rađena je u programskom jeziku C++, a korištene su biblioteke otvorenog koda GTK2.0+. Biblioteka GTK2.0+ osim što koristi klasične funkcije koristi i signale te tzv. *callback* funkcije kako bi ostvarila svoju funkcionalnost, a svoj rad zasniva na sustavnim i lokalnim događajima. Svaki dio aplikacije, koji se naziva *widget* generira signale kada se unutar njih odvija neki događaj. Signal je posebno oblikovana podatkovna struktura koja kod određenih funkcija izaziva reakciju odnosno poziva funkciju koja prati posebnu vrstu signala. *Callback* funkcije se pozivaju ukoliko se preko nekog *widgeta* ostvaruje višestruka funkcionalnost. Za izradu aplikacije koristio se primjer scribble-simple pod GNU licencom čija je distribucija i izmjena dozvoljena.

Aplikacija razvijena za potrebe ovog diplomskog rada sastoji se od nekoliko *widgeta:*

- Osnovnog okvira (eng. *window*),
- prostora za iscrtavanje grafa,
- gumba za izlaz
- okvira koji sadržava gumb i graf

Unutar same aplikacije sučelje se inicijalizira sljedećim kodom:

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_widget_set_name (window, "Star tracker v1.0");


  vbox = gtk_vbox_new (FALSE, 0);
  gtk_container_add (GTK_CONTAINER (window), vbox);
  gtk_widget_show (vbox);
```

Sljedeće funkcije ispisuju prostor za graf:

```
  drawing_area = gtk_drawing_area_new ();
  gtk_widget_set_size_request (GTK_WIDGET
(drawing_area), 320, 240);
  gtk_box_pack_start (GTK_BOX (vbox), drawing_area,
TRUE, TRUE, 0);
```

Signali koji se koriste za događaje i upravljanje mapom piksela su sljedeći:

```
  g_signal_connect (drawing_area, "expose-event",
           G_CALLBACK (expose_event), NULL);
  g_signal_connect (drawing_area,"configure-event",
           G_CALLBACK (configure_event), NULL);
```

Najvažnija funkcija koja se koristi za osvježavanje aplikacije svakih 100ms je sljedeća:

```
   g_timeout_add    (100,  (GSourceFunc)
drawarea1_timeout, (gpointer) drawing_area);
```

Unutar navedene funkcije odvija se čitanje iz datoteke u koju Beagleboard upisuje koordinate središta zvijezda. Nakon što je upisana nova koordinata zvijezda program će je svakih 100 ms ispisivat na ekran. Ukoliko je broj koordinata prevelik, program će automatski pobrisati datoteku te je zamijeniti novom.

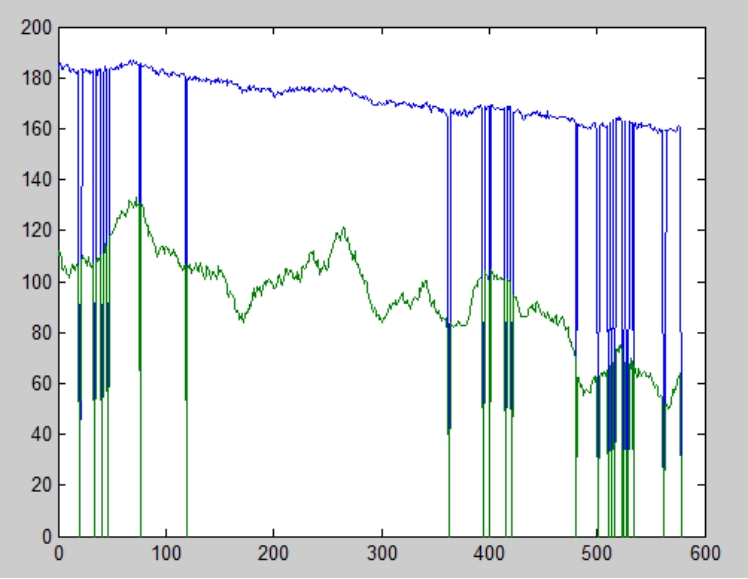
Prikaz grafičkog sučelja aplikacije je prikazan sljedećom slikom:

Slika 12: Grafičko sučelje programa

5. Rezultati algoritma

Algoritam je testiran na nekoliko izvora podataka. Prvi način testiranja bilo je umjetno napravljenim nizom od 87 slika koji se sastojao od jedne zvijezde koja je putovala križnom putanjom pod određenim kutom. Navedeni niz slika koristio se kako bi se emuliralo vođenje teleskopa po testnoj sekvenci od 10 kutnih stupnjeva u svim smjerovima okomitim na optičke osi.

Drugi niz fotografija sastojao se od 23.437 fotografija kako bi se testirala preciznost detekcije algoritma. Algoritam je u 98,3 % slučajeva uspješno detektirao najsjajnije zvijezde. U 1,7 % slučajeva zbog zamućenosti fotografija detekcija nije uspjela. Slika 23 prikazuje uspješnost detekcije zvijezde.



Slika 13: detekcija:plavom bojom izvedena je detekcija svjetlije zvijezde, zelenom bojom izvedena je detekcija tamnije zvijezde.

Pogreške su se javljale isključivo pri detekciji jedne zvijezde.

Treći način testiranja je prihvat slika uživo. Test se sastojao od toga da se prostorija zamrači te se na jednom kraju prostorije uključi jedna bijela LED

dioda, koju web kamera priključena na Beaglebord pokušava detektirati. Test je proveden uspješno.

6. Zaključak

Ovim diplomskim radom obuhvaćeni su principi rada algoritma za automatsko navođenje teleskopa. Razvijeno je nekoliko inačica algoritma u skladu sa potrebama sklopovlja na kojem se ono izvodi. Dobiveni rezultati su zadovoljavajući te je sljedeći korak ispitivanje na realnoj opremi odnosno na realnom teleskopu. Nažalost u vrijeme pisanja rada teleskop nije bio dostupan. Za ovaj algoritam važna je robusnost i izdržljivost na promjenjive uvjete pa je stoga dodan modul za ublažavanje pogrešaka. Naravno, za što točnija vođenja zahtijeva se redovitiju kalibraciju i provjeru ispravnosti. Pa bi ukoliko je to moguće bilo dobro provesti kalibraciju svakih nekoliko serija uslikavanja.

Što se tiče implementacijskih mogućnosti te isplativosti rješenja Beaglaboard pločica se nije pokazala kao dobro rješenje zbog svoje kompleksnosti i visoke cijene kako same pločice kao i dodatnih komponenata. Dok je najveći problem bila nepouzdanost pločice koja se rušila i pregrijavala. Jedina prednost implementacije na BeagleBoard pločici je bežična povezanost s računalom.

Teleskop bi se jednostavnim sklopom mogao bežično povezati na računalo te bi najbolja mogućnost bila kad bi se algoritam mogao pokretati na osobnom računalu dok bi jedini aktivni element na teleskopu bio bluetooth ili sličan bežični modul.

7. Sažetak i ključne riječi

U okviru diplomskog rada uspješno je izrađen sustav za automatsko praćenje pogleda teleskopa ostvaren na ugradbenom sustavu Beagleboard. Sustav je testiran te je pogodan za korištenje. Korišteno je nekoliko algoritama za ostvarivanje funkcionalnosti. Prvi je algoritam za detekciju zvijezda temeljen na detekciji praga najsvjetlije zvijezde, a zatim izolacije područja nad kojim se računa težinski centar svjetline. Drugi algoritam koji se koristi je algoritam usporedbe blokova slika, kako bi se odredio vektor pomaka zvijezda, te isto tako se potvrdio način pomicanja zvijezda. Posljednji algoritam koristi se za određivanje zakreta koordinatnog sustava kamere i koordinatnog sustava teleskopa, u slučaju da je kamera zakrenuta u odnosu na polje pogleda teleskopa. Od sklopovlja je izrađen priključak za teleskop koji se sa standardnim priključkom s jedne strane povezuje na teleskop, dok se s druge strane ide kabelom povezuje na Beagleboard sustav.

ključne riječi: praćenje pogleda, teleskop, ekvatorijalna montaža, beagleboard, centriranje, blok detekcija.

8. Summary and keywords

For purposes of this thesis, a system for automatic telescope guidance was successfully developed on a Beagleboard system. System was tested and it is suitable for usage. A few algorithms where used to accomplish it's functionality. First algorithm was used for star detection and it was based on threshold detection of brightest star and then it isolated an area of interest upon which a centeroiding algorithm was employed. Second algorithm was a block matching algorithm, used for detecting motion vectors of stars, and to determine ways of star movement. Last algorithm was used to detect a coordinate system rotation. Different coordinate systems are used for telescopes' view and other was used for camera's view. Additionally a connecting circuitry was developed to connect beagleboard system to the telescope.

Keywords: automatic guidance, telescope, equatorial mount, beagleboard, centeroiding, block matching

9. Reference

[1] W.Romanishin : An Introduction to Astronomical Photometry Using CCDs, http://observatory.ou.edu,2006

[2] Beagleboard projects, http://beagleboard.org/

[3] Make Magazine, blinking LEDs with beagleboard http://blog.makezine.com/2009/02/03/blinking-leds-with-the-beagle-board/, 11.9.2012.g

[4] Telescope mounts –Telescopes.com, http://www.telescopes.com/telescopes/telescopemountsarticle.cfm, siječanj 12.12.2012.

[5] Liebe C.C., Accuracy Performance of StarTrackers–A Tutorial

[6] Handbook of astronomical image processing

[7] CMOS Active Pixel Sensor Specific Performance Effects on Star Trackernmager Position Accuracy Bruce R. Hancock, Robert C. Stirbl, Thomas J. Cunningham,

[8] Subpixel centroiding algorithm for EMCCD star tracker, Yufeng Lia, Dongmei Lib, Lanbo Liang

[9] ATTITUDE CONTROL SYSTEM AND STAR TRACKER PERFORMANCE F THE WIDE-FIELD INFRARED EXPLORER SPACECRAFT Russ Laher Joe Catanzarite Tim Conrow Tom Correll, Roger Chen, et al.

[10] Telescopes – I. Optics & MountsDave Kilkenny

Dodatak A izvorni kod:

Glavna funkcija:

```
%konacni program 3 debug levela
%debug level 0 = bez debuga
%debug level 1 = sa debugom beagle razina
%rgb2 gray mora biti rjeseno vani
% N -broj zvijezda koji se promatra
% cat_path -putanja do FK5 kataloga
%fbuff_size -velicina frame buffera
%centering mode = 0 -najsjajnija zvijezda je u sredini
%centering_mode = 1 - težište N zvijezda je u sredini slike
%transformirane_koordinate i koordinate središta šalju se van
%na PID regulator ako je to potrebno ovisno o odabranom modu
centriranja
function [exit_code, transformirane_koordinate, koordinate_sredista] =
konacni_v01(slika2, dbug_mode, purge_buffer, centering_mode)
% novi prihvat slika
N =1;
fbuff_size = 83; % velicina cirkularnog spremnika slika
cat_path = 'C:\Users\Nikola\Desktop\diplomski\catalog\catalog';

sequence_code =1;
    koordinate = zeros(N,7,fbuff_size);
    %frame_buffer = zeros(size(slika,1),size(slika,2),fbuff_size);
    brojac_sekvence = 1;
    first_run =1;
    [fps_max, numpix] =  izracunaj_max_okret(1,1,50);% koliko dugo
treba vrtiti u jednom smjeru definicija funkcije :
izracunaj_max_okret(pixel_size, focal_length,promjer_kotaca);
    fbuff_pointer = 1;
    delta = zeros(5*brojac_sekvence*fbuff_size);
    final_delta = 0;
    broj_pogresaka = 0;
    pomak_pixla = zeros(3,fbuff_size); %pomak pixla isto mora biti
cirkularni
    vrijeme_izvodjenja =zeros(7);
    vrijeme_izvodjenja(1,7)= 100; %pocetna vrijednost u ms
    pis2 =1;
    koordinate_centra = zeros(1,7);
folder = 'C:\Users\Nikola\Desktop\diplomski\nanovo2\matlab\pict';

if ~isdir(folder)
    disp('greska nemam folder dobar');

end
dbug_mode = 1;
filepattern = fullfile(folder, '*.bmp');
jpegfiles = dir(filepattern);
jpegMAX = length(jpegfiles);
jpegfiles = jpegMAX;
trenutni = 0;
```

```
delta2 = zeros(1,6);
while trenutni < jpegMAX
%ucitaj i zbroji sliku
slika = imread
(sprintf('C:\\Users\\Nikola\\Desktop\\diplomski\\nanovo2\\matlab\\pict\
\%03d.bmp',trenutni));



if (dbug_mode==1)
    coder.extrinsic('imshow');
    coder.extrinsic('figure');
    coder.extrinsic('plot');
    coder.extrinsic('hold');
    coder.extrinsic('title');
    coder.extrinsic('display');
    coder.extrinsic('rgb2gray');
   % slika =rgb2gray(slika);
%     figure;
%     imshow(slika);
end
%slika =rgb2gray(slika);
slika = im2double(slika);
%slika = bpass(slika,1,10);
size_slika(1) = size(slika,1);
size_slika(2) = size(slika,2);



% persistent fbuff_pointer;
% persistent koordinate;
% persistent brojac_sekvence;
% persistent frame_buffer;
% persistent first_run;
% persistent fps_max;
% persistent vrijeme_izvodjenja;
% persistent sequence_code;
% persistent delta;
% persistent final_delta;
% persistent broj_pogresaka;
% persistent pomak_pixla;
% persistent pis2; %prvo izvođenje stupnja 2

%if (isempty(koordinate) || isempty(brojac_sekvence)||
isempty(frame_buffer) || isempty(first_run) || purge_buffer == 1)
    %run once

%end
%učitavanje slike u buffer
if (fbuff_pointer == fbuff_size)
    frame_buffer(:,:,1) = frame_buffer(:,:,fbuff_pointer-1);
    frame_buffer(:,:,2) = slika(:,:);
    fbuff_pointer =2;
```

```
end
%učitavanje slike u buffer
frame_buffer(:,:,fbuff_pointer) = slika(:,:);

%prvi i drugi stupanj programa
if(first_run)

    %sekvenca uhodavanja

    naredba(sequence_code);
    koordinate(:,:,fbuff_pointer) = izracunaj_centar(slika,N,0);
    if (fbuff_pointer > 1)
        prethodna_slika = frame_buffer(:,:,fbuff_pointer-1);
%         if(dbug_mode == 1)
%             figure;
%             imshow(prethodna_slika);
%         end
         match = usporedi_slike(prethodna_slika, slika,
koordinate(:,:,fbuff_pointer-1:fbuff_pointer), N, dbug_mode)
        if match == 0
            broj_pogresaka = broj_pogresaka +1;
            delta(brojac_sekvence) =  0;
            pomak_pixla(1,fbuff_pointer) = 0;
            pomak_pixla(2,fbuff_pointer) = 0;
        else
            delta(brojac_sekvence)=
izracunaj_zakret(koordinate(:,:,:),fbuff_pointer, N); %ovaj kod nis ne
radi !
            pomak_pixla(1,fbuff_pointer) =
koordinate(1,1,fbuff_pointer)- koordinate(1,1,fbuff_pointer-1);
%najsjajnija zvijezda je referentna za brojanje pixla i smjera
            pomak_pixla(2,fbuff_pointer) =
koordinate(1,2,fbuff_pointer)- koordinate(1,2,fbuff_pointer-1);
            pomak_pixla(3,fbuff_pointer) =
sqrt((pomak_pixla(1,fbuff_pointer))^2 +
(pomak_pixla(2,fbuff_pointer))^2);
        end

    end
    pomak_pxX = floor(sum(pomak_pixla(1,:)));
    pomak_pxY = floor(sum(pomak_pixla(2,:))); %
    pomak_pxD = sum(pomak_pixla(3,:));

    %sekvenca RA -> 10'' kod:1 -> 1/2 pixela po kutnoj sekundi 20
    %px se treba pomaknuti u desno
    %sekvenca RA <- 20'' kod:2
    %sekvenca RA -> 10'' kod:3
    %sekvenca DEC -> 10'' kod:4
    %sekvenca DEC <- 10 '' kod:5
    %kod:6 kraj sekvence
    switch sequence_code
        case 1
             if pomak_pxD >= 13
```

```
                vrijeme_izvodjenja(1) = brojac_sekvence;
                sequence_code = 2;
                pomak_pixla = zeros(2,fbuff_size);
                if (abs(pomak_pxY) >1 || abs(pomak_pxY) >1)
                   delta2(1) = atan(pomak_pxX/pomak_pxY); %saznati
koja je prava formula
                   delta2(1) = radtodeg(delta2(1));
                end
            end
        case 2
            if pomak_pxD >= 26
                vrijeme_izvodjenja(2) = abs( brojac_sekvence -
vrijeme_izvodjenja(1));
                sequence_code = 3;
                pomak_pixla = zeros(2,fbuff_size);
                if (pomak_pxY >1 || pomak_pxY < -1)
                  delta2(2) = atan(pomak_pxX/pomak_pxY); %saznati koja
je prava formula
                   delta2(2) = radtodeg(delta2(2));
                end
            end
        case 3
            if pomak_pxD >= 13

            vrijeme_izvodjenja(3) = abs( brojac_sekvence -
vrijeme_izvodjenja(2));
                sequence_code = 4;
                pomak_pixla = zeros(2,fbuff_size);
                if (pomak_pxY >1 || pomak_pxY < -1)
                   delta2(3) = atan(pomak_pxX/pomak_pxY); %saznati
koja je prava formula
                   delta2(3) = radtodeg(delta2(3));
                end
            end
        case 4
            if pomak_pxD >= 13

            vrijeme_izvodjenja(4) = abs( brojac_sekvence -
vrijeme_izvodjenja(3)-vrijeme_izvodjenja(2));
                sequence_code = 5;
                pomak_pixla = zeros(2,fbuff_size);
                if (pomak_pxX >1 || pomak_pxX < -1)
                   delta2(4) = atan(pomak_pxX/pomak_pxY); %saznati
koja je prava formula
                   delta2(4) = radtodeg(delta2(4));
                end
            end
        case 5
            if pomak_pxD >= 13
                sequence_code = 6;
                pomak_pixla = zeros(2,fbuff_size);
           vrijeme_izvodjenja(5) = abs( brojac_sekvence -
vrijeme_izvodjenja(4) -vrijeme_izvodjenja(2)-vrijeme_izvodjenja(3));
                if (abs(pomak_pxX) >1 || abs(pomak_pxY) <1)
```

```
                    delta2(5) = atan(pomak_pxX/pomak_pxY); %saznati
koja je prava formula
                    delta2(5) = radtodeg(delta2(5));
                end
            end
        case 6
            if pomak_pxD >= 10;
                pomak_pixla = zeros(2,fbuff_size);
                if (pomak_pxX >1 || pomak_pxX < -1)
                    delta2(6) = atan(pomak_pxX/pomak_pxY); %saznati
koja je prava formula
                    delta2(6) = radtodeg(delta2(6));
                end
                sequence_code = 1;
                first_run =0; % kraj sekvence
                final_delta = sum(abs(delta2))/6;
                vrijeme_izvodjenja(6) = abs( brojac_sekvence -
vrijeme_izvodjenja(4) -vrijeme_izvodjenja(2)-vrijeme_izvodjenja(3)-
vrijeme_izvodjenja(5));
            end
        otherwise
                %std_error();
    end



    brojac_sekvence = brojac_sekvence+1;
    fbuff_pointer = fbuff_pointer+1;

else
      % run once: očisti buffer i koordinate, očitaj trenutnu
poziciju,
      % spremi i izađi. Pri drugom ulasku računaj sljedeći impuls i
vodi.
     if (pis2 == 1)
           vrijeme_izvodjenja(1) = sum(vrijeme_izvodjenja(1:6))/6;
           vrijeme_izvodjenja(7) = brojac_sekvence * fps_max;
           pis2=0;
           frame_buffer(:,:,1) = frame_buffer(:,:,fbuff_pointer-1);
           frame_buffer(:,:,2) = slika(:,:);
           fbuff_pointer = 2;
           koordinate = zeros(N,7,fbuff_size);
           koordinate_centra(:,:,fbuff_pointer) =
izracunaj_centar(slika, dbug_mode);
     end
     %guidance
     koordinate(:,:,fbuff_poiner) = izracunaj_centar(slika,
dbug_mode);
     if (fbuff_pointer > 1)
        prethodna_slika = frame_buffer(:,:,fbuff_pointer-1);
        match = usporedi_slike(prethodna_slika, slika,
koordinate(:,:,fbuff_pointer-1:fbuff_pointer));
        if match == 0
```

```
                broj_pogresaka = broj_pogresaka +1;
            end
        end
        if (fbuff_pointer < fbuff_size-1)
                attempt_catalogue(frame_data, cat_path) % predajemo
cijeli buffer i katalog
        end
        sequence_code = izracunaj_sljedeci_impuls(final_delta,
koordinate(:,:,fbuff_pointer),koordinate_centra,centering_mode,vrijeme_
izvodjenja, dbug_mode);
        naredba(sequence_code);
end

 %todo napraviti uspoređivač slika kakav spada

 %     dodati debug uvijete [v]
 %     napraviti funkcije koje nedostaju [v]
 %     napraviti prvi test i doraditi na verziju 1.0[v]
 %     kalman []
 %     gauss []
 %     kod errora ugraditi warnign sustav (ako javi gresku da ne moze
naci zvjezdu,
 %     ako se to vise puta ponovi pročisti sve buffere) []
 % end


trenutni = trenutni +1;

end% end while petlje



end
```

funkcija izračunaj okret:

```
%funkcija vraca
function [brojac_max, N_pixela] = izracunaj_max_okret(px, fl, R)
    fps = 1;
    pixel_size = 0.017; %velicina piksela u mm
    focal_length = 4.0 % fokalna duljina u mm
    promjer_kotaca = 50 % promjer kotaca u mm
    brzina_kretanja = 3; % brzina u mm/s
    aspp= px/fl *206265 ; % kutne sekunde * duljina pokreta(10 min tj
600 s)
    N_pixela = aspp * 600;
    O = 2 * R * pi ;
    vrijeme_1zakreta = floor(O/brzina_kretanja); %vrijeme potrebno za 1
zakret
    brojac_max = vrijeme_1zakreta/fps; %brojac max ce se staviti na
neko ne pretpostavljeno vrijeme
```

Izračunaj centar:

```
%slika = slika
%N- broj zvijezda koji planiramo pratiti
function [koordinate_centroida] = izracunaj_centar(slika,N, dbug_mode)
coder.extrinsic('display');
coder.extrinsic('figure');
coder.extrinsic('imshow');
coder.extrinsic('title');
coder.extrinsic('hold');
coder.extrinsic('plot');
coder.extrinsic('disp');
dbug_mode=1;
medijan = median(median(slika));
maksimum = max(slika(:));
treshold = abs(maksimum - medijan)* 0.4;
% if dbug_mode == 1
%    figure;
%    imshow(slika);
% end
kopija = slika;
[r,c] = find(slika == maksimum); % prva x, y koordinata slike
ima_neceg=1;
num_coor=1;
koordinate = zeros(N,4);

while ima_neceg
[koordinate(num_coor,:), slika] = roi(slika, r(1), c(1), treshold,
dbug_mode);

[r,c] = find(slika > (maksimum*0.7));
if ~any(r) | ~any(c)
    ima_neceg =0;
end
num_coor = num_coor+1;
end

% privremene koordinate
centar = zeros(N,2);
koordinate_centroida = zeros(N,7);

for i=1:1:(N) %for i=1:1:(num_coor-1)
      od_v=koordinate(i,3);
      do_v=koordinate(i,1);
      od_s=koordinate(i,4);
      do_s=koordinate(i,2);
      if (od_v == 0 || do_v ==0 || od_s == 0 || do_s == 0)
          if(dbug_mode == 1)
               disp('nemam vise koordinata');
          end
           break;
      end
      podmatrica = kopija(od_v:do_v,od_s:do_s);
     %izracun centra prve slike
```

```
      [centar(i,:)] = centeroid(podmatrica,dbug_mode);
      if(dbug_mode == 1)
%         figure
%         imshow(slika);
%         title ('ovo je slika');
%         hold on;
%         plot(koordinate(i,4),koordinate(i,3),'o');
%         plot(koordinate(i,2),koordinate(i,3),'o');
%         plot(koordinate(i,4),koordinate(i,1),'o');
%         plot(koordinate(i,2),koordinate(i,1),'o');
%         plot((koordinate(i,4)+ centar(i,1)),( koordinate(i,3)+
centar(i,2)),'r*');
%         hold off;
      end
      koordinate_centroida(i,1) = (koordinate(i,4)+ centar(i,1)); %x
koordinata centroida
      koordinate_centroida(i,2) = (koordinate(i,3)+ centar(i,2)); %y
koordinata centroida
      koordinate_centroida(i,3) = i; % marker centroida
      koordinate_centroida (i,4) = od_v %ROI y koordinata pocetna
      koordinate_centroida (i,5) = do_v %ROI y koordinata krajnja
      koordinate_centroida (i,6) = od_s %ROI x koordinata pocetna
      koordinate_centroida (i,7) = do_s %ROI x koordinata krajnja
end

if(dbug_mode == 1)
    display('broj koordinata');
end

end
```

Funkcija centeroid:

```
function [kvazi_koordinate] = centeroid(grsc, dbug_mode)
% coder.extrinsic('figure');
% coder.extrinsic('title');
% coder.extrinsic('imshow');
% coder.extrinsic('hold');
% coder.extrinsic('plot');
brightness = sum(grsc(:));
br_red = size(grsc,1);
br_stup = size (grsc,2);

% matrix_x = zeros(br_red:br_stup);
% matrix_y = zeros(br_red:br_stup);

% for i=1:1:br_red
%     for j=1:1:br_stup
%      matrix_x(i,j) =i-ceil(br_stup/2);
%      matrix_y(i,j) =j-ceil(br_red/2);
%     end
% end
```

```
matrix_x = ones(br_red,1)*[1:br_stup];
matrix_y = [1:br_red]'*ones(1,br_stup);

xi = sum ( sum (uint8(matrix_x) .* uint8(grsc)))/brightness;
yi = sum ( sum (uint8(matrix_y) .* uint8(grsc)))/brightness;

bright = max(max(grsc));
kvazi_koordinate = zeros(1,2);
kvazi_koordinate(1,1)= xi;
kvazi_koordinate(1,2)= yi;

% if(dbug_mode == 1)
%     figure;
%     title('ovo je izracun centroide');
%     imshow(grsc);
%     hold on;
%     plot(xi,yi,'*r');
%     hold off;
% end
end
```

funkcija usporedi slike

```
%koordinate_gl sastoje se od koordinate prve zvijezde, njegovog ID-a,
%koordinata roi-a i tako za svaki frame. Nama trebaju 2 susjedna framea
i
%njih moramo usporediti pa se funkciji predaju koordinate trenutne
slike i
%prethodnog framea.

function [da_ili_ne] = usporedi_slike(slika_stara, slika_nova,
koordinate_gl,N,dbug_mode)

    coder.extrinsic('figure');
    coder.extrinsic('plot');
    coder.extrinsic('imshow');
    coder.extrinsic('title');
    coder.extrinsic('hold');

koordinate_sl1= zeros(N,7);
koordinate_pomaka = zeros(N,8);%!!!!!!
da_ili_ne0 = zeros(N);
da_ili_ne =0;
for j=1:1: N
    koordinate_sl1(:,:) = squeeze(koordinate_gl(:,:,j));
end
for i = 1 : 1 : N% size(koordinate_sl1,2)
   sirina= abs(koordinate_sl1(i,7)-koordinate_sl1(i,6));
   visina= abs(koordinate_sl1(i,5)-koordinate_sl1(i,4));
```

```
     koordinate_pomaka (i,:) =
izracunaj_kor_pomaka(koordinate_sl1(i,6),koordinate_sl1(i,4),koordinate
_sl1(i,7),koordinate_sl1(i,5), slika_stara, slika_nova, dbug_mode);

        if dbug_mode == 1
%             figure
%             imshow(slika_stara);
%             title ('ovo je slika');
%             hold on;
%             plot(koordinate_pomaka(i,1),koordinate_pomaka(i,2),'y*');
%             plot(koordinate_pomaka(i,6),koordinate_pomaka(i,4),'yo');
%             plot(koordinate_pomaka(i,6),koordinate_pomaka(i,5),'yo');
%             plot(koordinate_pomaka(i,7),koordinate_pomaka(i,4),'yo');
%             plot(koordinate_pomaka(i,7),koordinate_pomaka(i,5),'yo');
%             plot(koordinate_sl1(i,1),koordinate_sl1(i,2),'go');
%             plot(koordinate_sl1(i,6),koordinate_sl1(i,4),'go');
%             plot(koordinate_sl1(i,6),koordinate_sl1(i,5),'go');
%             plot(koordinate_sl1(i,7),koordinate_sl1(i,4),'go');
%             plot(koordinate_sl1(i,7),koordinate_sl1(i,5),'go');
%
%             hold off;
        end
        if (koordinate_pomaka(i,8) <0.2)
            da_ili_ne0(i) = 1;
        else
            da_ili_ne0(i) = 0;
        end;
end
da_ili_ne = sum(da_ili_ne0(:));
        if  da_ili_ne == N
         da_ili_ne = 1;
        else
          da_ili_ne =0;
        end
end
```

Funkcija izracunaj_kooridnate_pomaka:

```
% izracunavanje pomaka
% x1-gornja lijeva X koordinata okvira kojeg pretrazujemo
% y1-gornja lijeva Y koordinata okvira kojeg pretrazujemo
% sirina -sirina stare slike
% visina - visina stare slike
% stara_slika -samo ROI od stare slike koji se traži na novoj slici
% nova slika - nova slika (cijela)
% koordinate_pomaka


function [koordinate_pomaka] = izracunaj_kor_pomaka(x1,y1,x2, y2,
stara_slika, nova_slika, dbug_mode)
sirina = abs(x2-x1);
visina = abs(y2-y1);
%#codegen
assert(x1<=800,'now is the time 1');
assert(x2<=800,'now is the time 2');
```

```
assert(y1<=800,'now is the time 3');
assert(y2<=800,'now is the time 4');
assert(sirina<=800,'now is the time 1');
assert(visina<=800,'now is the time 1');

novi_roi = zeros(sirina,visina);
% stara_slika = im2double(stara_slika);
% nova_slika = im2double(nova_slika);
%velicina bloka koju pretrazujemo mozemo nastelati
%vel_blok = max(sirina, visina)
%boundcheck
sirina_ns = size(nova_slika,1);
visina_ns = size(nova_slika,2);
% sirina_ns = 10;
% visina_ns = 10;

koordinate_pomaka=zeros(1,8);

if (x1-sirina<0)
  pocetnaX = 1;
else
  pocetnaX = x1 - sirina;
end
if (x1+sirina>sirina_ns)
  krajnjaX = sirina_ns;
else
  krajnjaX = x1 + 3*sirina;
end

if (y1-visina<0)
  pocetnaY = 1;
else
  pocetnaY = y1 - visina;
end

if (y1+visina>visina_ns)
  krajnjaY = visina_ns;
else
  krajnjaY = y1 + 3*visina;
end
%kraj bound checka
% nadodati dio koji ne dozvoljava da je matrica u ćošku i odsječena

nova_matrica =nova_slika(pocetnaY:krajnjaY,pocetnaX:krajnjaX);

% figure;
% subplot(3,1,1),imshow(stara_slika(y1:visina+y1-1,x1:sirina+x1-1));
% subplot(3,1,2),imshow(stara_slika);
% subplot(3,1,3),imshow(nova_matrica);

 [mjera_slicnosti, lokacija ] = MAD_funkcija(stara_slika(y1:visina+y1-
1,x1:sirina+x1-1), nova_matrica,dbug_mode);
```

```
%izdvaja se nova matrica gdje se sumnja da je novi centar


novi_roi = nova_matrica(lokacija(1):lokacija(1) + sirina, lokacija(2) :
lokacija(2) + visina);
% figure;
% imshow (novi_roi);
%računa se novi centar
      koordinate_pomaka(1:2) = centeroid(novi_roi,0);
      koordinate_pomaka(1) = koordinate_pomaka(1)+lokacija(2)+
pocetnaX;
      koordinate_pomaka(2) = koordinate_pomaka(2)+lokacija(1) +
pocetnaY;
%       koordinate_pomaka(3) = i; % marker centroida
      koordinate_pomaka (4) = lokacija(1)+pocetnaY; %ROI y koordinata
pocetna
      koordinate_pomaka (5) = lokacija(1) + pocetnaY + visina; %ROI y
koordinata krajnja
      koordinate_pomaka (6) = lokacija(2) + pocetnaX; %ROI x koordinata
pocetna
      koordinate_pomaka (7) = lokacija(2) + pocetnaX + sirina; %ROI x
koordinata krajnja
      koordinate_pomaka (8) = mjera_slicnosti;
% todo: preračunaj nove koordinate s obzirom na sliku

end
```

MAD funkcija:

```
function [cifra, lokacija] = MAD_funkcija(referentni_blok,
trenutni_blok, dbug_mode)% stara slika, nova slika je veća
coder.extrinsic('figure');
coder.extrinsic('surf');
sirina_bloka = size(referentni_blok,1);

visina_bloka = size(referentni_blok,2);
lokacija = zeros(2);
s_pretrage = size(trenutni_blok,1);
v_pretrage = size(trenutni_blok,2);
suma = zeros(s_pretrage, v_pretrage);
dodatna_matrica_s = zeros (s_pretrage,visina_bloka);
dodatna_matrica_v = zeros (sirina_bloka,v_pretrage+visina_bloka);
trenutni_blok = horzcat(trenutni_blok, dodatna_matrica_s);
trenutni_blok = vertcat(trenutni_blok, dodatna_matrica_v);
minimum = 10000000000;
iznos = 0;
suma2 = zeros (visina_bloka,sirina_bloka);
lokacijaX =1;
lokacijaY =1;
%vanjska petlja koja vrti okvire
```

```
for i = 1 : (s_pretrage-1)
    for j = 1 : (v_pretrage-1)
     %unutarnja petlja koja racuna mad za blokove
       for k = 1 : sirina_bloka-1
           for l = 1 : visina_bloka-1
               iznos =(iznos +abs(trenutni_blok(k +(i-1),l +(j-1))-
referentni_blok(k,l) ));
           end
       end
        iznos = iznos /(sirina_bloka * visina_bloka);
       if (iznos < minimum)
                   minimum = iznos;
                   lokacijaX = i;
                   lokacijaY = j;
      for k = 1 : sirina_bloka-1
           for l = 1 : visina_bloka-1
               suma2 (k,l) =(abs(trenutni_blok( k +(lokacijaX-1),l
+(lokacijaY-1) )-referentni_blok(k,l) ));
                end
      end
        if dbug_mode == 1
          %  figure;surf(suma2);
        end
        end
       suma(i,j) = iznos;
    end
end

if dbug_mode == 0
  %  figure;surf(suma);
end
cifra = sum(sum(suma2))/(size(suma2,1)*size(suma2,2));

lokacija(1) = lokacijaX;
lokacija(2) = lokacijaY;

end
```

Funkcija izračunaj zakret :

```
function [delta] = izracunaj_zakret (koordinate, buffer_pointer, N)
broj_frameova =2;
delta1 = zeros(N,broj_frameova);
delta = zeros(N);
%ekstrakcija_točaka odnosno centara zvijezda

    for i= 1:1:N
        for j=1:1:broj_frameova-1
            delta1(i,j)= atan( (koordinate(i,1,j) - koordinate (i,1,j)
)/(koordinate(i,2,j) - koordinate (i,2,j) ) );
        end
    end
    %j ide po frameovima i racuna deltu
```

```
    %i ide po zvjezdama
    %racunamo kut za istu zvijezdu, ali između različitih frameova
    % uzimamo prosjecan delta u nekoliko frameova
    for k = 1:1:N
        delta(k) =sum (delta1(k,:))/N;
    end
    %prosjecan delta za zvijezde i dobivamo konacni delta odnosno
zakret
    %koordinatnog sustava
    delta = sum(delta(:));
end
```

Funkcija izračunavanja sljedećeg impulsa:

```
function [sequence_code trajanje]= izracunaj_sljedeci_impuls(delta,
koordinate, koordinate_centra,centering_mode,vrijeme_izvodjenja,slika,
dbug_mode)
koordinate = squeeze (koordinate);
sequence_code=0;
udaljenost = 0;
persistent preostalo_vremena;
if isempty(preostalo_vremena)
    preostalo_vremena = 0;
end

switch centering_mode
    case 0% centering mode je 0 znači koritistimo centar slike
        centar_slikeX = floor(size(slika,1))/2;
        centar_slikeY = floor(size(slika,2))/2;

        pozicija_zvijezdeX = koordinate(1,1);
        pozicija_zvijezdeY = koordinate(1,2);

        udaljenost = sqrt( (pozicija_zvijezdeX-centar_slikeX)^2 +
(pozicija_zvijezdeY-centar_slikeY)^2);
        udaljenost = floor(udaljenost)* cos(delta);

        if udaljenost > 0;
            preostalo_vremena = udaljenost * vrijeme_izvodjenja;
        end

        if preostalo_vremena ~= 0
            if(pozicija_zvijezdeX < centar_slikeX && pozicija_zvijezdeY
< centar_slikeY )
                sequence_code = 5;
            end
            if(pozicija_zvijezdeX > centar_slikeX && pozicija_zvijezdeY
< centar_slikeY )
                sequence_code = 7;
            end
```

```
            if(pozicija_zvijezdeX < centar_slikeX && pozicija_zvijezdeY
> centar_slikeY )
                sequence_code = 4;
            end
            if(pozicija_zvijezdeX > centar_slikeX && pozicija_zvijezdeY
> centar_slikeY )
                sequence_code = 6;
            end
            if(pozicija_zvijezdeX == centar_slikeX &&
pozicija_zvijezdeY < centar_slikeY )
                sequence_code = 3;
            end
            if(pozicija_zvijezdeX == centar_slikeX &&
pozicija_zvijezdeY > centar_slikeY )
                sequence_code = 2;
            end
            if(pozicija_zvijezdeX < centar_slikeX && pozicija_zvijezdeY
== centar_slikeY )
                sequence_code = 0;
            end
            if(pozicija_zvijezdeX > centar_slikeX && pozicija_zvijezdeY
== centar_slikeY )
                sequence_code = 1;
            end
        else
            sequence_code =9;
        end;
    case 1
        centar_slikeX = floor(koordinate_centra(1));
        centar_slikeY = floor(koordinate_centra(2));

        pozicija_zvijezdeX = koordinate(1,1);
        pozicija_zvijezdeY = koordinate(1,2);

        udaljenost = sqrt( (pozicija_zvijezdeX-centar_slikeX)^2 +
(pozicija_zvijezdeY-centar_slikeY)^2);
        udaljenost = floor(udaljenost)* cos(delta);

        if udaljenost > 0;
            preostalo_vremena = udaljenost * vrijeme_izvodjenja;
        end

        if preostalo_vremena ~= 0
            if(pozicija_zvijezdeX < centar_slikeX && pozicija_zvijezdeY
< centar_slikeY )
                sequence_code = 5;
            end
            if(pozicija_zvijezdeX > centar_slikeX && pozicija_zvijezdeY
< centar_slikeY )
                sequence_code = 7;
            end
            if(pozicija_zvijezdeX < centar_slikeX && pozicija_zvijezdeY
> centar_slikeY )
```

```
                sequence_code = 4;
            end
            if(pozicija_zvijezdeX > centar_slikeX && pozicija_zvijezdeY
> centar_slikeY )
                sequence_code = 6;
            end
            if(pozicija_zvijezdeX == centar_slikeX &&
pozicija_zvijezdeY < centar_slikeY )
                sequence_code = 3;
            end
            if(pozicija_zvijezdeX == centar_slikeX &&
pozicija_zvijezdeY > centar_slikeY )
                sequence_code = 2;
            end
            if(pozicija_zvijezdeX < centar_slikeX && pozicija_zvijezdeY
== centar_slikeY )
                sequence_code = 0;
            end
            if(pozicija_zvijezdeX > centar_slikeX && pozicija_zvijezdeY
== centar_slikeY )
                sequence_code = 1;
            end
        else
            sequence_code =9;
        end;
    otherwise
        %do nothing

end



end
```

Funkcija za izdavanje naredbi:

```
function fcn(komanda)
%#codegen
%komande
%000-desno
%001-lijevo
%010-gore
%011-dolje
%100-desno-gore
%101-desno-dole
%110-lijevo-gore
%111-lijevo-dole
% This MATLAB function shows how to setup and use a GPIO pin as output.
The
% function basically issues a number of system calls to do the
following:
% 1. Export GPIO pin for use
% 2. Setup GPIO direction as output
% 3. Write the value of input signal, u, to the GPIO pin
```

```
%
% Input to this function should be a boolean scalar. If the value of u
is positive
% then GPIO pin goes high. Else GPIO pin goes low.

persistent firstTime;
komanda =5;
gpioPin = '139';
gpioPin2 = '140';
gpioPin3 = '141';
gpioPin4 = '142';
coder.extrinsic('disp');
if isempty(firstTime)
    firstTime = 0;
    if isequal(coder.target, 'rtw')
        % Export GPIO pins
        cmd01 = c_string(['echo ', gpioPin, '>
/sys/class/gpio/export']);
        coder.ceval('system', coder.rref(cmd01));
        cmd02 = c_string(['echo ', gpioPin2, '>
/sys/class/gpio/export']);
        coder.ceval('system', coder.rref(cmd02));
        cmd03 = c_string(['echo ', gpioPin3, '>
/sys/class/gpio/export']);
        coder.ceval('system', coder.rref(cmd03));
        cmd04 = c_string(['echo ', gpioPin4, '>
/sys/class/gpio/export']);
        coder.ceval('system', coder.rref(cmd04));
        % Set GPIO pin direction to output
        cmd11 = c_string(['echo out> /sys/class/gpio/gpio', gpioPin,
'/direction']);
        coder.ceval('system', coder.rref(cmd11));
        cmd12 = c_string(['echo out> /sys/class/gpio/gpio', gpioPin2,
'/direction']);
        coder.ceval('system', coder.rref(cmd12));
        cmd13 = c_string(['echo out> /sys/class/gpio/gpio', gpioPin3,
'/direction']);
            coder.ceval('system', coder.rref(cmd13));
        cmd14 = c_string(['echo out> /sys/class/gpio/gpio', gpioPin4,
'/direction']);
        coder.ceval('system', coder.rref(cmd14));

    else
        disp('Initialize GPIO pin as output');
    end
end

% Set GPIO pin logic level using a system call
% if (u > 0)
%     cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin,
'/value']);
% else
%     cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin,
'/value']);
```

```
% end


switch komanda
    case 1
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 2
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 3
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 4
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 5
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 6
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
```

```
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 7
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    case 8
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin2,
'/value']);
        cmd = c_string(['echo 0 > /sys/class/gpio/gpio', gpioPin3,
'/value']);
        cmd = c_string(['echo 1 > /sys/class/gpio/gpio', gpioPin4,
'/value']);
    otherwise
        display('unknown command sent to output of beagleboard');
end

if isequal(coder.target, 'rtw')
    coder.ceval('system', coder.rref(cmd));
else
    disp(cmd);
end

end

function str = c_string(str)
% Convert MATLAB string to C-string by adding a string termination
% character
str = [str, 0];
end
```

Aplikacija:

```
#include <stdlib.h>
#include <stdio.h>
#include <gtk/gtk.h>

/* Backing pixmap for drawing area */
static GdkPixmap *pixmap = NULL;

/* Create a new backing pixmap of the appropriate size */
```

```
static gboolean configure_event( GtkWidget         *widget,
                                 GdkEventConfigure *event )
{
  if (pixmap)
    g_object_unref (pixmap);

  pixmap = gdk_pixmap_new (widget->window,
                           widget->allocation.width,
                           widget->allocation.height,
                           -1);
  gdk_draw_rectangle (pixmap,
                      widget->style->black_gc,
                      TRUE,
                      0, 0,
                      widget->allocation.width,
                      widget->allocation.height);


  return TRUE;
}

/* Redraw the screen from the backing pixmap */
static gboolean expose_event( GtkWidget      *widget,
                              GdkEventExpose *event )
{
  gdk_draw_drawable (widget->window,
                     widget->style->fg_gc[gtk_widget_get_state (widget)],
                     pixmap,
                     event->area.x, event->area.y,
                     event->area.x, event->area.y,
                     event->area.width, event->area.height);

  return FALSE;
}

/* Draw a rectangle on the screen */
static void draw_brush( GtkWidget *widget,
                        gdouble    x,
                        gdouble    y)
{
  GdkRectangle update_rect;
  update_rect.x = 5;
```

```
  update_rect.y = 5;
  update_rect.width = 1;
  update_rect.height = 1;
  gdk_draw_rectangle (pixmap,
                      widget->style->white_gc,
                      TRUE,
                      update_rect.x, update_rect.y,
                      update_rect.width, update_rect.height);


  gtk_widget_queue_draw_area (widget,
                              update_rect.x, update_rect.y,
                              update_rect.width, update_rect.height);

}

static gboolean button_press_event( GtkWidget      *widget,
                            GdkEventButton *event )
{
  if (event->button == 1 && pixmap != NULL)
    draw_brush (widget, event->x, event->y);

  return TRUE;
}

static gboolean motion_notify_event( GtkWidget *widget,
                             GdkEventMotion *event )
{
  int x, y;
  GdkModifierType state;

  if (event->is_hint)
    gdk_window_get_pointer (event->window, &x, &y, &state);
  else
    {
      x = event->x;
      y = event->y;
      state = event->state;
    }

  if (state & GDK_BUTTON1_MASK && pixmap != NULL)
    draw_brush (widget, x, y);
```

```
  return TRUE;
}

void quit ()
{
  exit (0);
}

static gboolean drawarea1_timeout (GtkWidget *widget)
{
    FILE * stream;
    static GdkPoint points[1];    /* test */
    static int flag = 0;

    int i, x, y;
    int  x1 = widget->allocation.width;
    int  y1 = widget->allocation.height;

  GdkRectangle update_rect;


  stream = fopen ("myfile","r+");
  if (stream==NULL)
  { printf("ne mogu otvoriti fajl");
     }
fscanf(stream,"%d %d", &x, &y);
    //fclose (stream);


    if (pixmap== NULL){
       return(TRUE);
    }
    if (flag != 0){
       gdk_draw_points(pixmap, widget->style->black_gc, points,1);
    }

            points[0].x = x;
            points[0].y = y;
    gdk_draw_points(pixmap, widget->style->white_gc, points, 1);
    gtk_widget_queue_draw_area (widget, 0, 0, x1, y1);
             gdk_draw_line(pixmap,widget->style->white_gc,160,0, 160,240);
```

```
 gdk_draw_line(pixmap,widget->style->white_gc,0,120,320,120);
update_rect.x = x;
  update_rect.y = y;
  update_rect.width = 3;
  update_rect.height = 3;
  gdk_draw_rectangle (pixmap,
                      widget->style->white_gc,
                      TRUE,
                      update_rect.x, update_rect.y,
                      update_rect.width, update_rect.height);

    flag = 1;
    return (TRUE);
}

int main( int   argc,
          char *argv[] )
{
  GtkWidget *window;
  GtkWidget *drawing_area;
  GtkWidget *vbox;

  GtkWidget *button;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_widget_set_name (window, "Star tracker v1.0");

  vbox = gtk_vbox_new (FALSE, 0);
  gtk_container_add (GTK_CONTAINER (window), vbox);
  gtk_widget_show (vbox);

  g_signal_connect (window, "destroy",
                    G_CALLBACK (quit), NULL);

  /* Create the drawing area */

  drawing_area = gtk_drawing_area_new ();
  gtk_widget_set_size_request (GTK_WIDGET (drawing_area), 320, 240);
  gtk_box_pack_start (GTK_BOX (vbox), drawing_area, TRUE, TRUE, 0);
```

```
  gtk_widget_show (drawing_area);

  /* Signals used to handle backing pixmap */

  g_signal_connect (drawing_area, "expose-event",
                    G_CALLBACK (expose_event), NULL);
  g_signal_connect (drawing_area,"configure-event",
                    G_CALLBACK (configure_event), NULL);

  /* Event signals */

  g_signal_connect (drawing_area, "motion-notify-event",
                    G_CALLBACK (motion_notify_event), NULL);
  g_signal_connect (drawing_area, "button-press-event",
                    G_CALLBACK (button_press_event), NULL);

  gtk_widget_set_events (drawing_area, GDK_EXPOSURE_MASK
                                | GDK_LEAVE_NOTIFY_MASK
                                | GDK_BUTTON_PRESS_MASK
                                | GDK_POINTER_MOTION_MASK
                                | GDK_POINTER_MOTION_HINT_MASK);

  /* .. And a quit button */
  button = gtk_button_new_with_label ("Izlaz");
  gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
          g_timeout_add    (100,  (GSourceFunc) drawarea1_timeout, (gpointer) drawing_area);
  g_signal_connect_swapped (button, "clicked",
                                G_CALLBACK (gtk_widget_destroy),
                                window);
  gtk_widget_show (button);

  gtk_widget_show (window);

  gtk_main ();



  return 0;
}
```

Dodatak B: plan bušenja i položajni nacrt: