SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI PROJEKT

**Upravljanje i identifikacija broda CyberShip**

*Ines Šoić, Viktor Vladić*

**Nastavak razvoja platforme PlaDyPos**

*Berislav Marszalek, Nikola Stilinović, Zoran Triska*

**Komunikacija sa USBL modulom**

*Dean Ivošević*

**Razvoj driver modula za GOPRO kameru, CyberShip i PlaDyPos**

*Darko Dujmović, Ivana Mikolić*

**Program za kalibraciju kompasa**

*Daniel Zima*

**Spajanje CTD mjerne sonde s LinkQuest modemom**

*Mato Tomić*

Mentor: prof.dr.sc. Zoran Vukić

Zagreb, siječanj, 2012.

Sadržaj

1. Puni naziv projekta 5

2. Opis problema/teme projekta 5

2.1 Upravljanje i identifikacija modela broda CyberShip (*Viktor Vladić i Ines Šoić*) 5

2.2 Nastavak razvoja platforme PlaDyPos (*Berislav Marszalek, Nikola Stilinović, Zoran Triska*) 5

2.3 Komunikacija sa USBL modulom (*Dean Ivošević*) 5

2.4 Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 5

2.5 Program za kalibraciju kompasa (*Daniel Zima*) 5

2.6 Spajanje CTD mjerne sonde s LinkQuest modemom (*Mato Tomić*) 5

3. Voditelj studentskog tima 5

4. Uvod 6

4.1 Upravljanje i identifikacija modela broda CyberShip (*Viktor Vladić i Ines Šoić*) 6

4.2 Nastavak razvoja platforme PlaDyPos (*Berislav Marszalek, Nikola Stilinović, Zoran Triska*) 6

4.3 Komunikacija sa USBL modulom (*Dean Ivošević*) 6

4.4 Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 6

4.5 Program za kalibraciju kompasa (*Daniel Zima*) 6

4.6 Spajanje CTD mjerne sonde s LinkQuest modemom (*Mato Tomić*) 6

5. Upravljanje i identifikacija modela broda CyberShip (Viktor Vladić i Ines Šoić) 7

5.1 VLASTITE OSCILACIJE 7

5.2 IZRAČUNAVANJE PARAMETARA 7

5.3 Implementacija u LabVIEW-u 8

6. Nastavak razvoja platforme PlaDyPos (*Berislav Marszalek, Nikola Stilinović, Zoran Triska*) 11

6.1 Matematičko modeliranje laboratorijske platforme 11

6.1.1 Model potisnika 12

6.1.2 Alokacija aktuatora 13

6.1.3 Dinamika 13

6.1.4 Kinematika 14

6.2 Praćenje putanje korištenjem virtualnog cilja 14

6.2.1 Model kinematičke pogreške 14

6.2.2 Projektiranje nelinearnog kinematičkog regulatora 17

6.2.3 Proširenje na dinamiku 18

6.3 sbRIO-9642 19

6.4 Veza sa platformom 19

7. Komunikacija sa USBL modulom (*Dean Ivošević*) 19

7.1 USBL 19

7.2 Čitanje podataka sa *USBL* modula 20

7.2.1 USBL i USBL HUB 21

7.2.2 SEANET 21

7.2.3 USBL CLIENT 21

7.2.4 MOOS 22

7.2.5 USBL LOCATOR 22

7.2.6 USBL DATA READER 24

7.3 Provjera rezultata 25

8. Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 27

8.1 GoPro HD Hero kamera 27

8.2 Hardware 29

8.2.1 Odabir mikrokontrolera 29

8.2.2 Serijska komunikacija 29

8.2.3 Elektronička pločica 31

8.3 Software 33

9. Program za kalibraciju kompasa (*Daniel Zima*) 33

9.1 Kompas 33

9.2 Odstupanje kompasa 33

9.3 Algoritam za kalibraciju kompasa[3] 33

9.3.1 Prvi korak kalibracijskog postupka 34

9.3.2 Drugi korak kalibracijskog postupka 35

9.3.3 Rekurzivna metoda najmanjih kvadarata (RLS metoda) 36

10. Spajanje CTD mjerne sonde s LinkQuest modemom (*Mato Tomić*) 38

10.1 Minos CTD 38

10.2 LinkQuest modem (UMW2000) 38

10.3 Komunikacija CTD sonde s udaljenim računalom pomoću LinkQuest modema 39

10.4 Osnovne upute za korištenje 41

11. Zaključak 41

11.1 Upravljanje i identifikacija modela broda CyberShip (*Viktor Vladić i Ines Šoić*) 41

11.2 Nastavak razvoja platforme PlaDyPos (*Berislav Marszalek, Nikola Stilinović, Zoran Triska*) 41

11.3 Komunikacija sa USBL modulom (*Dean Ivošević*) 41

11.4 Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 41

11.5 Program za kalibraciju kompasa (*Daniel Zima*) 42

11.6 Spajanje CTD mjerne sonde s LinkQuest modemom (*Mato Tomić*) 42

12. Literatura 42

12.1 Upravljanje i identifikacija modela broda CyberShip (*Viktor Vladić i Ines Šoić*) 42

12.2 Nastavak razvoja platforme PlaDyPos (*Berislav Marszalek, Nikola Stilinović, Zoran Triska*) 42

12.3 Positioning - Modeling and Identification. OCEANS, 2010.Komunikacija sa USBL modulom (*Dean Ivošević*) 43

12.4 Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 43

12.5 Program za kalibraciju kompasa (*Daniel Zima*) 43

12.6 Spajanje CTD mjerne sonde s LinkQuest modemom (*Mato Tomić*) 43

13. Sažetak 43

13.1 Upravljanje i identifikacija modela broda CyberShip (*Viktor Vladić i Ines Šoić*) 43

13.2 Nastavak razvoja platforme PlaDyPos (*Berislav Marszalek, Nikola Stilinović, Zoran Triska*) 43

13.3 Komunikacija sa USBL modulom (*Dean Ivošević*) 44

13.4 Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 44

13.5 Program za kalibraciju kompasa (*Daniel Zima*) 44

13.6 Spajanje CTD mjerne sonde s LinkQuest modemom (*Mato Tomić*) 44

14. Resursi 45

15. Struktura raspodijeljenog posla (engl. *Work Breakdown Structure* - WBS) 46

16. Gantogram 48

17. Zapisnici sastanaka 48

18. Tehnička dokumentacija 54

18.1 Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (*Ivana Mikolić, Darko Dujmović*) 54

18.1.1 KOD MIKROKONTROLER 54

18.1.2 KOD DRIVER 56

18.2 Program za kalibraciju kompasa (*Daniel Zima*) 61

18.2.1 TESTNI.H 61

18.2.2 ALGORITAM.CPP 61

18.3 Komunikacija sa USBL modulom (*Dean Ivošević*) 73

18.3.1 MAIN.CPP 73

# Puni naziv projekta

Razvoj broda CyberShip, platforme PlaDyPos i prateće opreme

# Opis problema/teme projekta

## ****Upravljanje i identifikacija modela broda CyberShip**** (Viktor Vladić i Ines Šoić)

U projektu je potrebno brod dovesti u takvo stanje da ga studenti jednostavno mogu koristiti na laboratorijskim vježbama. To uključuje omogućavanje daljinskog upravljanja i jednostavnog programiranja autopilota u programskom jeziku po želji (Labview, Java).

Program za identifikaciju parametara matematičkog modela broda korištenjem vlastitih oscilacija treba dovesti u oblik da je primjenjiv na stvanom sustavu u stvarnom vremenu i da je jednostavan za korištenje. Program treba implementirati na CyberShip-u i platformi PlaDyPos.

## Nastavak razvoja platforme PlaDyPos (Berislav Marszalek, Nikola Stilinović, Zoran Triska)

U sklop ovog projekta nastavlja se rad na platformi PlaDyPos. Potrebno je napraviti sljedeće stvari:

* Iskoristiti postojeći hardware za komunikaciju sa senzorima te razvijenu elektroniku te sve zajedno implementirati tako da se postigne funkcionalnost platforme da se mogu izvoditi testovi u stvarnim vremenskim uvjetima
* Podrebno je iskoristiti SBRio od National Instrumentsa i ostvariti upravljanje korištenjem te pločice
* Napraviti upravljanje korištenjem metode virtualnog cilja gdje će platforma pratiti CyberShip (ili voziti iza njega za njegovom putanjom, ili pokraj njega)

## Komunikacija sa USBL modulom (Dean Ivošević)

U sklopu ovog dijela projekta potrebno je iskoristiti USBL modul preko kojeg će se obavljati lociranje i praćenje izvora signala. Podatke koje USBL sustav računa potrebno je preuzeti i spremiti u bazu podataka. Također, potrebno je provjeravati točnost preuzetih podataka koje treba grafički prikazivati.

## Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

Potrebno je razviti driver module za kameru te dva plovila: CyberShip i Pladypos. Driver modul treba biti napravljen u skladu sa SensorDriver modulom kakav je definiran u LABUST repozitoriju. Definirati standardne i dodatne naredbe koje će kamera podržavati. Minimalno omogućiti paljenje i gašenje kamere, izbor moda rada, pokretanje/zaustavljanje snimanja. Po mogućnosti osigurati rad na Windows i Linux sustavu. Driveri za plovila moraju biti u skladu s driverima koji su razvijeni za ostala plovila u LAPOST-u.

## Program za kalibraciju kompasa (Daniel Zima)

Potrebno je izraditi C++ program koji će odrediti faktor skaliranja i DC pomak mjerenja magnetskog senzora (hard-iron calibration). Program treba uzorkovati veći broj mjerenja iz senzora i temeljem njih odrediti korekcijske parametre. Potrebno je istražiti literaturu na temu auto-kalibracije senzora. Implementaciju postupka potrebno je testirati sa Microstrain IMU-om i OceanServer kompasom.

## Spajanje CTD mjerne sonde s LinkQuest modemom (Mato Tomić)

Potrebno je osigurati spajanje CTD mjerne sonde s LinkQuest hidroakustičkim modemom. Komunikcija s modemom je već ostvarena i u ovom zadatku je pootrebno osigurati da se podaci dobiveni iz sonde šalju putem modema korištenjem postojećeg softwarea. Na zahtjev korisnika (zahtjev se upućuje isto hidroakustičkim kanalom) potrebno je odgovoriti podacima koji su traženi.

# Voditelj studentskog tima

Nikola Stilinović

# Uvod

## ****Upravljanje i identifikacija modela broda CyberShip**** (Viktor Vladić i Ines Šoić)

Vlastite oscilacije su ponašanje koje je karakteristično za nelinearne sustave i sastavni su dio svega što se giba (npr. avioni, brodovi, itd). Također, koriste se pri identifikaciji parametara prijenosne funkcije proizvoljnog reda i oblika te za identifikaciju parametara određene klase nelinearnih sustava. U ovom radu je bilo potrebno pomoću vlastitih oscilacija izračunati parametre PID regulatora za dinamičko pozicioniranje broda.

## Nastavak razvoja platforme PlaDyPos (Berislav Marszalek, Nikola Stilinović, Zoran Triska)

Oko 70% površine Zemlje je prekriveno oceanima i predstavlja beskrajno prostranstvo prirodnih resursa. Život je nastao u vodi, a neki znanstvenici prognoziraju u dalekoj budućnosti život u dubini kao jedini mogući. Kako bi optimalno iskoristili resurse koji su nam trenutno na raspolaganju nužno je razvijati i koristiti, kako ronilice tako i površinska plovila. Podvodna robotika, kao posebna grana robotike je područje koje se neprestano razvija zahvaljujući napretku tehnologije. Primjenom znanja o kontroli i automatizaciji procesa, podvodna robotika s raznim bjektima postaje zanimljivo područje za upravljanje i vođenje. Tema ovog dijela dokumentacije je upravo iz tog područja i vezana je uz jedan relativno novi i zanimljiv koncept upravljanja. Praćenje putanje korištenjem virtualnog cilja u području zemaljskih vozila već je dosta istraženo što nije slučaj za plovila. Oceani i jezera predstavljaju jedno od najtežih okruženja za istraživanja. Razvoj pouzdanih i robusnih plovila skupa sa naprednim algoritmima upravljanja čovjeku olakšava istraživanje tih nepreglednih prostranstava i pri tome ne ugrožava njegov život.Podvodna robotika je još uvijek mlada znanost i još dosta vremena će proći dok robotska vozila ne budu mogla sama lutati po prostranstvima plavetnila i obavljati sve zadatke umjesto čovjeka. Način upravljanja izložen u ovom radu je još jedan mali korak prema ostvaranjenju tog cilja.

## Komunikacija sa USBL modulom (Dean Ivošević)

Jedna od misija koja će se morati obavljati je lociranje i praćenje izvora signala. Da bi se ova misija mogla ostvariti, koristiti će se USBL tehnologija. Ova nam tehnologija omogućava dobivanje ažurnih i točnih podataka za ostvarenje našeg cilja.

Podatke koje *USBL* sustav računa, potrebo je preuzeti i spremiti u bazu podataka na taj način da se kasnije ti podaci mogu koristiti za obavljanje misije od strane platforme. Osim preuzimanja podataka, provjeravati će se točnost preuzetih podataka, te će se oni grafički prikazivati.

## Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

Fascinantni svjetovi prepuni boja i detalja koji će fotografskom oku ostati neotkriveni bez posebne opreme: podvodna fotografija ubraja se među posljednje velike avanture rada modernih aparata i pruža motive ljepote koja oduzima dah – ukoliko imate pravu opremu i neke osnovne vještine.

U ovom radu je objašnjen rad GoPro HD Hero kamere koja je zbog svojih malih dimenzija, dugotrajnog rada baterije, visoke rezolucije i mogućnosti montaže na skoro bilo kakvu podlogu, gotovo idealna za podvodno snimanje i slikanje. Zbog specifikacija kamere dobivamo kvalitetnu i stabilnu sliku čak i pri brzim promjenama položaja kamere (tj. na primjer ronilice na koju je montiramo).

## Program za kalibraciju kompasa (Daniel Zima)

U današnje vrijeme se u navigaciji i dalje često koristi kompas, osobito u pomorskom i zračnom prometu. Nakon nekog vremena pod utjecajem elektromagnetskih smetnji iz okoline može doći do manje ili veće pogreške u pokazivanju kompasa i zbog toga ge je potrebno kalibrirati.

U sklopu ovog projekta potrebno je izraditi C++ program koji će odrediti faktor skaliranja i DC pomak mjerenja magnetskog senzora (hard-iron calibration). Program treba uzorkovati veći broj mjerenja iz senzora i temeljem njih odrediti korekcijske parametre. Podaci se ne uzimaju direktno iz senzora nego se čitaju iz datoteke u koju su prethodno zapisana mjerenja senzora. Potrebno je i istražiti literaturu na temu auto-kalibracije senzora. Implementaciju postupka potrebno je testirati sa Microstrain IMU-om i OceanServer kompasom.

## Spajanje CTD mjerne sonde s LinkQuest modemom (Mato Tomić)

U svrhe podvodnih istraživanja, ovaj seminarski zadatak opisuje mogućnosti spajanja mjerne sonde Minos CTD sa LinkQuest modemom preko odgovarajuće upravljačke jedinice. Cilj zadatka je uspostaviti komunikaciju mjerne sonde, koja će se nalaziti pod vodom, sa centralnom jedinicom koja je stacionirana na površini vode (platforma, brod i sl.).

Minos CTD je mjerna sonda koja služi za podvodna istraživanja i njome se mogu mjeriti karakteristike vode kao što su temperatura, tlak, dubina, gustoća, slanoća i sl. Može se koristiti kao samostalni instrument na podvodnom vozilu, ali i kao uređaj za motrenje u stvarnom vremenu (engl. *real time*).

LinkQuest (UMW2000) je podvodni akustički modem koji služi za ostvarivanje komunikacije nekog uređaja koji se nalazi pod vodom sa drugim koji je na površini. Za prijenos informacija koristi se zvučnim signalima. Sastoji se od dva dijela pomoću kojih ostvaruje bežičnu komunikaciju koja je omogućena u oba smjera.

# ****Upravljanje i identifikacija modela broda CyberShip**** (Viktor Vladić i Ines Šoić)

## VLASTITE OSCILACIJE

Oscilacije sustava služe za identifikaciju parametara regulatora. Posebno se koriste za autotuning identifikaciju. Kod prvotnih algoritama za autotuning identifikaciju, korištenjem proporcionalnog dijela regulatora, oscilatorni režim rada se postizao dovođenjem sustava na sam rub stabilnosti. Tada bi se korištenjem pojačanja regulatora i frekvencije uspostavljenih oscilacija odredili parametri PID regulatora (Ziegler-Nichols metoda). Ova metoda baš i nije zahvalna upravo zbog dovođenja sustava do ruba stabilnosti što može biti opasno po sustav te se stoga ova metoda zamjenjuje primjenom nelinearnih elemenata u sustavu. Oscilacije dobivene korištenjem nelinearnih elemenata nazivaju se vlastite oscilacije. One su stalne i stabilne i to ponašanje je karakteristično samo za nelinearne sustave.

## IZRAČUNAVANJE PARAMETARA

Prijenosna funkcija procesa jest



T i K smo dobili preko vlastitih oscilacija (identificirali smo proces).

Prijenosna funkcija regulatora jest



Na temelju ove dvije prijenosne funkcije dobije se prijenosna funkcija zatvorenog regulacijskog kruga.



Željena modelska funkcija sustava je binomna funkcija trećeg reda, oblika:



Na temelju željene modelske funkcije i stvarne prijenosne funkcije sustava došli smo do parametara regulatora koji iznose







Implementacija izračunavanja parametara PID regulatora te sama implementacija PID regulatora prikazana je u sljedećem poglavlju.

## Implementacija u LabVIEW-u

Kod dinamičkog pozicioniranja napravljena je mogućnost manualnog upravljanja brodom te upravljanje preko parametara dobivenih pomoću vlastitih oscilacija. Omogućeno je zasebno upravljanje po napredovanju, zaošijanju i zanošenju.

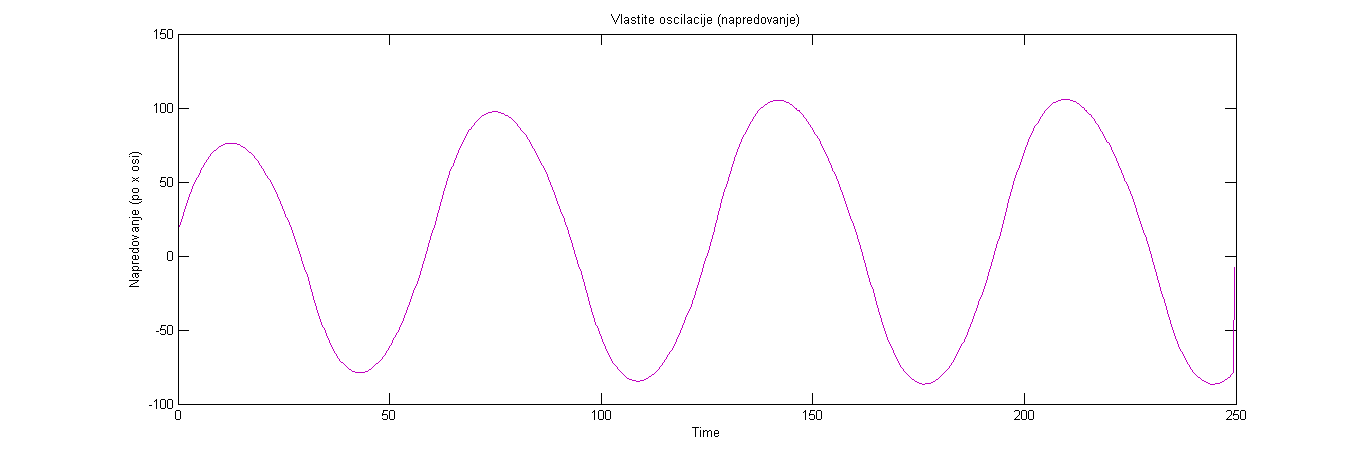
Prije same implemetacije u LabVIEW-u, bilo je potrebno mapirati thrustere, napraviti inverznu alokaciju aktuatora te inverznu rotacijsku matricu kako bi dobili koordinate u brodu te na taj način mogli regulirati brod.

U nastavku je prikazan izgled vlastitih oscilacija, parametri procesa i regulatora koji se izračunaju na temelju njih.

Tablica ‑ Surge (napredovanje)

|  |  |  |
| --- | --- | --- |
| **Karakteristike histereze** | **Konstante PID regulatora** | **Konstante procesa** |
| Ref. Vrijednost =0 | Kp=0.00017 | K=246.058 |
| Širina histereze=25 | Ki=1.18733e-6 | T=36,509 |
| Visina histereze=0.1 | Kd=0.00484 |
| Broj oscilacija=2 |

Na slici 5-1 su prikazane vlastite oscilacije po napredovanju.

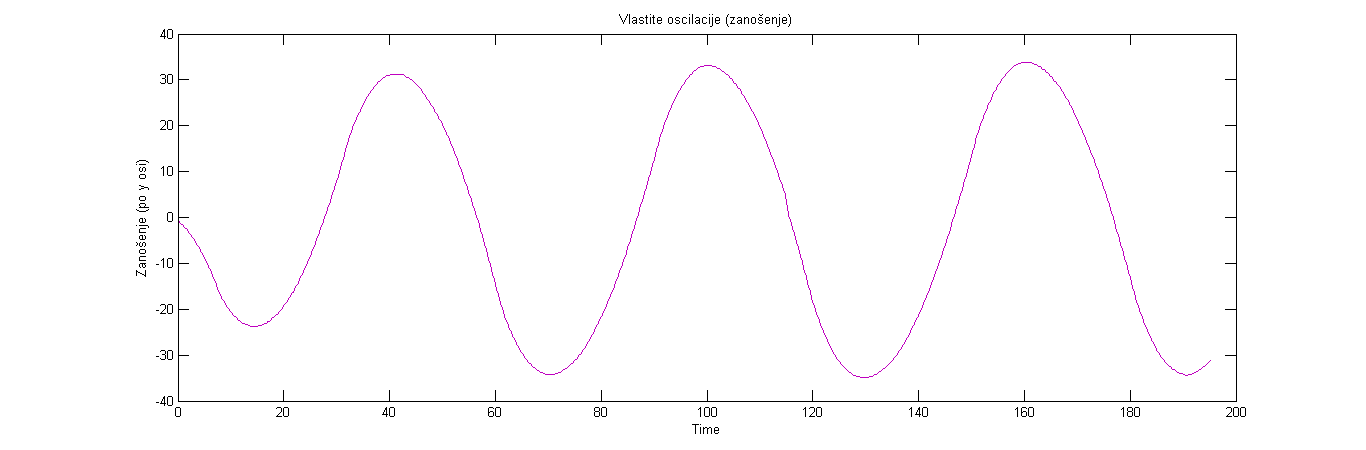


Slika ‑ Vlastite oscilacije (surge)

Prvu oscilaciju odbacujemo jer je nepravilna tj. nju ne koristimo pri izračunu parametara procesa i PID regulatora, već koristimo dvije pravilne oscilacije.

Tablica ‑ Sway (zanošenje)

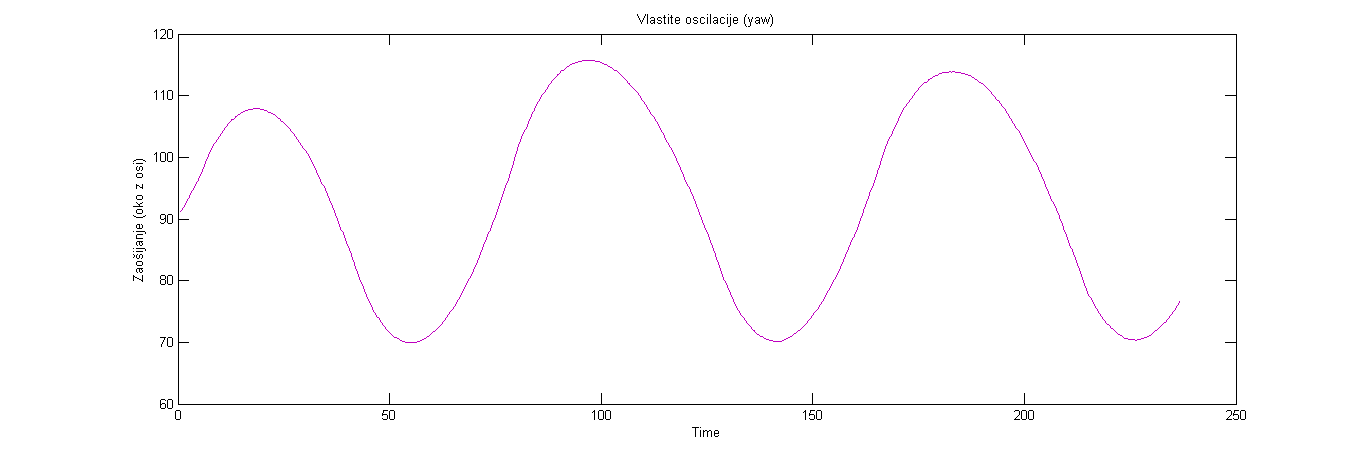
|  |  |  |
| --- | --- | --- |
| **Karakteristike histereze** | **Konstante PID regulatora** | **Konstante procesa** |
| Ref. Vrijednost =0 | Kp=0.000724 | K=29.131 |
| Širina histereze=15 | Ki=4.83261e-6 | T=17.5973 |
| Visina histereze=0.2 | Kd=0.00191 |
| Broj oscilacija=2 |



Slika ‑ Vlastite oscilacije (sway)

Tablica ‑ Yaw (zaošijanje)

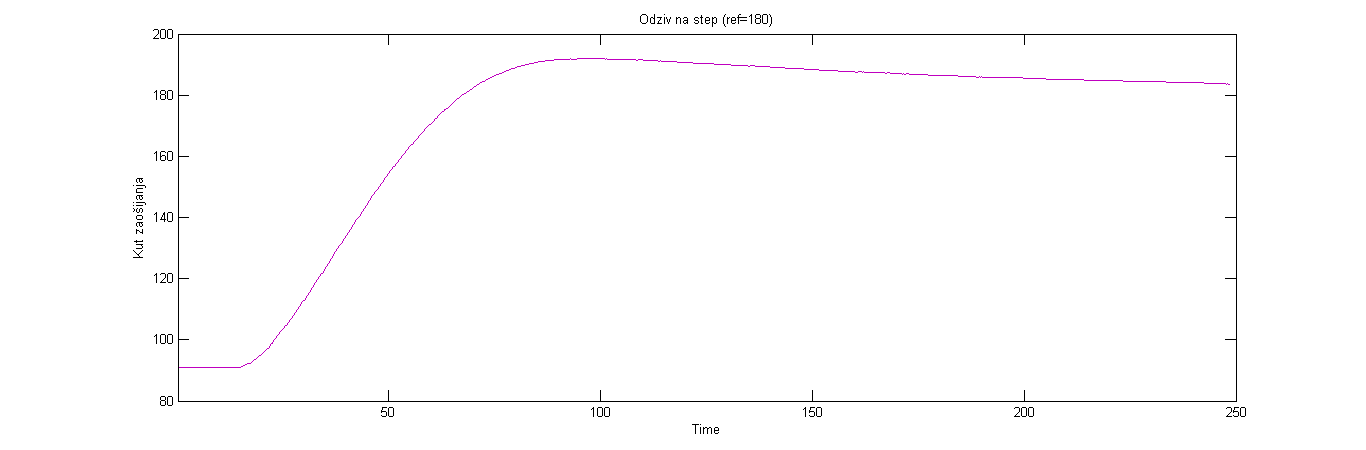
|  |  |  |
| --- | --- | --- |
| **Karakteristike histereze** | **Konstante PID regulatora** | **Konstante procesa** |
| Ref. Vrijednost =90° | Kp=0.001113 | K=146.2925 |
| Širina histereze=10° | Ki=7.4226e-6 | T=8.6870 |
| Visina histereze=0.15 | Kd=0.020715 |
| Broj oscilacija=2 |



Slika ‑ Vlastite oscilacije (yaw)

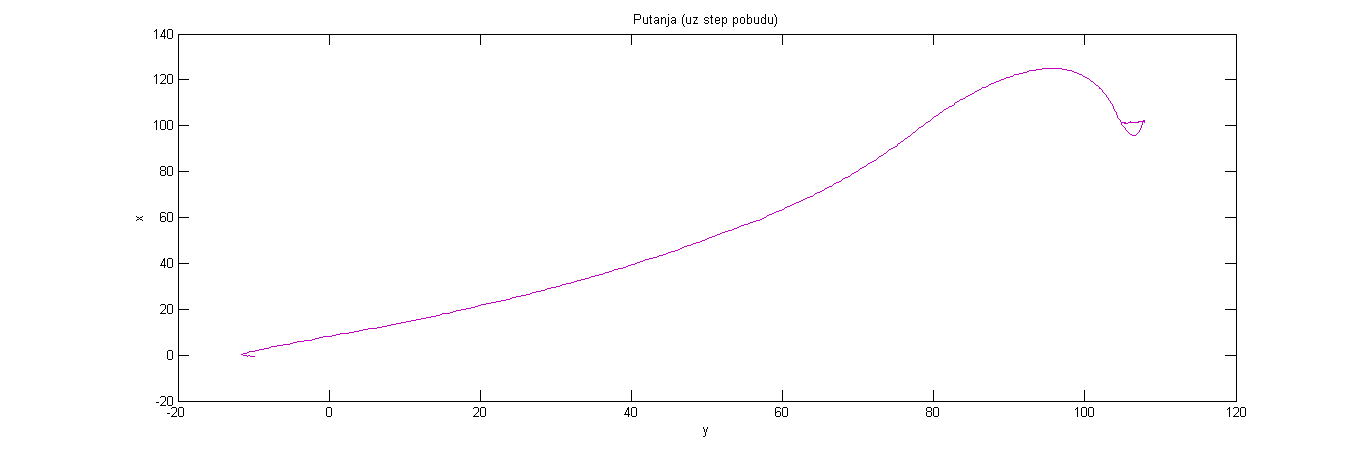
Nakon što smo dobili parametre PID regulatora napravili smo testove na skokovitu pobudu kako bi vidjeli rade li ovako dobiveni regulatori pravilno.

Na prvoj slici u nastavku je prikazan odziv kuta zaošijanja uz step pobudu sa 90° na 180°. Iz slike se može vidjeti da je rezultat zadovoljavajući. Postoji nadvišenje, ali nakon određenog vremena postigne se željena referentna vrijednost.

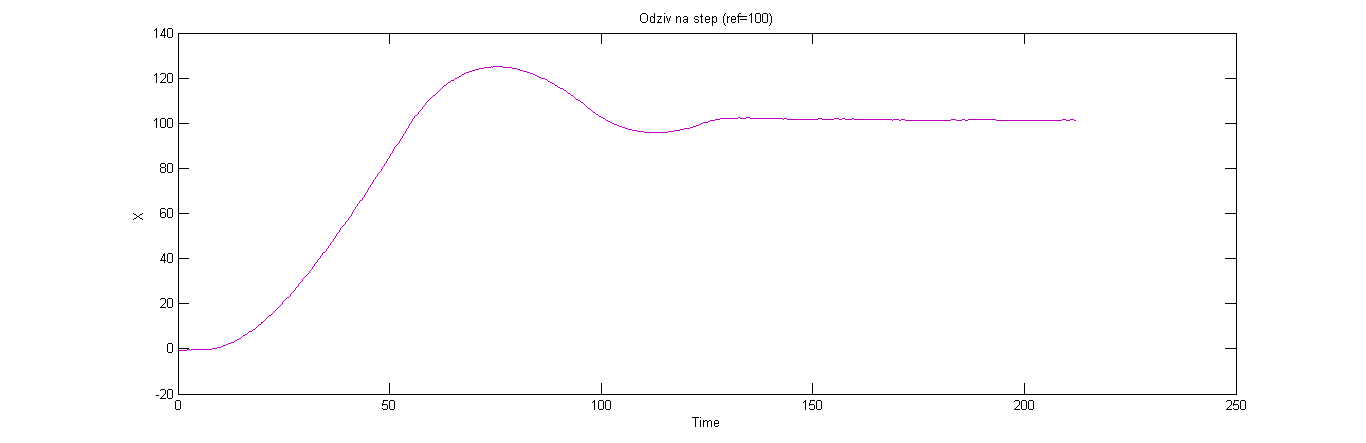


Slika ‑ Odziv na step (kut zaošijanja)

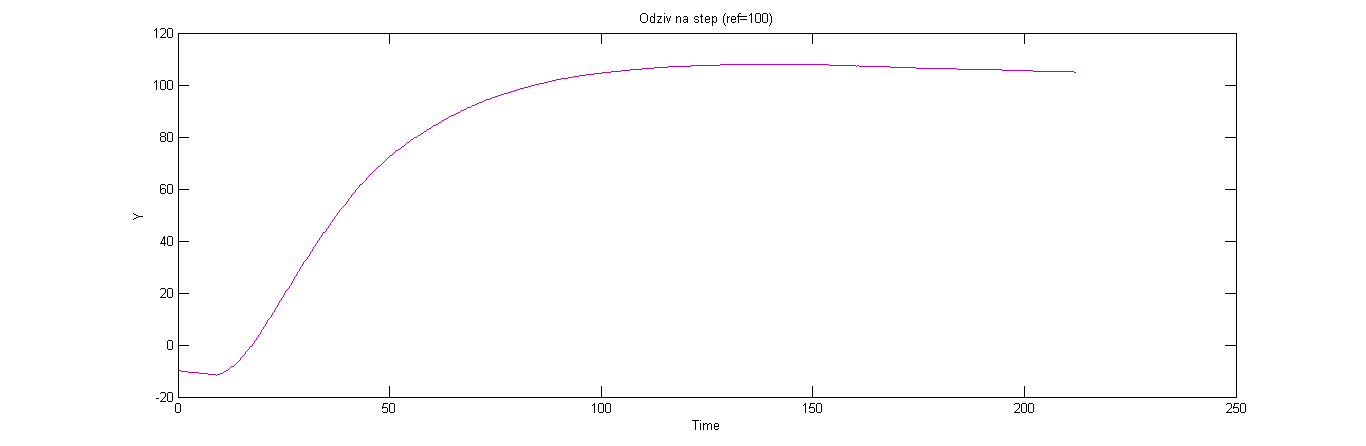
Nakon završetka ovog testa vratili smo brod u početnu poziciju (0,0,90°) te mu zadali da dođe na poziciju (100,100,90°). U nastavku su prikazane putanja za ovaj test te odzivi na pobudu po x i y osi.



Slika ‑ Putanja broda ze referencu (100,100,90°)



Slika ‑ Odziv na step pobudu (zaošijanje)



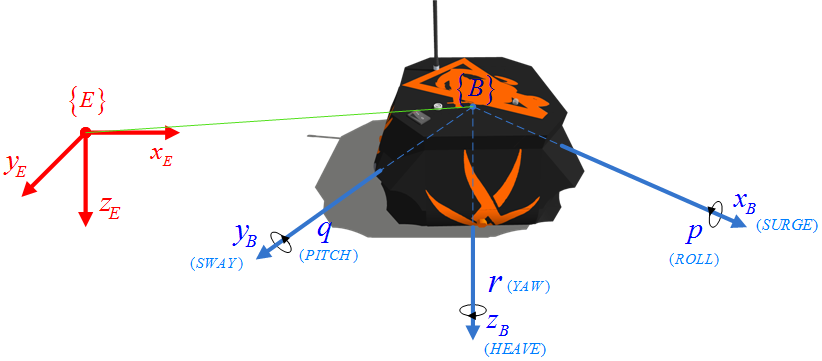
Slika ‑ Odziv na step pobudu (napredovanje)

Sa gornjih slika je vidljivo da regulatori rade zadovoljavajuće. Na skokovitu pobudu brod dođe u traženu poziciju. Dobili smo odziv kakav smo i htjeli tj. koji odgovara odzivu modelske funkcije.

# Nastavak razvoja platforme PlaDyPos (Berislav Marszalek, Nikola Stilinović, Zoran Triska)

## Matematičko modeliranje laboratorijske platforme

Za definiranje matematičkog modela platforme potrebno je definirati dva koordinatna sustava: koordinatni sustav vezan za zemlju {*E*} koji je nepokretan i često se zove inercijski koordinatni sustav; i drugi vezan za plovilo {*B*}, a ishodište mu se obično nalazi u centru mase.



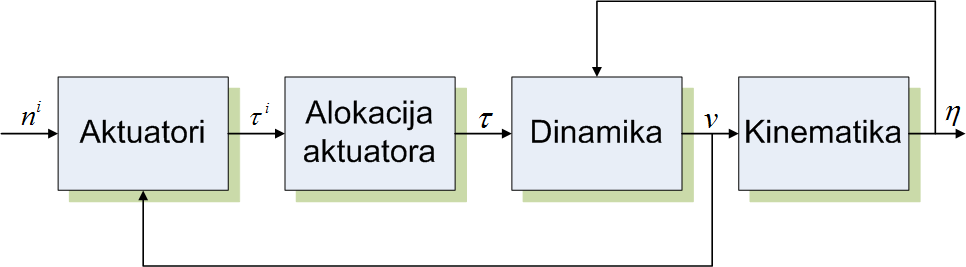
Slika ‑ Definirani koordinatni sustav

Promatrajući poopćeni model plovila može se uočiti 6 stupnjeva slobode pri čemu se poniranje, valjanje i zaranjanje može zanemariti limitirajući prostor gibanja na 2-dimenzionalni. Sve moguće varijable prikazane su u tablici 6-1.

Tablica ‑ Notacija koja se koristi kod plovila

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Stupanj slobode | Surge | Sway | Heave | Roll | Pitch | Yaw | Definiran u |
| brzine |  |  |  |  |  |  | {B} |
| pozicija |  |  |  |  |  |  | {E} |
| Sile |  |  |  |  |  |  | {B} |

Sila napredovanja i sila zanošenja su definirane kao translacijsko gibanje u *x-* i *y-* smjeru {*B*} koordinatnog sustava. S druge pak strane, zaošijanje je definirano kao rotacija oko *z-* osi u {*B*} koordinatnom sustavu. Zemaljski koordinatni sustav se koristi za definiranje pozicije [x y]T i orijentacije . Na isti način u {*B*} koordinatnom sustavu definiraju se linearne brzine (napredovanje i zanošenje) te kutna brzina (zaošijanje). Samo gibanje platforme ostvaruje se primjenom određenog iznosa sila i momenata [X Y N]T koji se također definiraju u {*B*} koordinatnom sustavu.



Slika ‑ Blok dijagram matematičkog modela

Osnovni matematički model platforme može se prikazati rastavljen na nekoliko dijelova koji su prikazani slikom 6-2. To su blokovi koji opisuju aktuatore te njihovu alokaciju i zatim blokovi za dinamiku i kinematiku platformu. U ovom radu od velikog su interesa upravo zadnja dva bloka dok su prva dva manje bitna za samu ideju upravljanja, no bit će spomenuta radi potpunosti modela.

### Model potisnika

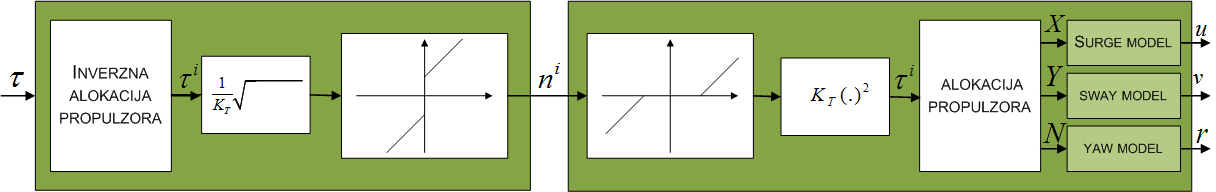
Potisnici su najčešći aktuatori koji pogone plovila, a sila koju razvijaju je u općenitom slučaju nekakva nelinearna funkcija ovisna o brzini vrtnje propelera. Sila koju daje potisnik može se opisati s

gdje su i pozitivne konstante, je brzina vrtnje propelera, a je okolna brzina vode koja može biti izražena s

gdje je takozvana *wake speed*, a *U* je apsolutna brzina platforme dana s

Ovakav model potisnika naziva se *bilinearnim*. Potisak je uvijek bolje opisan bilinearnim modelom, no ponekad se takav model može zamijeniti jednostavnijim *afinim* modelom.

Pojednostavljenje je opravdano ukoliko se platforma giba malim brzinama što je u ovom slučaju uvijek ispunjeno. Pri modeliranju potisnika važno je uzeti u obzir još jednu nelinearnu karakteristiku koja se javlja kod većine potisnika. Zbog trenja postoji pojas mrtve zone koji se treba identificirati i kompenzirati inverzom nelinearne funkcije. Osim toga, da bi dobili linearno upravljanje generiranim potiscima potrebno je unaprijed napraviti kompenzaciju kvadratne funkcije nakon inverzne alokacije.



Slika ‑ Alokacija propulzora i a-priori kompenzacija

### Alokacija aktuatora

Alokacija aktuatora je veza između sila i momenata koje proizvode aktuatori i sila i momenata koje djeluju na tijelo koje se giba. Ova veza između dviju vrsta sila najčešće se opisuje alokacijskom matricom čije dimenzije ovise o broju aktutora i broju stupnjeva slobode gibanja. Budući da smo model ograničili na tri stupnja slobode, a platforma posjeduje četiri aktuatora smještena u standardnoj *x-* konfiguraciji prikazanoj na slici 6-4, alokacijska matrica je

oblika:

Četiri aktuatora se koriste za ostvarivanje tri stupnja slobode pa je jasno da postoji redudancija u sustavu. Ona omogućava projektiranje raznih upravljačkih algoritama u slučaju kvara pojedinog aktuatora. Inverzna alokacijska matrica ne može biti određena jer se radi o matrici koja nije kvadratna, no zato se može izračunati pseudoinverzna matrica koja je po svojstvima najbliža inverznoj matrici.



Slika ‑ Platforma i x-konfiguracija propulzora

### Dinamika

Dinamički model platforme je izrazito spregnut i nelinearan. Donekle zahvaljujući *x -k*onfiguraciji potisnika može se svaki stupanj slobode promatrati odvojeno te je opravdano učiniti model raspregnutim. Pojednostavljenje je naročito prikladno za projektiranje regulatora jer znatno olakšava posao. U općem slučaju, tri stupnja slobode gibanja se mogu opisati s:

gdje je vektor brzine je vektor poopćene sile koji djeluje na tijelo tijekom gibanja. Platforma je simetrična u horizontalnoj ravnini što povlači identičnu diferencijalnu jednadžbu za unaprijedno gibanje i zanošenje. Parametar je konstantan parametar i u oba slučaja je isti; i kao otpor ovisan o brzini gibanja koji sadržava sve hidrodinamički efekte. Na sličan način kod zaošijanja je inercijski parametar, a hidrodinamički otpor.

### Kinematika

Kinematički model daje vezu između brzina u koordinatnom sustavu vezanom za plovilo {*B*} i prve derivacije pozicije i kuteva u Zemaljskom koordinatnom sustavu {*E*}. Budući da se platforma nalazi na površini vode potrebno je samo definirati rotacijsku matricu koja rotira varijable oko *z-* osi. Pozitivan smjer rotacije je definiran suprotno kazaljki na satu, a kut za koji se treba zarotirati sustav je označen sa koji je ujedno i orijentacija platforme.

## Praćenje putanje korištenjem virtualnog cilja

U prethodnim poglavljima definirana je dinamika i kinematika plovila koji treba slijediti nekakvu putanju. Dan je također uvod u nelinearni način upravljanja koji jamči asimptotsku stabilnost. U ovom poglavlju proširuje se sam koncept kinematičkog modeliranja s ciljem razvijanja upravljačkog algoritma kojem je glavni zadatak slijeđenje proizvoljne unaprijed definirane putanje.

### Model kinematičke pogreške

Prije nego što krenemo u samu problematiku potrebno je uvesti nove veličine kako bi se za potrebe sinteze regulatora moglo matematički opisati model.Kinematički model izrazit ćemo preko vektora totalne brzine

gdje je

označuju apsolutnu vrijednost brzine i njezinu orijentacije u *E* koordinatnom sustavu. Uz već definirane koordinatne sustave vezane za zemlju i za plovilo uvodi se još jedan koordinatni sustav, tzv. Serret- Frenet frame }. Taj koordinatni sustav vezan je uz točku P na putanji i giba se po njoj. Definirajmo još točku B kao centar mase plovila koje se poklapa sa ishodištem *{B}* koordinatnog sustava. Rješenje problema slijeđenja putanje koje je izloženo ovdje, temelji se donekle i na intuitivnom objašnjenjenju. Projektirani regulator bi morao obavljati dvije zadaće: računati udaljenost između centra mase plovila B i točke P, računati kut između vektora totalne brzine plovila i tangente u točki P na putanji i svoditi oboje na nulu. Ovo je glavni motiv za uvođenje i razvijanje kinematičkog modela u Serret-Frenet koordinatnom sustavu koji se giba duž putanje. Gibajući pokretni sustav, ilustrativno možemo shvatiti kao virtualnu pokretnu metu koja treba biti praćena stvarnim plovilom. Prethodno spomenuta udaljenost i kut postaju prostor kinematičke pogreške za koji treba kinematički formulirati i rješiti problem. Prilikom spominjanja udaljenosti plovila do ishodišta Serret-Frenet koordinatnog sustava, bitno je naglasiti da taj koordinatni sustav nije postavljen na način da je udaljenost minimalna, već je mjesto na kojem se nalazi rezultat definiranog upravljačkog algoritma. Na taj način uvodi se još jedan dodatni stupanj slobode pri projektiranju regulatora, a ujedno je rješen problem singularnosti koji se javljao u samim početcima razvoja ove metode. Prema slici 6-5 P je obična točka koju treba slijediti. Nalazi se u ishodištu Serret-Frenet koordinatnog sustava i giba se s njim po nekakvoj krivulji.

Krivulja je parametrizirana po parametru *s* koji predstavlja udaljenost po krivulji od početne točke krivulje za koju je *s = 0* i konačne točke u kojoj poprima vrijednost .



Slika ‑ Koordinatni sustavi i parametri

Koordinate točke *B* može se izraziti u zemaljskom koordinatnom sustava *{E}* kao ili kao u *{F}* koordinatnom sustavu. Veza među njima dana je

rotacijskom matricom s parametrom koji je kut koji zatvara tangenta u točki *P*.

Definirajući i označavajući sa *s* curvilinear abscisu duž putanje slijede izrazi:

gdje su sa i označene zakrivljenost i njezina derivacija. Brzina gibanja točke *P* u *{E}* može biti izražena u *{F}* kao

Isto tako, na jednoznačan način se može izračunati brzina točke *B* u *{E}*

gdje je *d* vektor usmjeren od točke *P* prema *B*. Množenjem izraza s lijeve strane s rotacijskom matricom R dobije se:

Uzevši u obzir relacije:

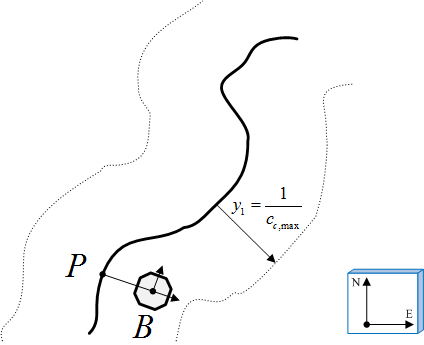
i vektorski produkt

Konačni model, uvrstivši sve izraze i uvodeći proizlazi kinematički model u (s,y) koordinatama kao

gdje je

U ovom trenutku važno je naglasiti da je točka *P* je definirana kao projekcija točke *B* na putanju, pretpostavljajući pritom da je projekcija dobro definirana. Taj kinematički model je jednak sa s1 = 0. Prva jednadžba u izrazu se tada rješava za jer je projekcijom s1

direktno postavljen na nula. u rješenju se pojavljuje singularitet za koji bitno ograničava početne uvjete prilikom pokretanja upravljačkog algoritma. Položaj točke B mora biti unutar polumjera koji je određen maksimalnom zakrivljenošću putanje



Slika ‑ Virtualna meta

Dozvolivši da s1 ne mora nužno biti jednak nuli, kreirana je virtualna meta koja se ne poklapa sa projekcijom plovila na putanju. Na ovaj način uvodi se dodatni stupanj slobode prilikom projektiranja regulatora. Upravljanje brzinom novodefinirane virtualne mete singularitet pri udaljenosti je uklonjen.

### Projektiranje nelinearnog kinematičkog regulatora

Sa opisanim kinematičkim sustavom u (s,y) koordinatama pred regulator se mogu postaviti sljedeći zadatci. Za zadanu putanju koju treba slijediti i za željenu unaprijednu brzinu treba odrediti upravljački algoritam za moment zaošijanja $N$ i brzinu napredovanja virtualne mete duž putanje da teže asimptotski prema nuli.

Pojednostavljeno gledano, zadatak se svodi na reduciranje k nuli udaljenosti između točke *P* i *B* i kuta β koje je razlika između vektora brzine *U* i tangente u točki *P*. Glavna ideja predstavljena u ovom odjeljku je razmatranje pokretne mete pridružene Serret- Frenet koordinatnom sustavu koja se giba po definiranom upravljačkom algoritmu. Cilj nam je odrediti zakon po kojem se virtualna meta giba i ostvariti gore navedene specifikacije. Prvo je potrebno definirati kut prilaza φ(y1) kao funkciju udaljenosti plovila do tangente na

krivulju u točki *P* izražene u koordinati y1. Kut prilaza je uobičajen u oblikovanju prijelaznih manevara pri fazi približavanja putanji. Potrebno je odabrati funkciju koja ima svojstva zasićenja te zadovoljava neka ograničenja.

Uvjeti osiguravaju usmjeravanje plovila prema putanji i zadržavanje na njoj jednom kad dođe do nje. Tangens hiperbolni sa podesivim parametrom pokazao se kao pogodna funkcija za

realizaciju prilaženja putanji. je kut pod kojim se plovilo giba u odnosu na virtualnu metu kad se nalazi na velikim udaljenostima od nje. Taj parametar treba odabrati u rasponu \

S izborom takve funkcije kutu prilaza β se nameće da prati željeni φ. Potrebno je pokazati asimptotsku stabilnost prilaženja krivulji, a samo razmatranje Lyapunovljevog kandidata će polučiti kakav treba biti upravljački signal da bi ostala zadržana stabilnost. Za Lyapunovljevu funkciju prikladno je izabrati kvadratnu formu oblika.

Vremenska derivacija iznosi:

Potrebno je izabrati upravljački algoritam takav da vremenska derivacija Lyapunovljeve funkcije bude negativna pa odabirom za:

je ostvareno upravo s k1 > 0. S ovim je zajamčena konvergencija kuta platforme prema kutu virtualne mete kako se približavamo putanji. Daljnje promatranje kinematičkog upravljanja vodi nas na razmatranje druge Lyapunovljeve funkcije vezane uz sam proces gibanja. I u ovom slučaju formirani kandidat je kvadratičnog oblika.

Računajući vremensku derivaciju dobivamo

Primjećuje se da je jedini podesivi stupanj upravo brzina virtualne mete koja može biti kontrolirana da osigura konvergenciju plovila na željenu putanju. Izborom za

gdje je proizlazi i time je pokazana asimptotska konvergencija na željenu putanju sa pravilnom orijentacijom. Nelinearni regulator će ispunjavati sve zadatke koje smo na početku poglavlja stavili pred njega. Nelinearni kut prilaza dopušta mijenjanje kuta upada s udaljenošću u odnosu prema putanji. Kad je udaljenost velika, orijentacija platforme je postavljena na Kako se smanjuje tako se i kut upada smanjuje i teži prema nuli kako postaje jednak nuli, odnosno kad se platforma nalazi na putanji. Još jednom se naglašava, princip virtualne mete,označen sa uvodi dodatni stupanj slobode koji se kontrolira izrazom i pri tome se uklanja bilo koja moguća pojava singularnosti koja bi značajno ograničila cijeli upravljački algoritam.

### Proširenje na dinamiku

S razvijenim kinematičkim upravljanjem ide se korak dalje te se dobiveni model proširuje na dinamiku. Plovilo kao realan sustav giba se pod utjecajem sila i momenata koji su narinuti na njega pa se ovo proširenje smatra opravdanim u smislu ostvarivanja konačnog upravljanja. Razvijene virtualne kinematičke algoritme upravljanja može se promatrati kao nadređenu petlju koja generira referencu za niži stupanj kontrole koji se odnosi upravo na iznos momenta zaošijanja koji treba primijeniti da se izvrši željeno gibanje. I u ovom slučaju

promatrat ćemo Lyapunovljevog kandidata s kojim bi integrirali samo dinamičko

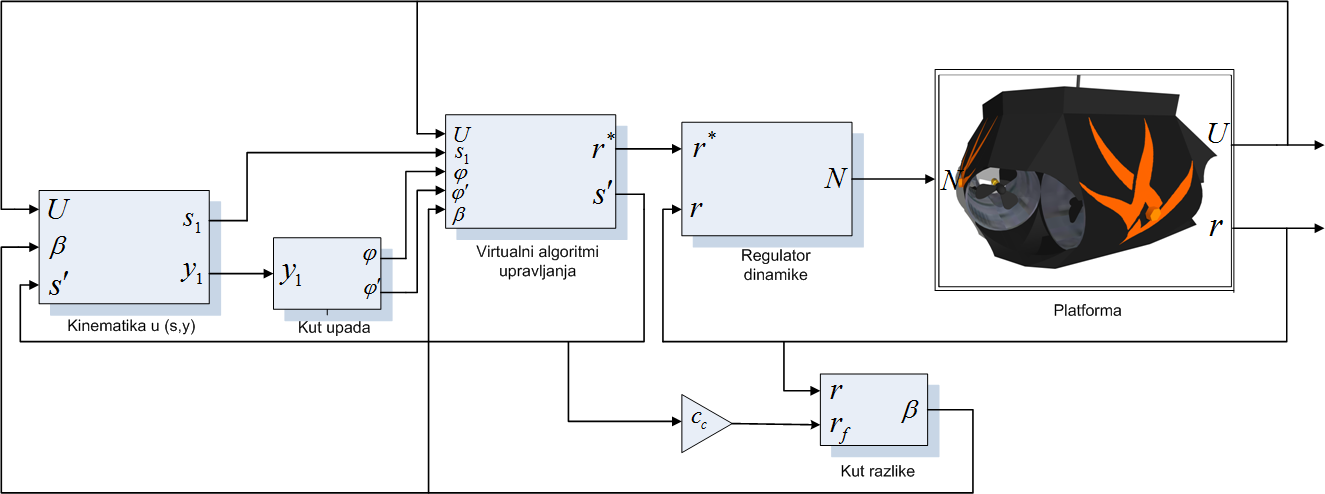
ponašanje s prethodno definiranim virtualnim algoritmima.

Vremenska derivacija funkcije iznosi

i cilj nam je svesti ju na oblik koji će cijelo vrijeme biti negativan uz Dinamika samog procesa zaošijanja dana je s $(2.8)$ i uz zanemarenje smetnje i nemodelirane dinamike, kutna akceleracija može biti izražena kao:

Uvrstivši prethodno napisani izraz u $(4.24)$ dobije se upravljački signal za *N* koji osigurava negativnost derivacije Lyapunovljeve funkcije.

gdje je podesivi parametar. S ovim je sustav upravljanja dan u cijelosti. Nad kinematičkim model u Serret-Frenet koordinatnom sustavu proračunavaju se virtualni algoritmi upravljanja koji se poslije uzimaju u obzir prilikom nižeg nivoa kontrole koju obuhvaća samu dinamiku platforme. Principijalna shema ostvarenog sustava je prikazana na slici:



Slika ‑ Principijalna shema sustava upravljanja

## sbRIO-9642

sbRIO-9642 je digitalno/analogni sustav tvrtke National Instruments koji je dobiven za testiranja. Sastoji se od real-time procesora i FPGA. U ovom projektu je zadatak implementirati sbRIO-9642 u platformu i povezati ga sa single board kompjuterom koji komunicira sa sbRIO-m tako da mu šalje podatke sa USBL-a, GPS-a, kompasa i ostalih navigacijskih uređaja te nakon toka vrši obradu svih podataka i upravlja platformom. Zbog real-time procesora, moguće je upravljati platformom s malim vremenom diskretizacije što ne bi bilo moguće da se cijeli upravljački sustav pokreće sa single board kompjutera. FPGA dio je trebao poslužiti da se ostvari direktna veza između sbRIO-a i GPS-a i kompasa, ali zbog USB veze koju navedeni uređaji koriste, nije moguće direktno spajanje na sbRIO pa se zbog toga podaci šalju preko etherneta sa single board kompjutera.

## Veza sa platformom

Platforma koristi nekoliko komunikacijskih kanala za povezivanje sa plovilima ili korisnikom. Pomoću USBL modula, moguće je ostvariti vezu između ronioca ili ronilice. Komunikacija sa CyberShipom je ostvarena pomoću XBee modula; platforma prima GPS lokaciju i orijentaciju CyberShipa i istovremeno može upravljati njime. Na kraju, ostvarena je WLAN veza sa korisnikom na kopnu preko koje je moguće nadzirati cijeli sustav i upravljati bilo kojim plovilom.

# Komunikacija sa USBL modulom (Dean Ivošević)

## USBL

USBL (*Ultra-short baseline*) je metoda podvodne akustičke lokalizacije. *USBL* sustav sastoji se od tri dijela: transpondera, primopredajnika i kontrolne jedinice, što je vidljivo na slici (Slika 7-1).

Primopredajnik se montira na dno broda, odnosno u našem slučaju na dno platforme, tako da je uronjen u vodu. Njegova zadaća je komuniciranje i primanje podataka od transpondera.

Transponder komunicira, odnosno odašilje podatke koje prima primopredajnik, a uglavnom se nalazi montiran na objektu koji se želi pratiti i s kojim se želi komunicirati poput ronioca ili *ROV* ronilice.

Kontrolna jedinica je računalo, koji je centralni dio *USBL* sustava, a zadaća je obrada svih podataka primljenih iz primopredajnika. Osim toga, služi kao posrednik između cjelokupnog sustava i korisnika sustava. Kontrolna jedinica je „suhi“ dio sustava i uglavnom se nalazi na brodu, odnosno, u našem slučaju, u platformi.

Računalo, odnosno kontrolna jedinica, preuzima podatke iz primopredajnika, obrađuje ih te računa sljedeće podatke:

* Udaljenost između primopredajnika i transpondera
* Smjer od primopredajnika prema transponderu
* Dubina primopredajnika
* Dubina transpondera
* Geografska dužina i širina

*USBL* sustav koji će se koristiti je *Tritech MicroNav* i prikazan je na slici (Slika 7-1).

****

Slika ‑ USBL sustav marke Tritech

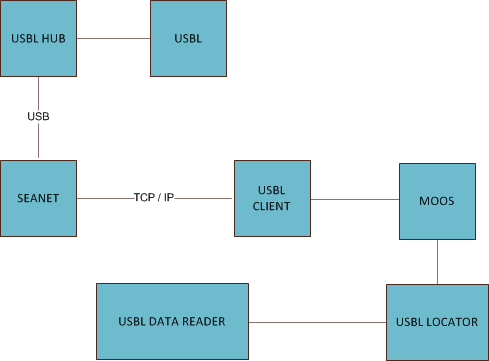
## Čitanje podataka sa *USBL* modula

Čitanje podataka sa *USBL* modula moguće je ostvariti na dva načina. Prvi način je korištenjem *SeaNet* programa na kojemu se mogu vidjeti svi podaci koje *USBL* računa, te su i grafički prikazani za lakše predočenje korisniku. Mana ovakvog prikupljanja podataka je to što se podaci koji se prime od strane *USBL* modula ne mogu koristiti, već se oni samo ispisuju na zaslon, a jedini način čuvanja podataka je spremanje cijelog mjerenja u obliku *log* datoteka. Ovakav način pogodan je kada se sustavom koristi čovjek, a ne drugo računalo. Pošto podatke iz *USBL* modula treba koristiti platforma za donošenje odluka i upravljanje, ovakav način preuzimanja nije moguće primijeniti u našem slučaju.

Drugi način je preuzimanje pojedinačnih podataka na udaljeno računalo putem lokalne mreže. Ova komunikacija odvija se pomoću *TCP/IP* protokola, te omogućuje mobilnost i neovisnost računala. Prednost ove metode je to što se svaki izmjereni podatak može zasebno koristiti u daljnjim proračunima i zasebno spremati u bazu podataka platforme.

Radi lakšeg predočenja i objašnjenja, čitav proces preuzimanja podataka je podijeljen u elemente. Svaki element ima svoju svrhu i ulogu, a njihova shema komunikacije i međuovisnosti prikazana je na slici (Slika 7-2).

U daljnjim poglavljima pobliže će se opisati svaki element ove sheme.



Slika ‑ Shema komunikacije i međuovisnosti

### USBL i USBL HUB

O ova dva elementa pisalo se u prethodnim poglavljima, tako da će ovdje biti manje obrađena.

Element *USBL* predstavlja transponder i primopredajnik u *USBL* sustavu, dok *USBL HUB* predstavlja kontrolnu jedinicu u *USBL* sustavu.

### SEANET

*SeaNet* element predstavlja vezu između *USBL* sustava i računala koji želi preuzeti podatke. Osim toga na ovom elementu u sustavu kasnije će se pokretati snimke snimljene u realnim uvjetima od strane djelatnika labosa.

### USBL CLIENT

*USBL Client* je dio složenog i povezanog programskog repozitorija koji su napisali djelatnici LABUST-a. Njegova zadaća je očitavanje podataka iz *Seaneta* koje *USBL* modul šalje, tako da se omogući daljnja obrada i korištenje istih podataka.

*USBL Client* se pomoću *TCP/IP* veze povezuje na računalo na kojem je pokrenut *Seanet* te od njega preuzima podatke. Ovakvom komunikacijom omogućila se mobilnost i neovisnost uređaja na kojem se nalazi *Seanet* i uređaja na kojem su potrebni podaci iz *USBL* modula.

Prilikom pokretanja *USBL Client* otvara konfiguracijsku *xml* datoteku iz koje očitava podatke potrebne za ostvarivanje *TCP/IP* veze s uređajem na kojem je pokrenuti *Seanet*. *Xml* datoteka naziva se *basic\_config.xml* i u nju je prije pokretanja programa potrebno napisati ili izmijeniti adresu računala na koji se želimo spojiti. Osim adrese računala, u *xml* datoteci, nalaze se i parametri komunikacije, ali oni su predefinirani i uvijek isti, pa ih stoga nije potrebno mijenjati.

Nakon što se komunikacija sa *Seanetom* ostvari, *USBL Client* konstantno preuzima podatke koje *USBL* očitava te ih zapisuje u *MOOS*, što omogućuje daljnju obradu preuzetih podataka.

### MOOS

MOOS (*Mission Orientated Operating Suite*) je platforma razvijena u programskom jeziku C++, zamišljena kao posrednik pri istraživanjima iz područja robotike. Njeno je podrijetlo vezano uz misije koje se odnose na more i podmorje, te stoga sadrži pojedine aplikacije koje se i danas koriste u zadaćama vezanim za taj medij.

*MOOS* ima zvjezdoliku topologiju, što znači da svaka aplikacija, tj. klijent u *MOOS* okruženju ima jedinstvenu vezu sa *MOOS* bazom podataka (*MOOSDB*).

*USBL Client* i *USBL Locator* uspostavljaju vezu sa *MOOS* bazom podataka, te na taj način postaju njegovi klijent. Kao klijenti sustava, mogu upotrebljavati sve funkcije *MOOS-a*, pa tako s jedne strane izvršavati pisanje podataka u bazu, a s druge strane čitati podatke iz baze.

### USBL LOCATOR

Uloga elementa *USBL Locator* je čitanje željenih podataka iz *MOOS* baze podataka, njihova obrada, te ispis na zaslon. *USBL Locator* napravljen je u programskom jeziku C++, tako da bude kompatibilan i da može iskorištavati već napisane dijelove repozitorija labosa.

Prilikom pokretanja *USBL Locator* otvara i čita konfiguracijsku *xml* datoteku, u kojoj se nalaze podaci za uspostavljanje veze sa *MOOS* bazom podataka te nazivi parametara koji će se preuzimati.

Primjer konfiguracijske *xml* datoteke može se vidjeti u nastavku:

<?xml version="1.0"?>

<configurations>

<config type="program" name="pUSBLLocator">

<param name="MOOSconfig" value="pLocator"/>

<param name="GPSLabel" value="GPS\_data"/>

<param name="USBLLabel" value="USBL\_data"/>

<param name="DodatnaPostavka" value="aaa"/>

<commsConfig name="pLocator" type="moos">

<param name="Hostname" value="localhost"/>

<param name="Port" value="9000"/>

<param name="ProcessName" value="USBLLocator"/>

<param name="AppTick" value="10"/>

<param name="CommsTick" value="10"/>

<param name="Subscription" value="GPS\_data"/>

<param name="Subscription" value="USBL\_data"/>

</commsConfig>

</config>

Dio programa koji se brine za pripremu i uspostavu veze s *MOOS* bazom ima sljedeći oblik:

…

\_xmlNode\* configNode = NULL;

std::string moosConfig = "", GPSLabel = "GPS\_data", USBLLabel = "USBL\_data";

double displayPeriod = 0.3;

if (reader.try\_value(configQuery, &configNode))

{

reader.useNode(configNode);

reader.try\_value("param[@name='MOOSconfig']/@value", &moosConfig);

reader.try\_value("param[@name='GPSLabel']/@value", &GPSLabel);

reader.try\_value("param[@name='USBLLabel']/@value", &USBLLabel);

}

reader.useNode(configNode);

LABUST::COMMUNICATION::GyrosMoosCommsInterface comms(reader, moosConfig);

…

Nakon uspostave veze sa *MOOS* bazom počinje neprekidno čitanje podataka iz baze podataka ostvareno beskonačnom *while* petljom:

…

comms.Receive(receivedData,true);

if(!receivedData.empty())

{

for(std::vector<LABUST::XML::GyrosReader>::iterator gyros = receivedData.begin(); gyros!=receivedData.end(); gyros++)

{

std::string label = gyros->GetLabel();

try

{

if(boost::iequals(label,GPSLabel))

{

gpsState.clear();

gyros->dictionary(gpsState);

}

else if(boost::iequals(label,USBLLabel))

{

usblState.clear();

gyros->dictionary(usblState);

}

}

catch (...)

{

}

}

receivedData.clear();

#### …

Gore napisanim kodom završava se primanje podataka, te započinje ispisivanje podataka na ekran. Ispisivanje se vrši u tekstualnom obliku u komandnom prozoru.

…

std::cout<<std::endl<<"USBL: "<<std::endl;

for(LABUST::VEHICLES::COREDATASET::stateMap::iterator stateElement = usblState.begin(); stateElement != usblState.end(); stateElement++)

{

std::cout<<LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first]<<"="<<stateElement->second<<std::endl;

…

}

std::cout<<std::endl<<"GPS: "<<std::endl;

for(LABUST::VEHICLES::COREDATASET::stateMap::iterator stateElement = gpsState.begin(); stateElement != gpsState.end(); stateElement++)

{

std::cout<<LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first]<<"="<<stateElement->second<<std::endl;

}

…

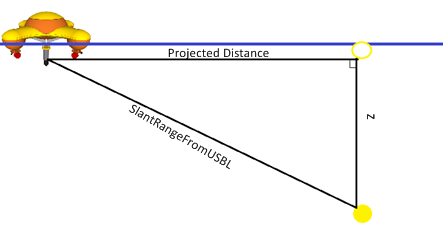
Podaci koji se ispisuju su sljedeći: z, latitude, longitude, SlantRangeFromUSBL, BearingFromUSBL, USBL\_yaw, USBL\_z, USBL\_lon, USBL\_lat, GPS\_lat, GPS\_lon. U tabeli (Tabela 1) može se vidjeti značenje i opis pojedinih podataka koji su važni za izvršavanje našeg zadatka.

Tablica ‑ Opis pojedinih podataka očitanih sa USBL modula

|  |  |
| --- | --- |
| **PODATAK** | **OPIS** |
| z | Dubina transpondera pod vodom |
| SlantRangeFromUSBL | Udaljenost između transpondera i primopredajnika |
| BearingFromUSBL | Kut između primopredajnika i transpondera |
| USBL\_z | Dubina uronjenosti glave primopredajnika u vodi |
| USBL\_yaw | Usmjerenje glave primopredajnika |

Podaci koji su nam važni za lociranje i praćenje transpondera su *z*, *SlantRangeFromUSBL* i *BearingFromUSBL*. Kako je već opisano u tablici, kut između platforme i našeg željenog cilja se odmah može očitati iz podataka preuzetih iz *USBL* modula. Da bi platforma uspjela pratiti i pozicionirati se iznad našeg cilja, potrebno je znati i udaljenost do istog. Iz podataka dobivenih iz *USBL* modula može se očitati prava udaljenost između platforme i cilja, međutim ta se vrijednost ne može primijeniti jer nije mjerodavna pošto se platforma može kretati samo po površini mora. Stoga se računa prividna udaljenost između platforme i cilja, koja će se dalje koristiti u proračunima.

Prividna udaljenost se računa tako da se pozicija transpondera (cilja) projicira na površinu vode, te se udaljenost računa naspram toj projiciranoj točki. Na slici (Slika 7-3) može se vidjeti princip računanja prividne udaljenosti.



Slika ‑ Prikaz prividne udaljenosti

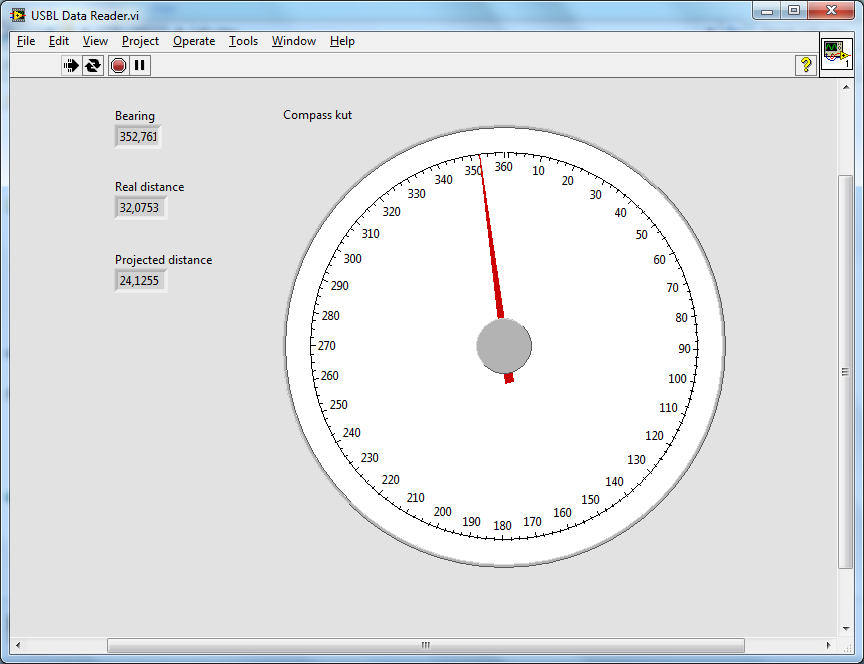
Kod računanja prividne udaljenosti korištena su jednostavna geometrijska pravila pravokutnog trokuta, a formula je:

### USBL DATA READER

*USBL Data Reader* je pomoćni element koji se koristi za kontrolu i prikaz preuzetih podataka.

Pošto se podaci u elementu *USBL Locatoru* iz baze podataka očitavaju velikom brzinom, ispisivanje na ekran je također brzo, pa je stoga nepregledno i komplicirano pratiti vrijednosti podataka. Zbog tog razloga napravljen je *USBL Data Reader* u programskom alatu *Lab View*. On preuzima podatke od elementa *USBL Locator*, te ih tekstualno i grafički iscrtava na ekran.

Primjer iscrtavanja podataka je prikazan na slici (Slika 7-4).



Slika ‑ Primjer ispisa elementa USBL Data Reader

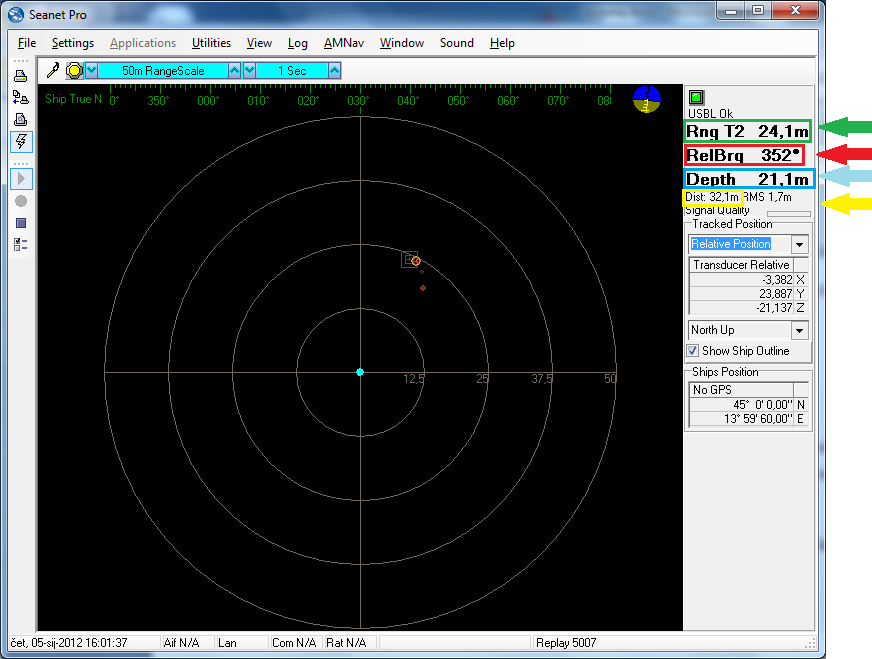
## Provjera rezultata

Da bi se provjerila točnost rada cijelog sustava napravljena je simulacija pomoću snimke stvarne uporabe *USBL*-*a* koju su snimili djelatnici labosa. Ova snimka sačuvana je u obliku *log* datoteke namjenjena za *SeaNet*, te je pokrenuta s istim.

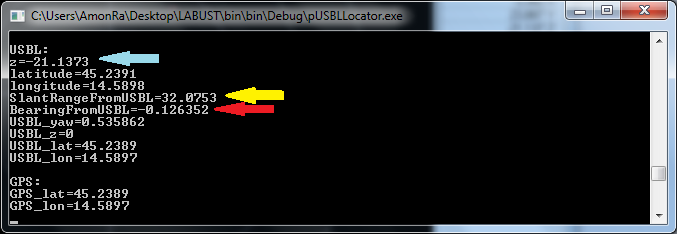
Slike ispod prikazuju ispis pojedinog dijela sustava za isti trenutak simulacije.

Na slici (Slika 7-5) vidi se prikaz *SeaNet* prozora na kojem je pokrenuta snimka *USBL* mjerenja. U gornjem desnom kutu prozora označeni su podaci koji će se uspoređivati.

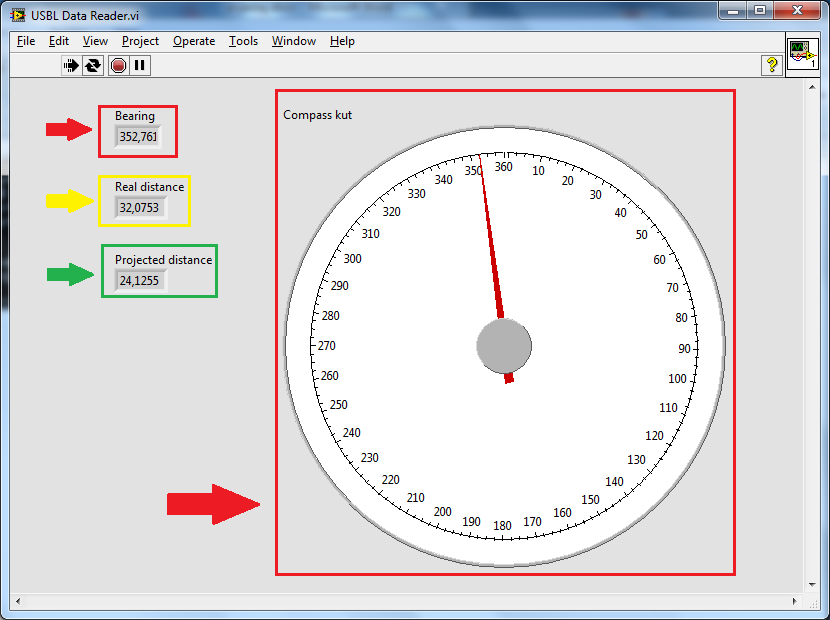
Podaci su markirani različitim bojama radi lakše usporedbe. Zelenom bojom označena je prividna udaljenost (*PrividnaUdaljenost*), crvenom bojom kut između primopredajnika i transpondera (*BearingFromUSBL*), plavom bojom dubina transpondera (*z*), te žutom bojom prava udaljenost između transpondera i primopredajnika (*SlantRangeFromUSBL*).



Slika ‑ Prikaz SeaNet ispisa u vremenu t=x simulacije



Slika ‑ Prikaz USBLLocator ispisa u vremenu t=x simulacije



Slika ‑ Prikaz USBL Data Reader ispisa u vremenu t=x simulacije

Na slici (Slika 7-6) može se vidjeti ispis podataka *USBL Locator* elementa. Može se primijetiti da se u ovom ispisu još uvijek ne nalazi prividna udaljenost, jer kao što je objašnjeno, ta vrijednost se ne dobiva direktno od *USBL* modula, već se mora naknadno računati. Osim toga, vidljivo je da se podatak *BearingFromUSBL* ne podudara s vrijednostima iz SeaNet i USBL Data Reader, jer su u ovom dijelu zapisane u radijanima.

Na slici (Slika 7-7) može se vidjeti ispis podataka *USBL Data Reader* elementa, te ako njih usporedimo sa početnim vrijednostima koje su vidljive u *SeaNet* programu (Slika 7-5), vidimo da je preuzimanje podataka napravljeno točno, a male pogreške koje su uočljive su rezultat zaokruživanja i pretvaranja iz radijana u stupnjeve.

# Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

## GoPro HD Hero kamera

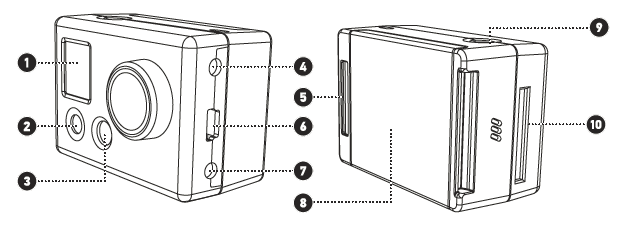
GoPro HD Hero kamera je nevjerojatan mali uređaj koji dolazi sa različitim dodacima koji se mogu montirati.

Prednosti ove kamere su dugotrajni rad baterije, stabilnost snimanja, dimenzije kamere, kvaliteta videa, jednostavna je za korištenje i postavljanje te ima vodotporno kučićte.   
Zbog svih ovih prednosti, kamera je idealna za podvodno slikanje ili snimanje.

Slika ‑ GoPro HD Hero kamera sa i bez vodootpornog kučišta

Dijelovi kamere su prikazani na *Slici 8-2*. Kamera sadrži gumb za paljenje i gašenje kamere (2), okidač za snimanje slika ili videozapisa (9), LCD ekran koji nam prikazuje stanje kamere te izbornik sa svim mogućnostima rada (1), LED signalno svjetlo (3), HDTV izlazni port (4), Hero BUS port (5), USB port (6), kombinirani TV/Audio izlaz (7), poklopac za bateriju (8) te utor za SD karticu (10). U pakiranju sa kamerom uz sve potrebne kablove, bateriju, dolazi i vodootporno kučište za kameru.



Slika ‑ Dijelovi kamere GoPro HD Hero

Specifikacije hardware-a kamere:

* Core: [Ambarella](http://www.ambarella.com/) A2S / A2S70-A1-RH (SoC, ARM Based, ARMv5TEJ Architecture) @ 216 MHz
* Image Sensor: [Micron MT9P401](http://download.micron.com/pdf/flyers/MT9P401%20Flyer.pdf) (1/2.5-Inch 5Mp CMOS)
* Audio Codec: [Texas Instruments TLV320AIC3101](http://focus.ti.com/docs/prod/folders/print/tlv320aic3101.html) (Low-Power Stereo Codec with 6 Inputs, 6 Outputs and DSP)
* Video Decoder: [Analog Devices ADV7403](http://www.analog.com/en/analog-to-digital-converters/video-decoders/adv7403/products/product.html)
* NAND Flash: [Hynix HY27US08561A](http://www.hynix.co.kr/datasheet/eng/nand/details/small_11_HY27US08561A.jsp?menu1=01&menu2=05&menu3=01&menuNo=1&m=5&s=1), 32MB, 256 MBit, SLC
* Hero BUS Connector: [JAE DD1P030MA1 I/O Connector 30P CABLE PLUG](http://www.jae-connector.com/en/general_en.cfm?series_code=DD1)
* I2C on Hero BUS connector
* UART (Serial / RS-232) with 115200/8/N/1, 3.3V Voltage Level

## Hardware

Pomoću mikrokontrolera kontroliramo rad kamere. Naredbe se prenose standardnom RS-232 komunikacijom.

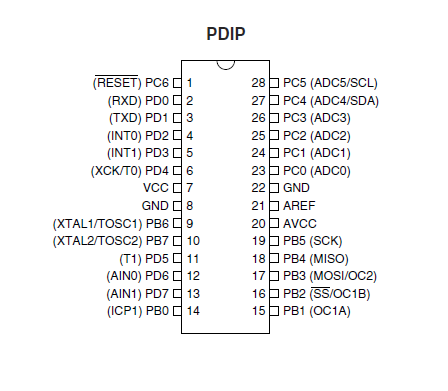
### Odabir mikrokontrolera

Koristiti ćemo Atmel ATmega8 AVR mikrokontroler zbog njegove široke dostupnosti. Atmega8 je 28-pinski mikrokontroler sa dovoljnim brojem ulaznih i izlaznih jedinica, ali ne i presloženom građom.

#### ATmega8

Atmel ATmega8 je 8-bitni AVR mikrokontroler baziran na naprednoj RISC arhitekturi. Ima 8K flash memorije sa čitaj-dok-pišeš mogućnosti, 512 bajtova EEPROM-a, 1 kilobajt SRAMA. Također, ima 23 I/O linije opće namjene, 32 radna registra opće namjene, tri fleksibilna clocka/countera sa usporednim načinom rada, unutarnje i vanjske interrupte , serijski programibilan USART te bajt-orijentirano dvožično serijsko sučelje. Ima 6 kanalni ADC (osam kanala u TQFP i FN/MLF paketu) sa 10 bitnom preciznošću. Također ima programibilni sigurnosni brojač  sa unutarnjim oscilatorom, SPI serijski pristup i pet softverskih izbora za načine štednje energije.

.



Slika ‑ Konfiguracija pinova mikrokontrolera ATmega8

#### Programiranje mikrokontrolera

Mikrokontroler programiramo kako bi mu zadali što će koji pin raditi.

Na protoboard stavimo mikrokontroler ATmega8 i žicama povežemo pinove koje želimo programirati sa programatorom preko podbreakout-a (koristimo najčešće korištene pinove - 1, 3, 4, 5, 6, 8, 13, 12, 21, 23, 25, 27, 30). Programator spajamo preko USB priključka sa kompjutorom.

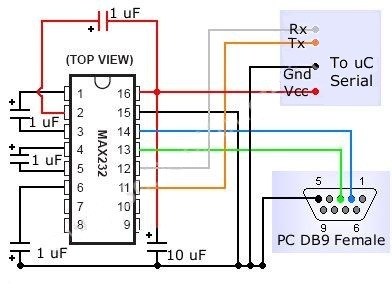
Nakon što smo složili sklop, želimo staviti kod na mikrokontroler. Pokrećemo command prompt i naredbom avrdude pokrećemo AVR. Sada želimo zadati koji ćemo programator (usbtiny) i mikrokontroler (ATmega8) koristiti. Popis programatora izlistavamo naredbom 'avrdude –c asdf', a zadajemo usbtiny nardbom 'avrdude –c avrisp'. Pokrećemo program mikroC PRO i u njemu zadajemo šifru mikrokontrolera. U ovom programu pišemo kod kojim ćemo programirati mikrokontroler. Pokrenemo u terminalu naredbu 'avrdude –c usbtiny –p m328p –u –U flash:w:328Pprogram.hex ' i sad se mikrokontroler, prvo izbriše cijeli, i onda pin po pin nanovo postavlja.

### Serijska komunikacija

Serijska komunikacija se sastoji od dvije komponente. Mikrokontroler čiji su izlazni naponi 0 i 5 V na podatkovnom pinu PB0 i pinu DS275 koji pretvara TTL serijski napon u standardnu RS-232 komunikaciju. Kod TTL komunikacije napon od 0 V označava logičku 0, a od 5 V logičku 1. Kod RS-232 komunikacije logičku 0 predstavljaju naponi u rangu od 3 V do 25 V, a logičku jedinicu naponi u rangu od -3 V do -25 V.

Jedan okvir RS-232 komunikacije (koristeći 8N1: 8 podatkovnih bitova, bez pariteta, 1 zaustavljački bit) izgledat će:

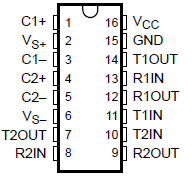
|  |  |  |
| --- | --- | --- |
| 0 | 8 bitova podataka | 1 |



Slika ‑ Standardna serijska komunikacija

#### MAX232

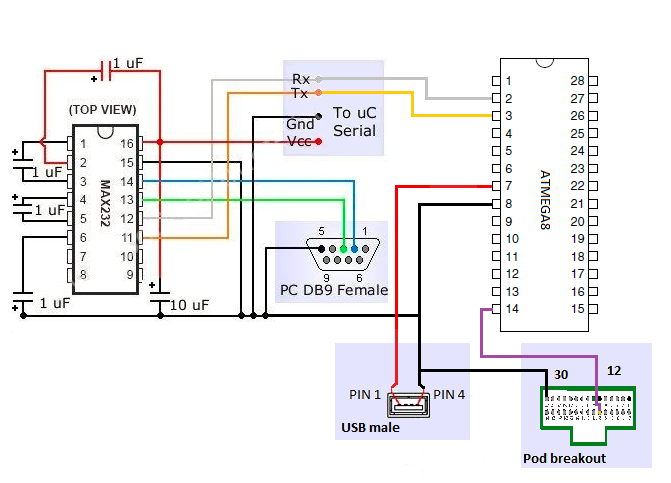
MAX232 je dualni driver/prijamnik koji uključuje kapacitivni generator napona za opskrbu EIA-232 naponskim razinama od jednog 5-V opskrbljivača. Svaki prijemnik pretvara EIA-232 ulaz na 5-V TTL / CMOS razinama. Ovi prijemnici imaju tipičan prag od 1.3 V i tipičnu histerezu od 0.5 V, a mogu prihvatiti ± 30-V ulaza. Svaki driver pretvara TTL / CMOS ulazne razine u EIA-232 razine.



Slika X. Pinovi MAX232 mikrokontrolera

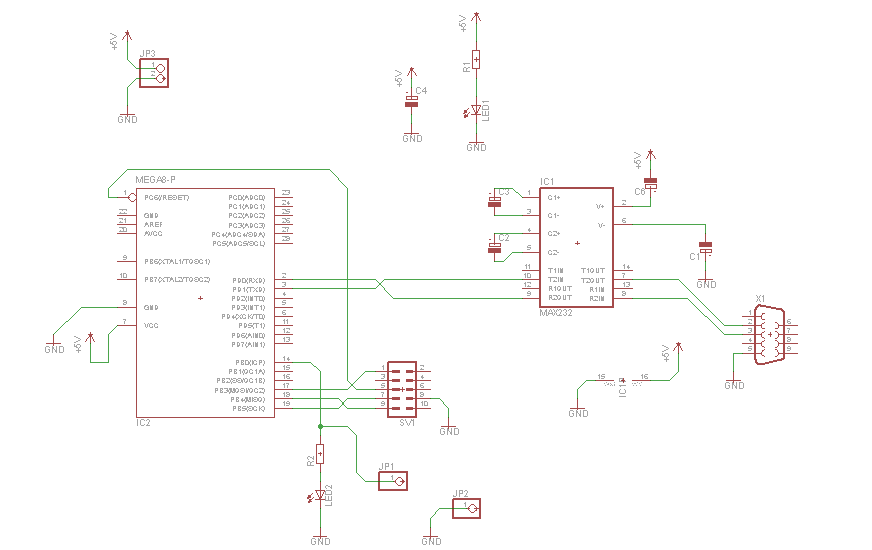
### Elektronička pločica

Na slici 8-5 je prikazana električna shema sklopa koji ćemo slagati. Na shemi je napajanje izvedeno preko USB priključka. Kamera sa spaja preko podbreakout priključka.

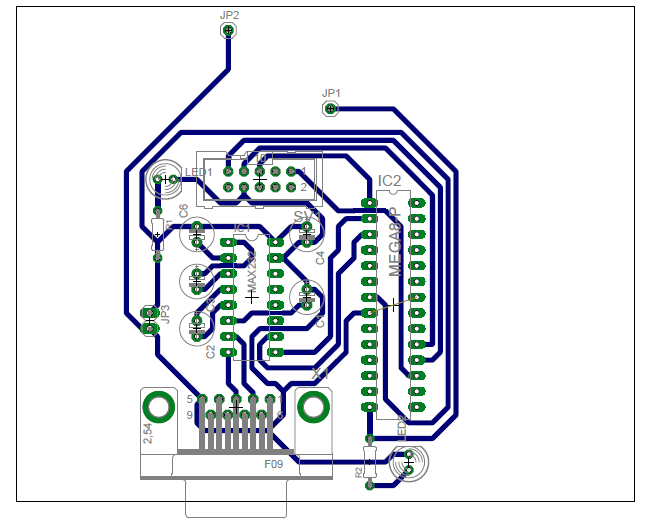


Slika ‑ Električna shema

Na slici 8-6 je prikazana slika električne sheme realizirane u programu Eagle. Ovdje smo dodali dvije diode, jednu koja nam signalizira je li na pločicu priključeno napajanje, te druga koja nam signalizira je li kamera upaljena, tj. ugašena. Za napajanje smo stavili samo dva pina na koja će se kasnije prema potrebi priključiti napajanje preko USB-a ili na bateriju.



Slika ‑ Shema elektroničke pločice u programu Eagle



‑ Shema elektroničke pločice spremne za izradu

## Software

U nastavku je priložen kod driver modula za kameru. Driver modul je razvijen tako da je kompatibilan sa SensorDriver modulom kakav je definiran u LABUST repozitoriju. Unutar drivera su definirane standardne i dodatne naredbe koje će kamera podržavati. Omogućeno je paljenje i gašenje kamere, izbor moda rada, pokretanje i zaustavljanje snimanje.

Driveri se baziraju na slanju riječi mikrokontroleru. Mikrokontroler u skladu sa zahtjevom izvršava svoju funkciju.

# Program za kalibraciju kompasa (Daniel Zima)

## Kompas

Kompas[1] je navigacijski instrument koji mjeri smjer s obzirom na referentni okvir koji je stacionaran s obzirom na površinu zemlje. Referentni okvir definira glavne strane svijeta, sjever, jug, istok i zapad, kao i sporedne strane svijeta, sjeveroistok, sjeverozapad, jugoistok i jugozapad, koje se na kompasu označavaju s pripadajućim slovima. Često se dodaju i oznake u stupnjevima, pri čemu sjever odgovara vrijednosti od 0°, istok 90°, jug 180° i zapad 270° i time se kompasu omogućuje pokazivanje azimuta. Postoji više vrsta kompasa ovisno o načinu izrade ili namjeni.

## Odstupanje kompasa

Magnetometar[2] je mjerni instrument koji se koristi za mjerenje jakosti ili smjera magnetskog polja. Da bi magnetometar bio savršeno kalibriran, mora detektirati samo Zemljino magnetsko polje. Sa idealnim troosnim magnetometrom, činit će se kao da mjerenja potuju duž površine sfere konstantnog radijusa, koju možemo nazvati „mjernim prostorom”. Tek kupljeni kompasi su kalibrirani, ali njihovim korištenjem u prostorima čija magnetska polja dovode do distorzija u tom mjernom prostoru dolazi do značajne degradacije u točnosti kursa.

Dva najčešća odstupanja od mjernog prostora su hard-iron i soft-iron distorzije. Hard-iron distorzija se može opisati kao DC pomak mjerenja koji pomiče izvorište mjerenog prostora. Uzroci tome su permanentni magneti montirani na ili okolo strukture kompasa. Soft-iron distorzija je distorzija ovisna o smjeru pojačanja magnetometara i uzrok joj je prisutnost materijala sa visokom permeabilnošću u okolici strukture kompasa.

## Algoritam za kalibraciju kompasa[3]

Neka je nedistorzirano Zemljino magnetsko polje označeno sa Be, a ono što mjerimo sa Bm. Te dvije matrice su povezane preko sljedećeg izraza:

 (1)

gdje je

S – soft-iron matrica dimenzija 3x3

H – hard-iron matrica dimezija 3x1.

Po koraku se dobiva šest mjerenja, tri iz magnetometra i tri iz akcelerometra i zapisuju se u datoteku za daljnje korištenje pri kalibraciji.

Također je potrebno postaviti dvije bitne pretpostavke:

* jačina Zemljinog magnetskog polja je konstantna, bez obzira na orijentaciju kompasa
* kut između vektora Zemljinog magnetskog polja i vektora ubrzanja Zemljine sile teže je konstantan, bez obzira na orijentaciju kompasa.

Algoritam se temelji na gornje dvije pretpostavke. Prva se koristi za matematičko manipuliranje osima magnetometara na taj način da su ortogonalne i jednakog pojačanja. U ovome koraku algoritma se računa i hard-iron matrica. Pomoću druge pretpostavke određujemo rotacijsku matricu koju koristimo za estimaciju soft-iron matrice i za poravnavanje koordinatnog sustava magnetometra sa koordinatnim sustavom akcelerometra. Zbog toga algoritam ima dva koraka, prvi smješta elipsoidni mjerni prostor u ishodište i pretvara ga u sferu, dok drugi korak rotira dobivenu sferu.

Pojačanje Zemljinog magetskog polja se može izraziti na sljedeći način.

 (2)

Sredni dio se može izraziti u obliku jedne matrice C.

 (3)

Korištenjem izraza (3) u izrazu (2) i daljnim množenjem dobivamo sljedeću kvadatnu jednadžbu.

 (4)

### Prvi korak kalibracijskog postupka

Prvo pretpostavimo da je soft-iron matrica gornja trokutasta.

(5)

(6)

Ako pretpostavimo da se mjereni podaci Bm mogu izraziti kao vektor [x, y, z]T, dobivamo sljedeće.

(7)

Jednadžba se može razdijeliti na tri dijela i riješavanju svakoga će se posebno pristupiti. Prvi dio se može raspisati na jednadžbu s kvadratnim članovima.

(8)

Nakon raspisivanju drugog dijela dobijemo jednadžbu s linearnim članovima,

(9)

pri čemu je:

.(10)

Konstantni članovi su smješteni u treći dio.

(11)

Jednadžba (7) može se prikazati kao umnožak dva matrična vektora: promjenjivog ulaznog vektora i vektora estimiranih parametara.

(12)

Iz gornjeg izraza se pomoću rekurzivne metode najmanjih kvadarat računaju parametri u matrici w. Iz prvih pet parametara dobijemo matricu C iz koje zatim pomoću Cholesky dekompozicije dobijemo gornju trokustasu soft-iron matricu S\_ut. Međutim, Cholesky dekompoziciju ne možemo primjeniti na bilo koju matricu. Da bi se ova metoda mogla primjeniti matrica C mora biti pozitivno definitna.

Hard-iron vektor može se dobiti invertiranjem izraza (10).

(13)

Time smo dobili korekciju prve magnetske distorzije, tj. vratili smo mjerni prostor u ishodište pomoću matrice H i postigli da su osi magnetometara ortogonalne i jednakog pojačanja pomoću matrice S\_ut. Korekcija se odmah primjenjuje na mjernje podatke.

(14)

### Drugi korak kalibracijskog postupka

Pretpostavka da je soft-iron matrica gornja trokutasta je očito loša. Poboljšavamo ju tako da estimiramo rotacijsku matricu koja poravnava koordinatni sustav magnetometra sa koordinatnim sustavom akcelerometra.

(15)

Pretpostavljamo da je koordinatni sustav akcelerometra ortogonalan i da je umnožak vektora akcelerometra i i magnetometra konstanta.

(16)

(17)

Kada pomožimo (16) i (17) dobijemo:

(18)

Nakon toga podijelimo (18) sa r11

(19)

i napravimo isti prikaz kao i u prvom koraku kalibracije.

(20)

Kao i u prvom koraku i sada se primjenjuje rekurzivna metoda najmanjih kvadrata za estimaciju parametara matrice w.

Tako dobivamo matricu T,

(21)

ali nam je vrijednost r11 i dalje nepoznata i zbog toga ćemo primjeniti važno svojstvo rotacijske matrice: determinanta joj je uvijek jednaka 1.

(22)

Samo ako je gornja determinanta pozitivna možemo ažurirati estimaciju vrijednosti r11.

(23)

Iz toga dobivamo rotacijski matricu i uvrštavamo ju u izraz (15) da bi izvršili korekciju mjernih podataka.

(24)

### Rekurzivna metoda najmanjih kvadarata (RLS metoda)

Za svaki korak rekurzije učitava se novi niz podataka: x, y, z dobivenih iz magnetometara i ax, ay i az iz akcelerometara i u svakom koraku kalibracije pokušava se pratiti odabrani (promatrani) signal obs(n). Praćenje se obavlja tako da estimiramo vektor w i pomnožimo ga sa ulaznim vektorom, smanjujući razliku između estimirane i promatrane vrijednosti.

Prije nego što uopće možemo započeti s RLS algoritmom, potrebno je obaviti početne incijalizacije. Za soft-iron i hard-iron matrice uzimamo sljedeće početne vrijednosti.

(25) (26)

P je matrica kovarijance i za oba koraka kalibracije je jedinična matrica pomnožena sa velikim skalarom.

(27)

Početne vrijednosti vektora sa estimiranim parametrima možemo postaviti na sljedeći način:

(28) (29)

Odabrani (promatrani) signali su:

(30) (31)

U svakom koraku čitamo novi set mjerenih podataka i izvršavamo RLS algoritam[5] prema izrazima (32)-(38). Originalni izrazi su razdijeljeni na manje podizraze zbog lakšeg računanja.

(32)

(33)

(34)

(35)

(36)

(37)

(38)

Pri početku rada algoritma postoji šansa da na izlazu dobijemo besmislene vrijednosti vektora estimiranih parametara koje mogu uzrokovati pojavu imaginarnih vrijednosti u soft-iron matrici. Da bi to izbjegli hard-iron i soft-iron matricu ažuriramo samo ako je matrica C pozitivno definitna u prvom koraku i ako je matrica T pozitivno definitna u drugom koraku kalibracije.

# Spajanje CTD mjerne sonde s LinkQuest modemom (Mato Tomić)

## Minos CTD

Minos CTD je jako brz i precizan uređaj visoke rezolucije koji se koristi za mjerenje vodljivosti, temperature i tlaka vode. Može slati podatke u stvarnom vremenu kad ga se spoji kabelom na računalo, ali se može koristiti i autonomno prilikom čega snima prikupljene podatke na internu memoriju uređaja napajanom vlastitom baterijom. Bilo da se uređaj koristi autonomno ili za mjerenja u stvarnom vremenu, prikupljeni podaci se uvijek snimaju na internu memoriju uređaja u dokument kojeg je korisnik prije zadao. Oblik tog dokumenta određuje također korisnik preko ekstenzije istoga (.txt, .raw, .dat i sl.).



Slika ‑ Mjerna sonda Minos CTD

Minos CTD može se koristiti i u slatkoj i u slanoj vodi na dubinama do 500 metara. Može se podesiti tako da vrši mjerenja ciklički po vremenu (svakih nekoliko sekundi, minuta ili sati), ali i tako da snima podatke samo pri promjenama tlaka vode (svakih nekoliko decibara). Serijska komunikacija sa računalom omogućena je preko RS-232, RS-485 i RS-422 priključaka, a interna memorija uređaja iznosi 968.5 MB.

## LinkQuest modem (UMW2000)

LinkQuest je podvodni bežični (akustički) modem koji omogućuje komunikaciju nekog uređaja pod vodom sa drugim, udaljenim uređajem koji se nalazi na površini vode (na plovilu, platformi i sl.). Sastoji se od dva dijela koji međusobno podatke izmjenjuju preko zvučnih signala. Oba dijela mogu primati i slati informacije što znači da je komunikacija ostvariva u oba smjera.



Slika ‑ LinkQuest akustički modem

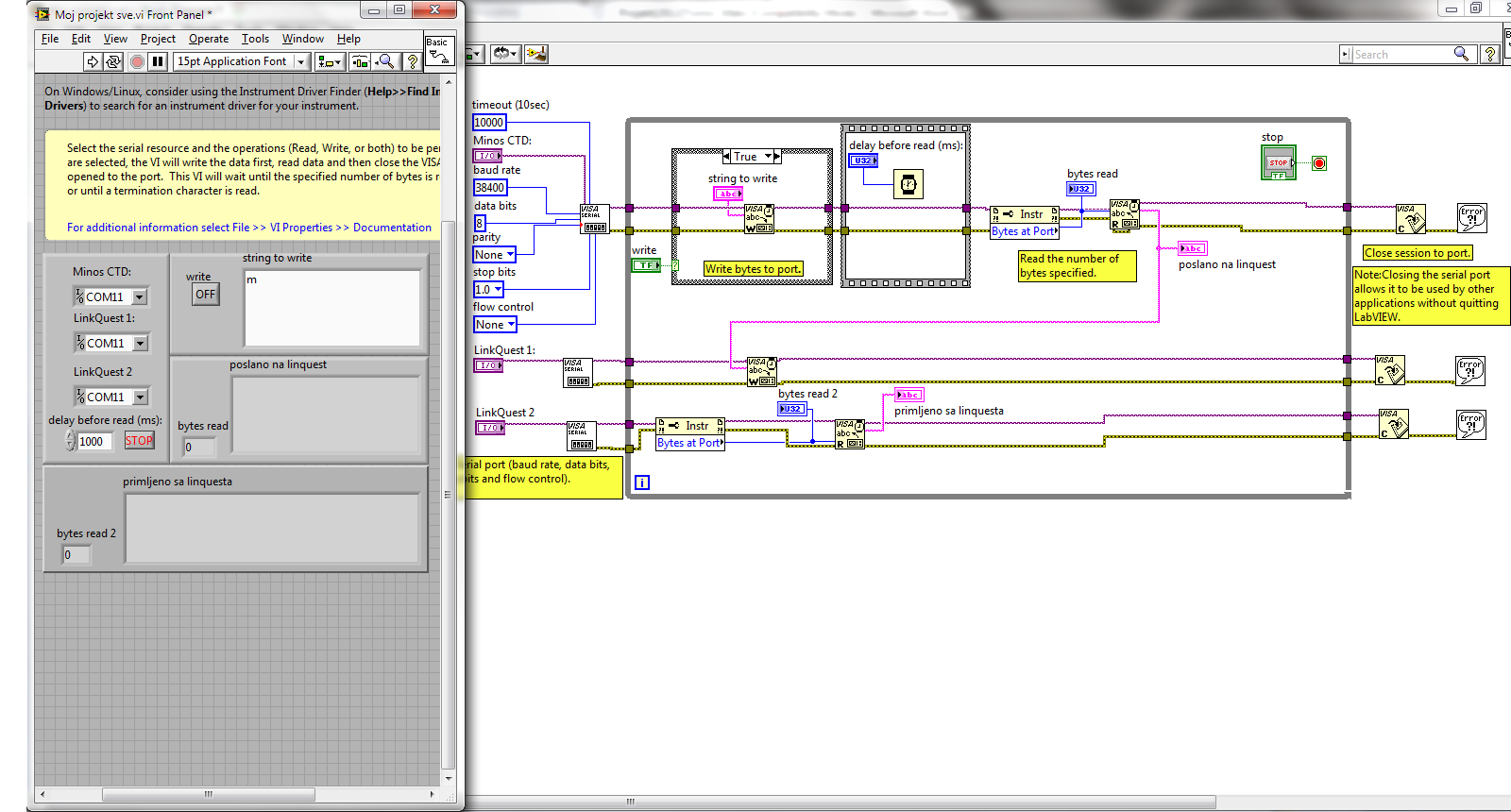
LinkQuest modem može se koristiti u slatkim i u slanim vodama na dubinama do 2000 metara. Brzina prijenosa podataka s jednog kraja modema na drugi pri normalnim uvjetima jest oko 17.8 kbit/s. Serijska komunikacija na oba kraja modema je omogućena preko RS-232 priključka što govori o raznovrsnosti primjene ovog uređaja. Nikakve posebne instrukcije nisu potrebne za odašiljanje i primanje podataka među dijelovima modema što je dodatna pogodnost.

## Komunikacija CTD sonde s udaljenim računalom pomoću LinkQuest modema

Prije bilo kakvog spajanja uređaja u neku zajedničku cjelinu, testiranje se provodilo po komponentama.

Testiranje se započelo sa mjernom sondom Minos CTD. Pregledana je funkcionalnost uređaja i vraćene su tvorničke postavke. Nakon što su proučene ključne riječi i instrukcije kojima se upravlja uređajem, iste su testirane u programskoj podršci koja se dobije uz samu sondu, *SeaCast*. Utvrđena je ispravnost uređaja i testiranje se nastavilo na modemu LinkQuest. I u ovom slučaju bilo je potrebno vratiti tvorničke postavke, a zatim se testirala ispravnost komunikacije među dva dijela modema. Utvrđeno je ispravnost uređaja i preostalo je samo još testirati oba uređaja u jednoj cjelini.

Testiranje komunikacije se obavljalo na jednom računalu u programskom paketu *LabVIEW.* U svrhe jednosmjerne komunikacije nastala je shema prikazana na slici 10-3.



Slika ‑ Shema testiranja komunikacije u jednom smjeru

Ideja sheme je vrlo jednostavna, pokrenuti snimanje karakteristike vode pomoću sonde te rezultate slati na LinkQuest u stvarnom vremenu. Isto testiranje se provodilo u oba smjera tako da se samo zamjeni kraj koji je spojen na sondu sa drugim krajem. Utvrđeno je kako brzina prijenosa podatak često nije jednaka u oba smjera, ali je funkcionalno ispravna.

Preostalo je samo još testirati istovremeno dvosmjernu komunikaciju preko LinkQuesta tako da bi se naredba za pokretanje sonde slala sa računala putem modema, a nakon pokretanja sonde se njeni mjerni podaci šalju natrag na računalo, opet putem modema. U svrhe ovog testiranja nastala je shema prikazana na slici 10-4.



Slika ‑ Shema komunikacije CTD-a preko LinkQuesta sa računalom

Ova shema nešto je složenija od sheme za jednosmjernu komunikaciju jer LinkQuest u ovom slučaju treba biti spreman i za čitanje i za slanje podataka, a da pritom ne naruši rad sonde. Dodano je nekoliko uvjeta pomoću kojih se zadaju prioriteti čitanja, odnosno pisanja LinkQuest modemu.

Testiranjem se utvrdilo da je komunikacija računala i sonde putem LinkQuest modema uspješno ostvarena.

## Osnovne upute za korištenje

Preporučljivo je da se provjere postavke uređaja prije nego se oni počnu koristiti kako ne bi došlo do neželjenih pojava. Prilikom svakog pokretanja sonde Minos CTD zgodno je prvo poslati naredbu oblika:

SET LOG filename.xyz ,

gdje je „filename“ naziv koji se želi pridijeliti datoteci u koju će se sljedeće mjerenje spremati, a „xyz“ oblik pod kojim će se ta datoteka spremiti (ekstenzija tipa .dat, .txt, .raw i sl.). Ovo je zgodno ukoliko nije potrebno podatke prenositi u stvarnom vremenu te nakon snimanja uređaja (snimanje se vrši u vlastitu memoriju uređaja) tim podacima pristupimo vrlo jednostavno i brzo naredbom:

DUMP filename.xyz

Ukoliko želimo pokrenuti snimanje uređaja dovoljno je poslati slovo „m“ (od engl. monitoring) na CTD sondu. Za prekid snimanja dovoljno je poslati bilo koju naredbu ili slovo, ali se ista tada ne smatra nikakvom instrukcijom. Detaljnije upute o korištenju Minos CTD sonde mogu se potražiti u priručniku koji se dobije uz sondu.

# Zaključak

## ****Upravljanje i identifikacija modela broda CyberShip**** (Viktor Vladić i Ines Šoić)

U sklopu projekta napravljeno je dinamičko pozicioniranje broda pomoću regulatora dobivenih vlastitim oscilacijama. Na temelju rada ovdje izloženog vidljivo je da se pomoću vlastitih oscilacija može identificirati proces te doći do parametera regulatora. Kroz razne testove koje smo napravili pokazalo se da regulatori rade zadovoljavajuće. Potrebno je malo duže vremena, no brod se pozicionira na traženu poziciju i ukoliko mu damo malu smetnju on će je kompenzirati i vratiti se na željenu poziciju.

## Nastavak razvoja platforme PlaDyPos (Berislav Marszalek, Nikola Stilinović, Zoran Triska)

U ovom seminaru obrađena je tehnika upravljanja bazirana na virtualnim ciljevima. Virtualni ciljevi korišteni su prvo za praćenje unaprijed definirane putanje. Pokazalo se da jednostavan izbor kvadratične forme za Lyapunovljevog kandidata osigurava asimptotsko približavanje putanji i ostajanje na njoj. Ova metoda pokazala se pogodnom za sintezu nelinearnih regulatora budu´ci da parametri samog procesa koji se kotrolira nisu od prevelikog utjecaja. Koncept jednostavnog upravljanja proširuje se na koordinaciju dva ili više plovila i pri tome najveća prednost ove metode je rasprezanje pojedinih zadataka slijeđenja putanje i koordinacije plovila. Time je znatno olakšano planiranje misija s većim brojem plovila. Upravljanje putem virtualnih ciljeva intenzivno se razvija zadnjih godina i vjerojatno postoje još neke mogućnosti koje bi se mogle daljnje iskorisititi za upravljanje gibaju-im objektima. Simulacijki eksperimentri prisutni u ovom radu, potvrđuju izloženu teoriju i daju joj određenu težinu potrebnu za samu validaciju koncepta upravljanja.

## Komunikacija sa USBL modulom (Dean Ivošević)

Ovim radom prikazano je funkcionalno preuzimanje podataka iz *USBL* modula, na taj način da se ti podaci spremaju pojedinačno u bazi podataka. Ovakav način preuzimanja podataka omogućava zasebno korištenje svakog parametra koji se dobiva iz *USBL* modula. Osim što se preuzimaju podaci koje računa sam modul, dodatno se podaci obrađuju i računaju u svrhu rješavanja praćenja objekta pod vodom, te navođenje platforme iznad samog objekta.

Sustav koji je napravljen je skiciran i opisan, te je pokazana njegova funkcionalnost. Dio koji je napravljen u ovom radu moguće je lako prepravljati i nadopunjavati za buduće potrebe i funkcionalnosti koje će se trebati izvršavati.

## Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

Rad predstavlja jednostavan pregled razvoja komponente za upravljanje kamerom sa pratećom softwareskom podlogom. Opisane su sve ključne komponente modula što daje potpun pogled sa svih razina. Modul omogućava robusno upravljanje kamerom, moguće je pojedinačno slikanje, slikanje u sekvenci kao i snimanje video zapisa zadane duljine. Ciljana primjena razvijene komponente su okruženja koje je potrebno digitalno bilježiti i analizirati.

## Program za kalibraciju kompasa (Daniel Zima)

Još od davnih vremena kompas je bitan navigacijski instrument, osobito u pomorskom i zračnom prometu. Zbog utjecaja elektromagnetskih smetnji iz neposredne okoline kompasa, dolazi do distorzije u mjerenju, tj. javljaju se pogreške u pokazivanju kompasa. Taj problem rješava se kalibracijom kompasa.

U sklopu ovog projekta izrađen je C++ program koji kroz dva koraka algoritma određuje faktor skaliranja i DC pomak mjerenja magnetskog senzora (hard-iron calibration). Pomoću dobivenih hard-iron i soft-iron matrica korigiraju se distorzije u mjerenju. Mjereni podaci, potrebni za rad algoritma, uzimaju se iz senzora i zapisuju u datoteku iz koje se kasnije pri postupku kalibracije čitaju i koriste u algoritmu pri izračunima.

## Spajanje CTD mjerne sonde s LinkQuest modemom (Mato Tomić)

Zadatak ovog dijela projekta bio je spojiti CTD mjernu sondu preko LinkQuest modema sa računalom. Kako bi se taj cilj ostvario bilo je potrebno testirati ispravnost svih komponenata pojedinačno prije spajanja u jednu veću, zajedničku cjelinu. Uočeno je da brzina prijenosa podataka LinkQuest modema često nije ista u oba smjera. Iako je test uspješno obavljen, često je dolazilo i do pogrješaka kojima je uzrok ostao nepoznat. Jedan od mogućih razloga je činjenica da su testiranja obavljana na kopnu, gdje je medij među odašiljačem i prijamnikom zrak, a u normalnim okolnostima bi to trebala biti voda. Valjalo bi napomenuti i da je shema programa prikazana na slici 4. namijenjena samo za testiranje iz razloga što se isto obavljalo na jednom računalu. Ukoliko se uređaji testiraju u realnim okolnostima potrebno je program razdvojiti u dvije cjeline, uz sitne preinake. Može se reći da je primjena ovih uređaja u ovakvoj kombinaciji vrlo praktična i raznovrsna.

# Literatura

## ****Upravljanje i identifikacija modela broda CyberShip**** (Viktor Vladić i Ines Šoić)

[1] Vukić Z.,Kuljača Lj., S. Tešnjak,  Nonlinear Control Systems, Marcel Dekker, Inc. 2003

[2] Vukić Z.,Kuljača Lj., S. Tešnjak,  Automatsko upravljanje,Kigen, Zagreb, 2005

## Nastavak razvoja platforme PlaDyPos (Berislav Marszalek, Nikola Stilinović, Zoran Triska)

[1] M. Aicardi, G. Casalino, A. Balestrino, i A. Bicchi. Closed Loop Smooth Steering Of Unicycle-like Hehicles. Proceedings of the 33rd conference on Decision and Control, Lake Buena Vista, December, 1994.

[2] M. Bibuli. Advanced guidance techiques and mission Cotrol Architectures Development Towards Marine Robotic Systems Enhacement. Doktorska disertacija, University of Genova, Faculty of Engineering, 2010.

[3] M. Bibuli, G. Bruzzone, M. Caccia, i L. Lapierre. Path-following algorithm and experiments for an unmanned surface vehicle. Journal of Field Robotics, 2009.

[4] M. Bibuli, M. Caccia, i L. Lapierre. Virtual target based coordinated pathfollowing for multi-vehicle systems. Technical report, Centre National de la Recherche Scienti¯que, Laboratoire d’Informatique, de Robotique et de Microlectronique de Montpellier, 2009.

[5] M. Bibuli, O. Parodi, i L. Lapierre. Vehicle-following guidance for unmanned marine vehicles. Technical report, CNR-CNRS bilateral agreement "Coordinated mission control for autonomous marine vehicles, 2010.

[6] M. Breivik. Topics in Guided Motion Control of Marine Vehicles. Doktorska disertacija, Norwegian University of Science and Technology, Faculty of Information Technology, Mathematics and Electrical Engineering, 2010.

[7] M. Breivik i I. T. Fossen. Guidance Laws for Autonomous Underwater Vehicles. Norwegian University of Science and Technology, Centre for Ships and Ocean Structures, 2008.

[8] L. Lionel i D. Soetanto. Nonlinear path-following control of an auv. Science Direct, Elsevier, 2006.

37

[9] A. Micaelli i C. Samson. Trajectory tracking for unicycle-type and two-steeringwheels mobile robots. Technical report, Unite´ de recherche INRIA Sophia- Antipolis, 1993.

[10] N. Miškovi´c. Bespilotne ronilice - identifikacija i upravljanje.

[11] N. Mišković. Use of self-oscillations in guidance and control of marine vessels. Doktorska disertacija, University of Zagreb Faculty of electrical engineering and computing, 2010.

[12] N. Mišković, Ð. Nađ, i Z. Vukić. Guidance of Laboratory Platforms. MIPRO, 2010.

[13] N. Mišković, Ð. Nađ, Z. Vukić, i B. Marszalek. Laboratory Platforms for Dynamic Positioning - Modeling and Identification. OCEANS, 2010.

## Positioning - Modeling and Identification. OCEANS, 2010.Komunikacija sa USBL modulom (Dean Ivošević)

[1] Tritech, MicroNav Datasheet http://www.tritech.co.uk/products/datasheets/micron-nav.pdf

[2] Milat Tonči, Simulator plovila temeljen na MOOS-u, Završni rad br. 78, lipanj 2010

[3] Labustov repozitorij https://labust.fer.hr/svn/labust\_svn/trunk

## Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

[1] http://www.youtube.com/watch?v=YtTpUbnct6s&feature=relmfu  
[2] http://www.youtube.com/watch?feature=endscreen&NR=1&v=zRotGzBhx08  
[3] http://shop.kineteka.com/products/118-ipod-iphone-ipad-podbreakout-mini.aspx  
[4] http://www.coolcomponents.co.uk/catalog/product\_info.php?products\_id=48  
[5] http://www.pyroelectro.com/tutorials/simple\_serial\_rs232\_interface/parts.html  
[6] http://www.mikekohn.net/micro/bitbanger\_uart.php  
[7] http://chdk.setepontos.com/index.php?topic=5890.0  
[8] http://gopro.com/products/?gclid=CKTo94ue060CFUaDDgodcUk9mg  
[9] http://gopro.com/

## Program za kalibraciju kompasa (Daniel Zima)

[1] Encyclopædia Britannica Online, "compass", <http://www.britannica.com/EBchecked/topic/129690/compass>, 6. siječanj 2012.

[2] Encyclopædia Britannica Online, "magnetometer", 6. siječanj 2012. <http://www.britannica.com/EBchecked/topic/357442/magnetometer>.

[3] Sodhi, R: “Automatic Calibration of a Three-Axis Magnetic Compass“, 1. lipanj 2005.

[4] Haykin, S: “Adaptive Filtering Theory: Chapter 9“, Prentice Hall, New Jersey, 2002.

## Spajanje CTD mjerne sonde s LinkQuest modemom (Mato Tomić)

[1] Applied Microsystems, Minos CTD User Manual, 6. kolovoz 2009.

[2] LinkQuest Inc., Underwater Acoustic Modem User's Guide, 2004.

[3] LinkQuest Inc., SoundLink Acoustic Modems In Action, www.link-quest.com, 2010.

# Sažetak

## ****Upravljanje i identifikacija modela broda CyberShip**** (Viktor Vladić i Ines Šoić)

U ovom radu napravljena ja identifikacija broda te izračun parametara PID regulatora pomoću metode vlasitith ocilacija. Napravljeni su regulatora po zaošijanju, napredovanju i zanošenju. Takvim upravljanjem je omogućeno dinamičko pozicioniranje broda. Kroz razne testove ustavnovljeno je da regulatori rade zadovoljavajuće.

## Nastavak razvoja platforme PlaDyPos (Berislav Marszalek, Nikola Stilinović, Zoran Triska)

Tema ovog dijela dokumentacije je bila praćenje putanje korištenjem virtualanog cilja. U radu je postupno razrađen postupak upravljanja preko virtualnog cilja. Postavljanje problema je prvo limitirano na jednostavnu konvergenciju u ishodište koordinatnog sustavav. Ovdje je pokazano da forma kvadratne Lyapunovljeve funkcije za kontrolnu funkciju asimptotski vodi sustav u željeno ishodište. Dalje u radu, koncept problema je proširen na gibajući koordinatni sustav, a prethodni slučaj se može promatrati kao stabilizacija u pokretnom koordinatnom sustavu, tzv. *Serret-Frenet* okviru. Rezultati dobiveni nad kvadratnim oblikom Lyapunovljeve funkcije i ovdje su iskorišteni za dobivanje kinematičkih virtualnih algoritama upravljanja. Upravljanje se dalje proširuje na dinamiku i s tim upravljačkim signalima se djeluje na plovilo. U radu je pokazana asimptotska konvergencija plovila na željenu putanju. Izvršeni su simulacijski eksperimenti kako bi potvrdili teorijska razmatranja u cijelom

radu. Koncept upravljanja se proširuje na koordinaciju dva plovila. Opisan je zadatak u kojem jedno plovilo slijedi drugo te zatim koordinacija dva plovila koja trebaju održati paralelnu formaciju gibajući se po predefiniranim putanjama. I ovdje su izvršeni određeni simulacijski eksperimenti. Uspješno je implementiran sbRIO-9642 modul koji svojim mogućnostima odgovara svim zahtjevima upravljanja.

## Komunikacija sa USBL modulom (Dean Ivošević)

Jedan od zadataka koje platforma mora obavljati je praćenje objekta pod vodom, te pozicioniranje nad praćenim objektom. Da bi se ovaj zadatak moga ostvariti platforma će koristiti *USBL* modul. Ovaj modul omogućava lociranje objekta, te nam daje esencijalne podatke za praćenje objekta, te pozicioniranje nad istim.

Preuzimanje podataka iz *USBL* modula ostvareno je pomoću *TCP/IP* veze, a preuzeti podaci zapisuju se u *MOOS* bazu podataka. Cijeli sustav preuzimanja podataka je napisan u programskom jeziku C++, a grafički prikaz preuzetih podataka u programu *Lab View.*

Pošto je preuzimanje podataka s modula i zapisivanje istih u bazu podataka bilo već ostvareno, te se je ono samo implementiralo u ovom radu, glavnina rada je fokusirana na dijelu čitanja iz baze podataka i prikaz tih podataka na ekranu.

## Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

U dokumentaciji projektnog zadatka je analiziran pristup razvoja modula za upravljanje kamerom GoPro. Iznesene su sve faze razvoja ,opisane specifikacije kamere, kao i razvijene komponente sa pratećom softwareskom podlogom za računalo.

## Program za kalibraciju kompasa (Daniel Zima)

Kompas je navigacijski instrument koji mjeri smjer s obzirom na referentni okvir koji je stacionaran s obzirom na površinu zemlje. Tek kupljeni kompasi su kalibriranim, ali njihovim korištenjem u prostorima s jakim magnetskim poljima dolazi do značajne degradacije u točnosti kursa. Dva najčešće posljedice utjecaja magnetskih polja na kompas su hard-iron i soft-iron distorzija. Hard-iron distorzija se može opisati kao DC pomak mjerenja koji pomiče izvorište mjerenog prostora. Soft-iron distorzija je distorzija ovisna o smjeru pojačanja magnetometara.

Cilj ovog projekta bila je izrada C++ program koji će odrediti faktor skaliranja i DC pomak mjerenja magnetskog senzora (hard-iron calibration). Mjereni podaci, koji su prethodno prikupljeni sa kompasa i zapisani u datoteku, koriste se pri izvođenju kalibracijskog algoritma, tj. izračunu korekcijskih parametara (hard-iron i soft-iron matrice).

## Spajanje CTD mjerne sonde s LinkQuest modemom (Mato Tomić)

U svrhe podvodnih istraživanja, ovaj seminarski zadatak opisuje mogućnosti spajanja mjerne sonde Minos CTD sa LinkQuest modemom preko odgovarajuće upravljačke jedinice.

Minos CTD je mjerna sonda koja služi za podvodna istraživanja i njome se mogu mjeriti karakteristike vode kao što su temperatura, tlak, dubina, gustoća, slanoća i sl.

LinkQuest (UMW2000) je podvodni akustički modem koji služi za ostvarivanje komunikacije nekog uređaja koji se nalazi pod vodom sa drugim koji je na površini. Za prijenos informacija koristi se zvučnim signalima.

Zadatak ovog dijela projekta bio je spojiti CTD mjernu sondu preko LinkQuest modema sa računalom. Kako bi se taj cilj ostvario bilo je potrebno testirati ispravnost svih komponenata pojedinačno prije spajanja u jednu veću, zajedničku cjelinu.

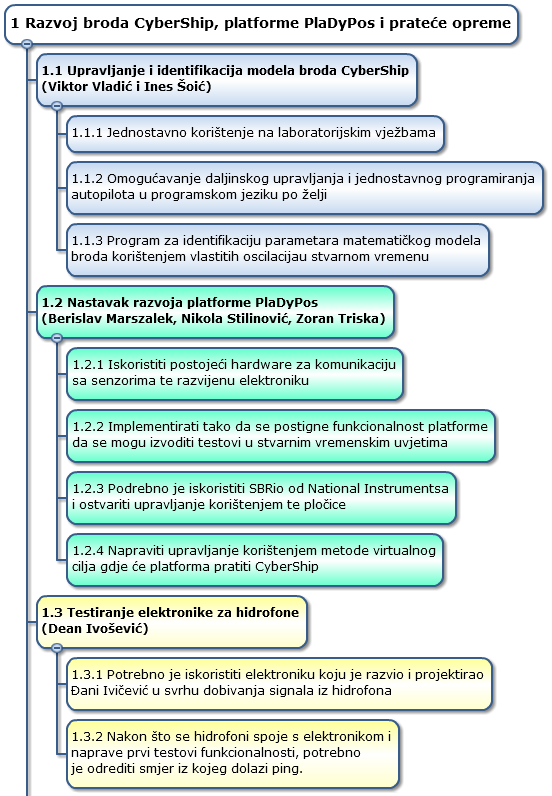
Može se reći da je primjena ovih uređaja u ovakvoj kombinaciji vrlo praktična i raznovrsna.

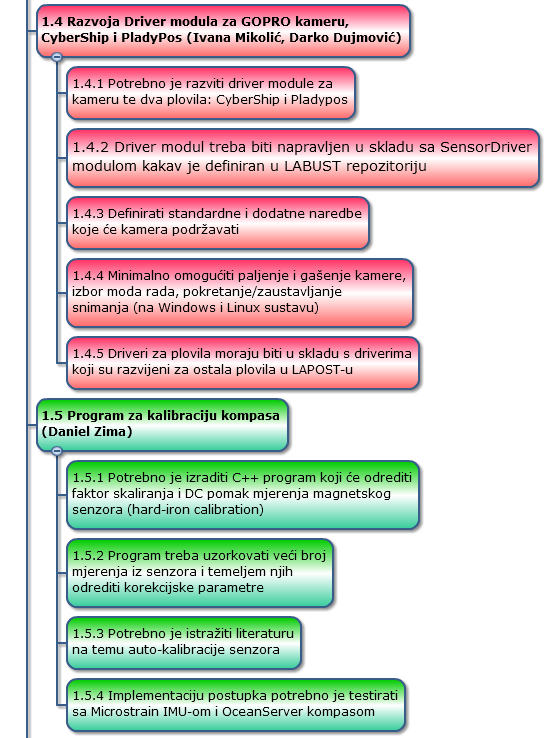
# Resursi

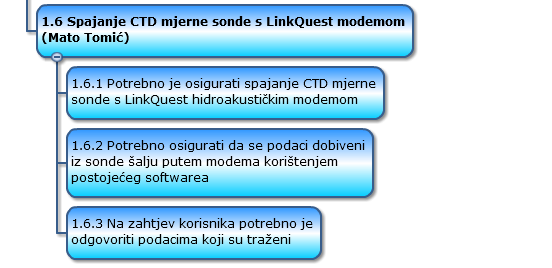
**Tablica ljudskih resursa**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ime i prezime** | **E-mail adresa** | **GSM broj** | **Napomene** |
| Darko Dujmović | darko.dujmovic@fer.hr |  |  |
| Dean Ivošević | dean.ivosevic@fer.hr |  |  |
| Berislav Marszalek | berislav.marszalek@fer.hr |  |  |
| Ivana Mikolić | ivana.mikolic@fer.hr |  |  |
| Nikola Stilinović | nikola.stilinovic@fer.hr |  |  |
| Ines Šoić | ines.soic@fer.hr |  |  |
| Mato Tomić | mato.tomic@fer.hr |  |  |
| Zoran Triska | zoran.triska@fer.hr |  |  |
| Viktor Vladić | viktor.vladic@fer.hr |  |  |
| Daniel Zima | daniel.zima@fer.hr |  |  |

# Struktura raspodijeljenog posla (engl. *Work Breakdown Structure* - WBS)







# Gantogram

# Zapisnici sastanaka

**Zapisnik, 21. listopada 2011.**

Na dan 21. listopada 2011. održan je uvodni sastanak na kojemu su definirane i dodijeljene teme za diplomski projekt. Sljedeći sastanak održat će se u srijedu 26. listopada 2011. u 13 sati u Knjižnici na zavodu na IX. katu C zgrade.

**Zapisnik, 26. listopada 2011.**

Na dan 26. listopada 2011. održan je sastanak na kojemu su dodijeljene teme za diplomski projekt studentima koji nisu mogli prisustvovati uvodnom sastanku. Nazočni su bili:

Darko Dujmović  
Dean Ivošević  
Ines Šoić  
Mato Tomić  
Zoran Triska  
Viktor Vladić  
Daniel Zima  
Nikola Stilinović

Đula Nađ  
Tomislav Lugarić

Na sastanku su studenti iznosili dosadašnja dostignuća na svojim temama i raspravljalo se o pitanjima u vezi pojedinih tema. Sljedeći sastanak održat će se u srijedu 2. studenog 2011. u 13 sati u Knjižnici na zavodu na IX. katu C zgrade.

**Zapisnik, 2. studenog 2011.**

Na dan 2. listopada 2011. održan je sastanak na kojemu se predstavio rezime dostignuća u proteklih tjedan dana,raspravljalo se o pitanjima vezana uz pojedine teme, te se dogovorio daljnji rad na projektu.

Nazočni su bili: Dean Ivošević, Berislav Marszalek, Ivana Mikolić, Nikola Stilinović,  Ines Šoić, Zoran Triska, Nikola Mišković i Đula Nađ

Sljedeći sastanak održati će se u četvrtak 10. listopada 2011. u 16 sati u labosu.

**Zapisnik, dana 10.11.2011.**

Dana 10.11.2011. održan je sastanak u prostorijama Laboratorija za podvodne sustave i tehnologije. Svatko je iznio svoj dosadašnji napredak, rezultate i probleme. Također svatko se pojedinačno dogovorio sa asistentom što treba napraviti do sljedećeg sastanka.

Svi smo pozvani na radionicu "Japanska robotska tehnologija" koja će se održati 17.11.2011. s početkom u 14:00. Nakon završetka radionice dogovoren je sastanak u Laboratorija za podvodne sustave i tehnologije.

Na današnjem sastanku prisustvovali su : Dean Ivošević, Berislav Marszalek, Nikola Stilinović,  Ines Šoić, Mato Tomić, Zoran Triska, Viktor Vladić, Daniel Zima te  Nikola Mišković i Đula Nađ

**Zapisnik, 17.11.2011.**

Većina studenata i asistenata bila je u nazočnosti na današnjem predavanju prof. Hiroshija Ishiguroa i dr. Nobue Yamatoa pod nazivom "Japanska robotska tehnologija". Sastanak nije održan zbog drugih fakultetskih obveza asistenata i studenata.  
Sljedeći sastanak trebao bi se održati 24.11.2011. u 14 sati.

**Zapisnik, 1. 12. 2011.**



Nakon dvotjednog razdoblja međuispita, na dan 1. prosinca 2011. održan je sastanak na kojem su dogovoreni planovi što tko treba ostvariti do idućeg susreta.

Nakon dvotjednog razdoblja međuispita, na dan 1. prosinca. 2011. održan je sastanak na kojem su dogovoreni planovi što tko treba ostvariti do idućeg susreta.  
  
  
**Prisutni:**  
Nikola Stilinović  
Mato Tomić  
Darko Dujmović  
Dean Ivošević  
Daniel Zima  
Zoran Triska  
Nikola Mišković  
Ines Šoić (kasni :))  
  
Odmah u uvodu je zaključeno da nema značajnog pomaka od prošlog sastanka. Najveći razlog tome je ispitno razdoblje koje je za nama te je ovaj sastanak poslužio da se postave kratkoročni ciljevi koje bi trebalo ispuniti za idući put:  
Iznosim kratke crtice o čemu se pričalo!  
  
**Nikola Stilinović**  
- cRIO se ponaša bugovito s instalacijom koja nije na engleskom jeziku  
- elektronika i dalje problem  
- kako uspostaviti komunikaciju između plovila  
  
**Mato Tomić**  
- Sonda kad je spojena s računalom nema problema  
- problem se javlja kad ju treba povezat s Labviewom  
- ispitati u čemu je problem i pokušati otkloniti (zatražit pomoc Đule)  
  
**Darko Dujmović  
-** nastavak rada na kameri  
- postaviti novi firmware na nju  
- kupit AVR chip u Elmatisu ( opet Đula :) )  
  
**Ines Šoić   
-** nedostaju neki blokovi u Edinovom simulatoru kojeg je dobila  
- rješiti problem, vidjeti što od tog Đula ima  
- pokušati zgotoviti blok za identifikaciju putem vlastitih oscilacija  
  
**Daniel Zima**  
- spojit kompas IMU, OS kompas  
- napraviti dio za obradu podataka  
  
**Dean Ivošević  
-** preformuliran zadatak pred kraj sastanka  
  
**Zoran Triska  
-** otkloniti eventualne greške u VT algoritmu  
- povezivati Labview i Matlab s ciljem lakšeg testiranja algoritama realiziranih u Labviewu  
- upogoniti obradu slike koja je razvijena na preddiplomskom projektu  
- identificirati platformu (ak bude u plovnom stanju) zajedno s Ines  
  
**Neki problemi:  
-** elektronika (valjda bude konačno ok)  
**-** Postavlja se pitanje komunikacije između platforme i broda!!  
- postoji samo jedan wireless uređaj!!  
- USBL ?  
- nova folija za bazen stiže za 15 radnih dana (od prošlotjedne uplate)  
  
Na koncu je Deanu konačno dodijeljena tema.  
USBL i izvlačenje podataka o geografskim koordinatama nekog uronjenog objekta.  
Dodatno je moguća njegova projekcija na površinu.  
  
Idući sastanak je ponovo u četvrtak 8. prosinca 2011. u 16:00 sati  u knjižnici zavoda

**Zapisnik 8.12.2011.**

Prošli tjedan u četvrtak 8.12. održan je sastanak u knjižnici na 9. katu C-zgrade. Pričali smo o stvarima vezanim za diplomske projekte. Dosta ljudi je prisustvovalo sastanku i bilo je super. Sljedeći sastanak je ovaj tjedan u četvrtak u 14:00 sati...

Pozdrav

**Zapisnik sastanka, 15.12.2011.**

Sastanak je održan 15.12. u 14h.

Prisutni: Mato Tomić, Ines Šoić, Dean Ivošević, Daniel Zima, Nikola Stilinović, Zoran Triska i Ivana Mikolić.

Bliži se rok za predaju projekta pa polagano svi završavaju radove. Prezentacije će biti nakon Nove Godine.

Dogovoreno je da je sljedeći sastanak u sri, 21.12. u 14h u knjižnici na Zavodu.

Pozdrav.

|  |  |
| --- | --- |
| **Suglasan s dokumentom (potpisuju članovi tima):** |  |
| Darko Dujmović | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Dean Ivošević | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Berislav Marszalek | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Ivana Mikolić | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Nikola Stilinović | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Ines Šoić | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Mato Tomić | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Zoran Triska | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Viktor Vladić | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |
| Daniel Zima | Datum: 17.1.2012. Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |

|  |  |
| --- | --- |
| **Odobrio(potpisuje nastavnik):**  prof. dr. sc. Zoran Vukić |  |

|  |  |
| --- | --- |
| Datum:17.1.2012. | Potpis: \_\_\_\_\_\_\_\_\_\_\_\_ |

# Tehnička dokumentacija

## Razvoj Driver modula za GOPRO kameru, CyberShip i PladyPos (Ivana Mikolić, Darko Dujmović)

### KOD MIKROKONTROLER

#include <stdint.h>

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#define F\_CPU 8000000ul

#define USART\_BAUD 2400ul

#define USART\_UBBR\_VALUE ((F\_CPU/(USART\_BAUD<<4))-1)

#define NAREDBA\_OKIDACA\_FOTO 0x41//A

#define NAREDBA\_OKIDACA\_FILM 0x42//B

#define NAREDBA\_OKIDACA\_ZADANO 0x43//C

#define STATE\_CEKA 0x01

#define STATE\_SPREMAN 0x02

uint8\_t u8Data;

int i;

int odbrojavanje;

int zastavica;

void USART\_vInit(void)

{

PORTB = 0xff;//radimo u negativnoj logici

/\*\* USART \*\*/

UCSRA |= 0x00; // U2X = 0

//UCSRA |= 0x02; // U2X = 1

UCSRB |= 0x98; //

UCSRC |= 0x86; //

UBRRH |= 0x00; // BAUD = 2400

UBRRL |= 0xCF; //

/\*\* PORTOVI \*\*/

// Kod portova DDRx (0 za ulazne, 1 za izlazne)

DDRB = 0x1F;

sei();

}

void USART\_vSendByte(uint8\_t u8Data)

{

// Wait if a byte is being transmitted

while((UCSRA&(1<<UDRE)) == 0);

// Transmit data

UDR = u8Data;

}

uint8\_t USART\_vReceiveByte()

{

// Wait until a byte has been received

while((UCSRA&(1<<RXC)) == 0);

// Return received data

return UDR;

}

int main(void)

{

// Initialise USART

USART\_vInit();

zastavica=0;

for(;;)

{

// Echo received characters

//u8Data = USART\_vReceiveByte();

//USART\_vSendByte(u8Data);

}

}

ISR(USART\_RXC\_vect)

{

if(zastavica){

cli();

zastavica=0;

u8Data = USART\_vReceiveByte();

odbrojavanje= (int)u8Data;

PORTB = 0x00;

\_delay\_ms(2000);//upali me pritiskom 1/4 sec

PORTB = 0xff;

for(i=0;i<odbrojavanje;i++){

\_delay\_ms(480000); //minuta

}

PORTB = 0x00;

\_delay\_ms(24000);//3sec gasi

PORTB = 0xff;

}

else{

cli();

//PORTB = 0x00;

//\_delay\_ms(8000);

PORTB = 0xff;

u8Data = USART\_vReceiveByte();

if(u8Data == NAREDBA\_OKIDACA\_FOTO){

PORTB = 0x00;

\_delay\_ms(2000);//upali me pritiskom 1/4 sec

PORTB = 0xff;

\_delay\_ms(25000); //3sec da uslika

PORTB = 0x00;

\_delay\_ms(24000);//3sec gasi

PORTB = 0xff;

u8Data =NAREDBA\_OKIDACA\_FOTO;

}

else if(u8Data ==NAREDBA\_OKIDACA\_FILM ){

PORTB = 0x00;

\_delay\_ms(2000);

PORTB = 0xff;

\_delay\_ms(100000);//2 min snimaj

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

\_delay\_ms(100000);

PORTB = 0x00;

\_delay\_ms(24000);

PORTB = 0xff;

u8Data =NAREDBA\_OKIDACA\_FILM;

}

else if(u8Data ==NAREDBA\_OKIDACA\_ZADANO ){

zastavica=1;

}

}//od elsa prvog

USART\_vSendByte(u8Data);

sei();

}

### KOD DRIVER

<?xml version="1.0" encoding="UTF-8"?>

<configurations>

<cameraDriverConfig name="a" type="GoProDriver">

<param name="PortName" value="COM1"/>

</cameraDriverConfig>

</configurations>

/\*

\* File: GOPRODriver.h

\* Author: darko dujmovic

\*

\* Created on January 16, 2012, 10:28 AM

\*/

#ifndef \_GOPRODRIVER\_H

#define \_GOPRODRIVER\_H

#include <CameraDriver.h>

#include "ExtendedDataSetCamera.h"

#include "CoreDataSetCamera.h"

#include <boost/asio/serial\_port.hpp>

#include <boost/asio.hpp>

#include <iostream>

namespace LABUST

{

namespace CAMERA

{

/\*

\* This class abstracts a Hero GoPro HD camera

\*/

class GoProDriver : public CAMERA::Driver, private boost::noncopyable

{

public:

/\*\*

\* Constructor from a preloaded XML config file

\* Searches the configuration file for a <cameraDriverConfig> element of type "CameraDriver" and optionally with a specified name

\*

\* \param reader XMLReader with a preloaded configuration file

\* \param configToUse name of configuration element to use, if empty, first one will be used

\* \throws SensorException if configuration is not found in config file

\*/

GoProDriver(const XML::Reader &reader, std::string configToUse = "");

/\*\*

\* Constructor from path to config file

\* Searches the configuration file for a <cameraDriverConfig> element of type "CameraDriver" and optionally with a specified name

\*

\* \param configPath path to configuration file

\* \param configToUse name of configuration element to use, if empty, first one will be used

\* \throws SensorException if configuration is not found in config file

\*/

GoProDriver(const std::string configPath, std::string configToUse = "");

virtual ~GoProDriver();

/\*\*

\* \see LABUST::CAMERA::Driver::setCommand

\*

\*/

int setCommand(CAMERA::COREDATASET::commandMap &command);

/\*\*

\* \see LABUST::CAMERA::Driver::getState

\*

\*/

int getState(CAMERA::COREDATASET::stateMap& state);

/\*\*

\* \see LABUST::CAMERA::Driver::setOptional

\*/

int setOptional(const CAMERA::EXDATASET::dataMap& commands);

/\*\*

\* \see LABUST::CAMERA::Driver::getData

\*/

int getData(CAMERA::EXDATASET::dataMap& dataMap);

private:

boost::asio::io\_service io;

boost::asio::serial\_port serialPort;

std::string portName;

};

}

}

#endif /\* \_DVLDRIVER\_H \*/

/\*

\* File: GOPRODriver.cpp

\* Author: darko dujmovic

\*

\* Created on January 16, 2012, 10:28 AM

\*/

#include <map>

#include <vector>

#include "GoProDriver.h"

#include "Gyros.h"

#include "GyrosScalar.h"

#include <boost/algorithm/string.hpp>

#include <boost/regex.hpp>

#include <XPathQuerier.h>

#include <TimingTools.hpp>

#include <fstream>

#include <SerialConfig.hpp>

#ifdef \_WIN32

// ovisi o prikljucenom kablu

#define DEFAULTPORT "COM21";

#else

// \*nix com ports

#define DEFAULTPORT "dev/ttyS21";

#endif

namespace LABUST

{

namespace CAMERA

{

GoProDriver::GoProDriver(const XML::Reader &reader, std::string configToUse) :

Driver(),

serialPort(io)

{

std::string configQuery;

if (configToUse.empty())

{

configQuery = "cameraDriverConfig[@type='GoProDriver']";

}

else

{

configQuery = "cameraDriverConfig[@type='GoProDriver' and @name='" + configToUse + "']";

}

std::cout << "Configuring GoPro driver " << std::endl;

\_xmlNode\* configNode = NULL;

if (reader.try\_value(configQuery, &configNode))

{

std::string tmp = "";

reader.useNode(configNode);

if(!reader.try\_value("param[@name='PortName']/@value", &portName))

{

portName = DEFAULTPORT;

}

serialPort.open(portName);

if (serialPort.is\_open())

{

using namespace boost::asio;

serialPort.set\_option(serial\_port::baud\_rate(2400));

serialPort.set\_option(serial\_port::flow\_control(serial\_port\_base::flow\_control::none));

serialPort.set\_option(serial\_port::parity(serial\_port\_base::parity::none ));

serialPort.set\_option(serial\_port::stop\_bits(serial\_port\_base::stop\_bits::one ));

serialPort.set\_option(serial\_port::character\_size(8));

}

else

{

throw CameraException("Unable to start GoPro driver, unable to open serial port");

}

std::cout<<"GoPro driver running"<<std::endl;

}

else

{

if(configToUse.empty())

{

throw CameraException("Unable to start GoPro driver, no configurations of type \"GoProDriver\" found.");

}

else

{

throw CameraException("Unable to start DVL driver, no configuration of type \"GoProDriver\" with name \""+configToUse+"\" found.");

}

}

}

GoProDriver::GoProDriver(const std::string configPath, std::string configToUse) :

serialPort(io)

{

LABUST::XML::Reader reader(configPath, true);

std::string configQuery;

if (configToUse.empty())

{

configQuery = "//cameraDriverConfig[@type='GoProDriver']";

}

else

{

configQuery = "//cameraDriverConfig[@type='GoProDriver' and @name='" + configToUse + "']";

}

std::cout << "Configuring GoPro driver: " << std::endl;

reader.useRootNode();

\_xmlNode\* configNode = NULL;

if (reader.try\_value(configQuery, &configNode))

{

std::string tmp = "";

reader.useNode(configNode);

if(!reader.try\_value("param[@name='PortName']/@value", &portName))

{

portName = DEFAULTPORT;

}

serialPort.open(portName);

if (serialPort.is\_open())

{

using namespace boost::asio;

serialPort.set\_option(serial\_port::baud\_rate(2400));

serialPort.set\_option(serial\_port::flow\_control(serial\_port\_base::flow\_control::none));

serialPort.set\_option(serial\_port::parity(serial\_port\_base::parity::none ));

serialPort.set\_option(serial\_port::stop\_bits(serial\_port\_base::stop\_bits::one ));

serialPort.set\_option(serial\_port::character\_size(8));

}

else

{

throw CameraException("Unable to start GoPro driver, unable to open serial port");

}

std::cout<<"GoPro driver running"<<std::endl;

}

else

{

if(configToUse.empty())

{

throw CameraException("Unable to start GoPro driver, no configurations of type \"GoProDriver\" found.");

}

else

{

throw CameraException("Unable to start GoPro driver, no configuration of type \"GoProDriver\" with name \""+configToUse+"\" found.");

}

}

}

GoProDriver::~GoProDriver()

{

}

int GoProDriver::setCommand(CAMERA::COREDATASET::commandMap &commandMap)

{

using namespace boost::asio;

unsigned char command[] = {0x41,0x42,0x43}; // 0x41,0x42 prvi je naredba za slikanje drugo za snimanje sekvence 2 minute, trece sekvenca po varijabli

unsigned char com[2]={};

com[0]=static\_cast<unsigned char>(command[0]);

unsigned char varSeq[1]={} ;

CAMERA::COREDATASET::commandMap::iterator element;

element = commandMap.find(CAMERA::COREDATASET::takePhoto);

if (element != commandMap.end())

{

//take photo :)

com[0]=static\_cast<unsigned char>(command[0]);

write( serialPort, buffer( com, 1 ) );

}

element = commandMap.find(CAMERA::COREDATASET::takeSequence);

if (element != commandMap.end())

{

if(commandMap[CAMERA::COREDATASET::takeSequence]!=0)

{

//custom length

com[0]=static\_cast<unsigned char>(command[2]);

write( serialPort, buffer( com, 1 ) );

boost::this\_thread::sleep(boost::posix\_time::milliseconds(1)); //cekaj trenutak

varSeq[0] =static\_cast<unsigned char>(commandMap[CAMERA::COREDATASET::takeSequence]);

if (varSeq[0]>30)

varSeq[0] =30;

write( serialPort, buffer( varSeq, 1 ) );

}

else

{

//2

com[0]=static\_cast<unsigned char>(command[1]);

write( serialPort, buffer( com, 1 ) );

}

}

return 0;

}

int GoProDriver::getState(CAMERA::COREDATASET::stateMap& state)

{

return 0;

}

int GoProDriver::getData(CAMERA::EXDATASET::dataMap& dataMap)

{

return 0;

}

int GoProDriver::setOptional(const CAMERA::EXDATASET::dataMap& commands)

{

return 0;

}

}

}

## Program za kalibraciju kompasa (Daniel Zima)

### TESTNI.H

#ifndef TESTNI\_H

#define TESTNI\_H

#include <stdio.h>

#include <iostream>

#include <cstdlib>

#include <conio.h>

#include <vector>

#include <math.h>

using namespace std;

void MatrixInversion(float \*\*A, int order, float \*\*Y);

void GetMinor(float \*\*src, float \*\*dest, int row, int col, int order);

double CalcDeterminant( float \*\*mat, int order);

void Transpose(float \*\*A, int xorderA, int yorderA, float \*\*ATrans);

int cholesky(float \*\*orig, int n, float \*\*aug, int mcol,float \*\*chol, float \*\*cholaug, int ofs);

int Multiply(float \*\*A, int xorderA, int yorderA, float \*\*B, int xorderB, int yorderB, float \*\*C);

int Div(float \*\*A, int xorderA, int yorderA, float \*\*C, int xorderC, int yorderC, float \*\*s);

int Sum(float \*\*A, int xorderA, int yorderA, float \*\*B, int xorderB, int yorderB, float \*\*C, int xorderC, int yorderC);

int Sub(float \*\*A, int xorderA, int yorderA, float \*\*B, int xorderB, int yorderB, float \*\*C, int xorderC, int yorderC);

int Copy(float \*\*A, int xorderA, int yorderA, float \*\*C, int xorderC, int yorderC);

void RLS(float \*\*P, float \*\*w, float \*\*u, float \*\*obs, float \*\*lambda, FILE \* pFile, int check);

void RLS2(float \*\*P, float \*\*w, float \*\*u, float \*\*obs, float \*\*lambda, float \*\*Stage1, float \*\*S\_ut, float \*\*H, FILE \* pFile, int check);

int Reading(FILE \* pFile, float \*\*data, int check);

#endif

### ALGORITAM.CPP

#include "testni.h"

float \*\*data;

float \*\*P;

float \*\*lambda;

float \*\*w;

float \*\*u;

float \*\*S;

float \*\*H;

int check;

//zbrajanje matrica

int Sum(float \*\*A, int xorderA, int yorderA, float \*\*B, int xorderB, int yorderB, float \*\*C, int xorderC, int yorderC)

{

int i,j;

if ((xorderA != xorderB)||(yorderA != yorderB)||(xorderA != xorderC)||(yorderA != yorderC))

return 1;

for (i=0; i<xorderA; i++)

{

for (j=0; j<yorderA; j++)

{

C[i][j] = A[i][j]+B[i][j];

}

}

return 0;

}

//oduzimanje matrica

int Sub(float \*\*A, int xorderA, int yorderA, float \*\*B, int xorderB, int yorderB, float \*\*C, int xorderC, int yorderC)

{

int i,j;

if ((xorderA != xorderB)||(yorderA != yorderB)||(xorderA != xorderC)||(yorderA != yorderC))

return 1;

for (i=0; i<xorderA; i++)

{

for (j=0; j<yorderA; j++)

{

C[i][j] = A[i][j]-B[i][j];

}

}

return 0;

}

//mnozenje matrica

int Multiply(float \*\*A, int xorderA, int yorderA, float \*\*B, int xorderB, int yorderB, float \*\*C)

{

int i,j,k, sum;

if (yorderA != xorderB)

return 1;

for (i=0; i<xorderA; i++)

{

for (j=0; j<yorderB; j++)

{

sum = 0;

for (k=0; k<yorderA; k++)

{

sum = sum + A[j][k]\*B[k][j];

}

C[i][j] = sum;

}

}

return 0;

}

//dijeljenje matrice i skalara

int Div(float \*\*A, int xorderA, int yorderA, float \*\*C, int xorderC, int yorderC, float \*\*s)

/\* A je ulazna matrica

C je izlazna matrica

s je sklar s kojim se dijeli matrica A

\*/

{

int i,j;

if ((xorderA != xorderC)||(yorderA != yorderC)) // matrice moraju biti istih dimenzija

return 1;

for (i=0; i<xorderA; i++)

{

for (j=0; j<yorderA; j++)

{

C[i][j] = A[i][j]/s[0][0];

}

}

return 0;

}

//kopiranje matrice

int Copy(float \*\*A, int xorderA, int yorderA, float \*\*C, int xorderC, int yorderC)

{

int i,j;

if ((xorderA != xorderC)||(yorderA != yorderC))

return 1;

for (i=0; i<xorderA; i++)

{

for (j=0; j<yorderA; j++)

{

C[i][j] = A[i][j];

}

}

return 0;

}

// matrix transposition

// the result is put in ATrans

void Transpose(float \*\*A, int xorderA, int yorderA, float \*\*ATrans)

{

if(xorderA==yorderA)

{

for(int i=0;i<xorderA;i++)

for(int j=i;j<yorderA;j++)

{

if(j==i)

ATrans[i][j]=A[i][j];

else

{

ATrans[i][j]=A[j][i];

ATrans[j][i]=A[i][j];

}

}

}

else

{

for(int i=0;i<xorderA;i++)

for(int j=0;j<yorderA;j++)

{

ATrans[j][i]=A[i][j];

}

}

return;

}

int cholesky(float \*\*orig, int n, float \*\*aug, int mcol,float \*\*chol, float \*\*cholaug, int ofs)

/\*

Do the augmented cholesky decomposition as described in FA Graybill

(1976) Theory and Application of the Linear Model. The original matrix

must be symmetric positive definite. The augmentation matrix, or

series of column vectors, are multiplied by C^-t, where C is the

upper triangular cholesky matrix, ie C^t \* C = M and M is the original

matrix. Returns with a value of 0 if M is a non-positive definite

matrix. Returns with a value of 1 with succesful completion.

Arguments:

orig (input) double n x n array. The matrix to take the Cholesky

decomposition of.

n (input) integer. Number of rows and columns in orig.

aug (input) double n x mcol array. The matrix for the augmented

part of the decomposition.

mcol (input) integer. Number of columns in aug.

chol (output) double n x n array. Holds the upper triangular matrix

C on output. The lower triangular portion remains unchanged.

This maybe the same as orig, in which case the upper triangular

portion of orig is overwritten.

cholaug (output) double n x mcol array. Holds the product C^-t \* aug.

May be the same as aug, in which case aug is over written.

ofs (input) integer. The index of the first element in the matrices.

Normally this is 0, but commonly is 1 (but may be any integer).

\*/

{

int i, j, k, l;

int retval = 1;

for (i=ofs; i<n+ofs; i++) {

chol[i][i] = orig[i][i];

for (k=ofs; k<i; k++)

chol[i][i] -= chol[k][i]\*chol[k][i];

if (chol[i][i] <= 0) {

fprintf(stderr,"\nERROR: non-positive definite matrix!\n");

printf("\nproblem from %d %f\n",i,chol[i][i]);

retval = 0;

return retval;

}

chol[i][i] = sqrt(chol[i][i]);

/\*This portion multiplies the extra matrix by C^-t \*/

for (l=ofs; l<mcol+ofs; l++) {

cholaug[i][l] = aug[i][l];

for (k=ofs; k<i; k++) {

cholaug[i][l] -= cholaug[k][l]\*chol[k][i];

}

cholaug[i][l] /= chol[i][i];

}

for (j=i+1; j<n+ofs; j++) {

chol[i][j] = orig[i][j];

for (k=ofs; k<i; k++)

chol[i][j] -= chol[k][i]\*chol[k][j];

chol[i][j] /= chol[i][i];

}

}

return retval;

}

// matrix inversioon

// the result is put in Y

void MatrixInversion(float \*\*A, int order, float \*\*Y)

{

// get the determinant of a

double det=1.0/CalcDeterminant(A,order);

// memory allocation

float \*temp = new float[(order-1)\*(order-1)];

float \*\*minor = new float\*[order-1];

for(int i=0;i<order-1;i++)

minor[i] = temp+(i\*(order-1));

for(int j=0;j<order;j++)

{

for(int i=0;i<order;i++)

{

// get the co-factor (matrix) of A(j,i)

GetMinor(A,minor,j,i,order);

Y[i][j] = det\*CalcDeterminant(minor,order-1);

if( (i+j)%2 == 1)

Y[i][j] = -Y[i][j];

}

}

// release memory

//delete [] minor[0];

delete [] temp;

delete [] minor;

}

// calculate the cofactor of element (row,col)

void GetMinor(float \*\*src, float \*\*dest, int row, int col, int order)

{

// indicate which col and row is being copied to dest

int colCount=0,rowCount=0;

for(int i = 0; i < order; i++ )

{

if( i != row )

{

colCount = 0;

for(int j = 0; j < order; j++ )

{

// when j is not the element

if( j != col )

{

dest[rowCount][colCount] = src[i][j];

colCount++;

}

}

rowCount++;

}

}

return;

}

// Calculate the determinant recursively.

double CalcDeterminant( float \*\*mat, int order)

{

// order must be >= 0

// stop the recursion when matrix is a single element

if( order == 1 )

return mat[0][0];

// the determinant value

float det = 0;

// allocate the cofactor matrix

float \*\*minor;

minor = new float\*[order-1];

for(int i=0;i<order-1;i++)

minor[i] = new float[order-1];

for(int i = 0; i < order; i++ )

{

// get minor of element (0,i)

GetMinor( mat, minor, 0, i , order);

// the recusion is here!

det += (i%2==1?-1.0:1.0) \* mat[0][i] \* CalcDeterminant(minor,order-1);

//det += pow( -1.0, i ) \* mat[0][i] \* CalcDeterminant( minor,order-1 );

}

// release memory

for(int i=0;i<order-1;i++)

delete [] minor[i];

delete [] minor;

return det;

}

// citanje txt datoteke s podacima

int Reading(FILE \* pFile, float \*\*data, int check)

{

int i;

char c;

for(i=0;i<6;i++)

{

fscanf (pFile, "%f", &data[0][i]);

fscanf (pFile, "%c", &c);

if((i==5)&&( c == EOF ))

check=1;

}

return 0;

}

void RLS(float \*\*P, float \*\*w, float \*\*u, float \*\*obs, float \*\*lambda, FILE \* pFile, int check)

{

int i,j;

i=0;

j=1;

//inicijalizacija matrica

//matrica 1x9

float \*\*pi;

pi = new float\*[9];

for(int i=0;i<1;i++)

pi[i] = new float[9];

//matrica 9x1

float \*\*kgain;

kgain = new float\*[1];

for(int i=0;i<9;i++)

kgain[i] = new float[1];

//matrica 9x9

float \*\*pom;

pom = new float\*[9];

for(int i=0;i<9;i++)

pom[i] = new float[9];

//matrica 9x9

float \*\*pom2;

pom2 = new float\*[9];

for(int i=0;i<9;i++)

pom2[i] = new float[9];

//matrica 1x1

float \*\*gama;

gama = new float\*[9];

for(int i=0;i<9;i++)

gama[i] = new float[9];

//matrica 1x1

float \*\*alfa;

alfa = new float\*[9];

for(int i=0;i<9;i++)

alfa[i] = new float[9];

// algorithm start

//pi=uTrans\*P;

Transpose(u, 9, 1, pom);

Multiply(pom, 1, 9, P, 9, 9, pi);

//gama=lambda+pi\*u;

Multiply(pi, 1, 9, u, 9, 1, pom);

Sum(lambda, 1, 1, pom, 1, 1, gama, 1, 1);

//kgain=piTrans/gama;

Transpose(pi, 1, 9, pom);

Div(pom, 9, 1, kgain, 9, 1, gama);

//alfa=d-wTrans\*u;

Transpose(w, 9, 1, pom);

Multiply(pom, 1, 9, u, 9, 1, pom2);

Sub(obs, 1, 1, pom2, 1, 1, alfa, 1, 1);

//wnew=w+kgain\*alfa;

Multiply(kgain, 9, 1, alfa, 1, 1, pom);

Sum(w, 9, 1, pom, 9, 1, pom2, 9, 1);

Copy(pom2, 9, 1, w, 9, 1);

//P'=kgain\*pi

Multiply(kgain, 9, 1, pi, 1, 9, pom);

//Pnew=1/lambda \* (P-P');

Sub(P, 9, 9, pom, 9, 9, pom2, 9, 9);

Div(pom2, 9, 1, P, 9, 1, lambda);

Reading(pFile, data, check);

if(check==0)

{

obs[0][0]=pow(data[0][0],2);

u[0][0]=-2\*data[0][0]\*data[0][1];

u[1][0]=-2\*data[0][0]\*data[0][2];

u[2][0]=-data[0][1]\*data[0][1];

u[3][0]=-2\*data[0][1]\*data[0][2];

u[4][0]=-data[0][2]\*data[0][2];

u[5][0]=2\*data[0][0];

u[6][0]=2\*data[0][1];

u[7][0]=2\*data[0][2];

u[8][0]=1;

RLS(P, w, u, obs, lambda, pFile, check);

}

return;

}

void RLS2(float \*\*P, float \*\*w, float \*\*u, float \*\*obs, float \*\*lambda, float \*\*Stage1, float \*\*S\_ut, float \*\*H, FILE \* pFile, int check)

{

int i,j;

i=0;

j=1;

//inicijalizacija matrica

//matrica 1x9

float \*\*pi;

pi = new float\*[9];

for(int i=0;i<1;i++)

pi[i] = new float[9];

//matrica 9x1

float \*\*kgain;

kgain = new float\*[1];

for(int i=0;i<9;i++)

kgain[i] = new float[1];

//matrica 9x9

float \*\*pom;

pom = new float\*[9];

for(int i=0;i<9;i++)

pom[i] = new float[9];

//matrica 9x9

float \*\*pom2;

pom2 = new float\*[9];

for(int i=0;i<9;i++)

pom2[i] = new float[9];

//matrica 1x1

float \*\*gama;

gama = new float\*[9];

for(int i=0;i<9;i++)

gama[i] = new float[9];

//matrica 1x1

float \*\*alfa;

alfa = new float\*[9];

for(int i=0;i<9;i++)

alfa[i] = new float[9];

float \*\*Bmjer;

Bmjer = new float\*[1];

for(int i=0;i<3;i++)

Bmjer[i] = new float[1];

// algorithm start

//pi=uTrans\*P;

Transpose(u, 9, 1, pom);

Multiply(pom, 1, 9, P, 9, 9, pi);

//gama=lambda+pi\*u;

Multiply(pi, 1, 9, u, 9, 1, pom);

Sum(lambda, 1, 1, pom, 1, 1, gama, 1, 1);

//kgain=piTrans/gama;

Transpose(pi, 1, 9, pom);

Div(pom, 9, 1, kgain, 9, 1, gama);

//alfa=d-wTrans\*u;

Transpose(w, 9, 1, pom);

Multiply(pom, 1, 9, u, 9, 1, pom2);

Sub(obs, 1, 1, pom2, 1, 1, alfa, 1, 1);

//wnew=w+kgain\*alfa;

Multiply(kgain, 9, 1, alfa, 1, 1, pom);

Sum(w, 9, 1, pom, 9, 1, pom2, 9, 1);

Copy(pom2, 9, 1, w, 9, 1);

//P'=kgain\*pi

Multiply(kgain, 9, 1, pi, 1, 9, pom);

//Pnew=1/lambda \* (P-P');

Sub(P, 9, 9, pom, 9, 9, pom2, 9, 9);

Div(pom2, 9, 1, P, 9, 1, lambda);

Reading(pFile, data, check);

if(check==0)

{

obs[0][0]=data[0][3]\*Stage1[0][0];

u[0][0]=-data[0][3]\*Stage1[1][0];

u[1][0]=-data[0][3]\*Stage1[2][0];

u[2][0]=-data[0][4]\*Stage1[0][0];

u[3][0]=-data[0][4]\*Stage1[1][0];

u[4][0]=-data[0][4]\*Stage1[2][0];

u[5][0]=-data[0][5]\*Stage1[0][0];

u[6][0]=-data[0][5]\*Stage1[1][0];

u[7][0]=-data[0][5]\*Stage1[2][0];

u[8][0]=1;

Bmjer[0][1]=data[0][0];

Bmjer[1][0]=data[0][1];

Bmjer[1][1]=data[0][2];

Sub(Bmjer,3,1,H,3,1,pom,3,1);

Multiply(S\_ut,3,3,pom,3,1,Stage1);

RLS2(P, w, u, obs, lambda, Stage1, S\_ut, H, pFile, check);

}

return;

}

void main(float \*\*P, float \*\*lambda, float \*\*w, float \*\*u, float \*\*S, float \*\*H, float \*\*data)

{

int i,j;

float \*\*Stage1;

Stage1 = new float\*[1];

for(int i=0;i<3;i++)

Stage1[i] = new float[1];

float \*\*Stage2;

Stage2 = new float\*[1];

for(int i=0;i<3;i++)

Stage2[i] = new float[1];

float \*\*R;

R = new float\*[3];

for(int i=0;i<3;i++)

R[i] = new float[3];

float \*\*T;

T = new float\*[3];

for(int i=0;i<3;i++)

T[i] = new float[3];

float \*\*C;

C = new float\*[3];

for(int i=0;i<3;i++)

C[i] = new float[3];

float \*\*S\_ut;

S\_ut = new float\*[3];

for(int i=0;i<3;i++)

S\_ut[i] = new float[3];

float \*\*L;

L = new float\*[1];

for(int i=0;i<3;i++)

L[i] = new float[1];

float \*\*obs;

obs = new float\*[1];

for(int i=0;i<1;i++)

obs[i] = new float[1];

float \*\*Bm;

Bm = new float\*[1];

for(int i=0;i<3;i++)

Bm[i] = new float[1];

float \*\*r11;

r11 = new float\*[1];

for(int i=0;i<1;i++)

r11[i] = new float[1];

float \*\*pomoc;

pomoc = new float\*[3];

for(int i=0;i<3;i++)

pomoc[i] = new float[3];

//inicijalizacija globalne

data = new float\*[6];

for(int i=0;i<1;i++)

data[i] = new float[6];

P = new float\*[9];

for(int i=0;i<9;i++)

P[i] = new float[9];

lambda = new float\*[1];

for(int i=0;i<1;i++)

lambda[i] = new float[1];

w = new float\*[1];

for(int i=0;i<9;i++)

w[i] = new float[1];

u = new float\*[1];

for(int i=0;i<9;i++)

u[i] = new float[1];

S = new float\*[3];

for(int i=0;i<3;i++)

S[i] = new float[3];

H = new float\*[1];

for(int i=0;i<3;i++)

H[i] = new float[1];

P = new float\*[9];

for(int i=0;i<9;i++)

P[i] = new float[9];

lambda[0][0]=0.9;

check=0;

FILE \* pFile;

pFile = fopen ("myfile.txt","r");

if (pFile == NULL) perror ("Error opening file");

for (i=0; i<3; i++)

{

for (j=0; j<3; j++)

{

if(i==j)

S[i][j]=1;

else

S[i][j]=0;

}

}

for (i=0; i<3; i++)

{

for (j=0; j<1; j++)

{

H[i][j]=0;

}

}

for (i=0; i<9; i++)

{

for (j=0; j<9; j++)

{

if(i==j)

P[i][j]=100000;

else

P[i][j]=0;

}

}

Reading(pFile, data, check);

Bm[0][1]=data[0][0];

Bm[1][0]=data[0][1];

Bm[1][1]=data[0][2];

for (i=0; i<9; i++)

{

for (j=0; j<1; j++)

{

if(i==8)

w[i][j]=10000;

else if(i==2 || i==4)

w[i][j]=1;

else

w[i][j]=0;

}

}

// Stage 1

obs[0][0]=pow(data[0][0],2);

u[0][0]=-2\*data[0][0]\*data[0][1];

u[1][0]=-2\*data[0][0]\*data[0][2];

u[2][0]=-data[0][1]\*data[0][1];

u[3][0]=-2\*data[0][1]\*data[0][2];

u[4][0]=-data[0][2]\*data[0][2];

u[5][0]=2\*data[0][0];

u[6][0]=2\*data[0][1];

u[7][0]=2\*data[0][2];

u[8][0]=1;

RLS(P,w,u,obs,lambda, pFile, check);

C[0][0]=1;

C[0][1]=w[0][0];

C[1][0]=w[0][0];

C[1][1]=w[2][0];

C[0][2]=w[1][0];

C[2][0]=w[1][0];

C[1][2]=w[3][0];

C[2][1]=w[3][0];

C[2][2]=w[4][0];

L[0][0]=w[5][0];

L[1][0]=w[6][0];

L[2][0]=w[7][0];

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

if(i==j)

pomoc[i][j] = 1;

else

pomoc[i][j] = 0;

}

}

cholesky(C,3,pomoc,3,S\_ut,P,0);

MatrixInversion(C,3,pomoc),

Multiply(pomoc,3,3,L,3,1,H);

Sub(Bm,3,1,H,3,1,pomoc,3,1);

Multiply(S\_ut,3,3,pomoc,3,1,Stage1);

fclose (pFile);

// Stage 2

obs[0][0]=data[0][3]\*Stage1[0][0];

for (i=0; i<9; i++)

{

for (j=0; j<9; j++)

{

if(i==j)

P[i][j]=100000;

else

P[i][j]=0;

}

}

for (i=0; i<9; i++)

{

for (j=0; j<1; j++)

{

if(i==8)

w[i][j]=50;

else if(i==3 || i==7)

w[i][j]=1;

else

w[i][j]=0;

}

}

pFile = fopen ("myfile.txt","r");

if (pFile == NULL) perror ("Error opening file");

u[0][0]=-data[0][3]\*Stage1[1][0];

u[1][0]=-data[0][3]\*Stage1[2][0];

u[2][0]=-data[0][4]\*Stage1[0][0];

u[3][0]=-data[0][4]\*Stage1[1][0];

u[4][0]=-data[0][4]\*Stage1[2][0];

u[5][0]=-data[0][5]\*Stage1[0][0];

u[6][0]=-data[0][5]\*Stage1[1][0];

u[7][0]=-data[0][5]\*Stage1[2][0];

u[8][0]=1;

check=0;

RLS2(P, w, u, obs, lambda, Stage1, S\_ut, H, pFile, check);

T[0][0]=1;

T[0][1]=w[0][0];

T[1][0]=w[2][0];

T[1][1]=w[3][0];

T[0][2]=w[1][0];

T[2][0]=w[5][0];

T[1][2]=w[4][0];

T[2][1]=w[6][0];

T[2][2]=w[7][0];

r11[0][0]=CalcDeterminant(T,3);

if(r11[0][0]>0)

{

pomoc[0][0]=1/r11[0][0];

r11[0][0]=pow(pomoc[0][0],2);

for (i=0; i<3; i++)

{

for (int j=0; j<3; j++)

{

if(i==j)

pomoc[i][j]=r11[0][0];

else

pomoc[i][j]=0;

}

}

Multiply(pomoc,3,3,T,3,3,R);

//Multiply(R,3,3,Stage1,3,1,Stage2);

Multiply(R,3,3,S\_ut,3,3,S);

fclose (pFile);

return;

}

}

## Komunikacija sa USBL modulom (Dean Ivošević)

### MAIN.CPP

#include <stdlib.h>

#include <iomanip>

#include <GyrosMoosCommsInterface.h>

#include <fstream>

#include <boost/algorithm/string.hpp>

#include <JoystickReader.h>

#include <CoreDataSet.h>

#include <ExtendedDataSet.h>

#include <GyrosWriter.hpp>

#include <GyrosReader.hpp>

#include <boost/thread.hpp>

//\*\*\*

#include <iostream>

#include <cstdlib>

#include <math.h>

#define M\_PI 3.14159265358979323846

//\*\*\*

#ifdef WIN32

#define WIN32\_LEAN\_AND\_MEAN

#endif

/\*

\*

\*/

int main(int argc, char\*\* argv)

{

using namespace LABUST;

using namespace COMMUNICATION;

//\*\*\* Ako se želi pisati u datoteke potrebno je sve odkomentirati

// i upisati putanje do željenih datoteka \*\*\*\*

//Definiranje i otvaranje datoteka

/\*

using namespace std;

ofstream BearingData;

double degreesBearing = 1;

BearingData.open(" \*\*\* PATH FOR TXT FILE \*\*\* ");

BearingData << "";

BearingData.close();

ofstream DistanceData;

double realDistance = 0;

DistanceData.open(" \*\*\* PATH FOR TXT FILE \*\*\* ");

DistanceData << "";

DistanceData.close();

ofstream projectedDistance;

double projectedDistance\_M = 0;

double sourceDepth = 0;

projectedDistance.open(" \*\*\* PATH FOR TXT FILE \*\*\* ");

projectedDistance << "";

projectedDistance.close();

\*/

// END Definiranje i otvaranje datoteka

//\*\*\*

std::string path, configToUse;

if(argc < 2)

{

std::cout<<"Please enter path to config file for pUSBLLocator"<<std::endl;

std::cin>>path;

}

else

{

path = argv[1];

}

if(argc < 3)

{

configToUse = "";

}

else

{

configToUse = argv[2];

}

LABUST::XML::Reader reader(path, true);

std::string configQuery;

if (configToUse.empty())

{

configQuery = "/configurations/config[@type='program']";

}

else

{

configQuery = "/configurations/config[@type='program' and @name='"+configToUse+"']";

}

\_xmlNode\* configNode = NULL;

std::string moosConfig = "", GPSLabel = "GPS\_data", USBLLabel = "USBL\_data";

double displayPeriod = 0.3;

if (reader.try\_value(configQuery, &configNode))

{

reader.useNode(configNode);

reader.try\_value("param[@name='MOOSconfig']/@value", &moosConfig);

reader.try\_value("param[@name='GPSLabel']/@value", &GPSLabel);

reader.try\_value("param[@name='USBLLabel']/@value", &USBLLabel);

}

reader.useNode(configNode);

LABUST::COMMUNICATION::GyrosMoosCommsInterface comms(reader, moosConfig);

using namespace VEHICLES;

COREDATASET::stateMap gpsState, usblState;

std::vector<LABUST::XML::GyrosReader> receivedData;

bool running = true;

while (running)

{

comms.Receive(receivedData,true);

if(!receivedData.empty())

{

for(std::vector<LABUST::XML::GyrosReader>::iterator gyros = receivedData.begin(); gyros!=receivedData.end(); gyros++)

{

std::string label = gyros->GetLabel();

try

{

if(boost::iequals(label,GPSLabel))

{

gpsState.clear();

gyros->dictionary(gpsState);

}

else if(boost::iequals(label,USBLLabel))

{

usblState.clear();

gyros->dictionary(usblState);

}

}

catch (...)

{

}

}

receivedData.clear();

}

/\*završeno primanje - USBL state sad ima usbl state u sebi, i gps isto - radi ovdje svoju (crnu) magiju\*/

std::cout<<std::endl<<"USBL: "<<std::endl;

for(LABUST::VEHICLES::COREDATASET::stateMap::iterator stateElement = usblState.begin(); stateElement != usblState.end(); stateElement++)

{

std::cout<<LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first]<<"="<<stateElement->second<<std::endl;

//\*\*\*

// PISANJE U DATOTEKE

/\*

//pisanje Bearing Data

if (LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first] == "BearingFromUSBL")

{

degreesBearing = stateElement->second \* 180 / M\_PI;

if (degreesBearing < 0) degreesBearing = 360 + degreesBearing;

BearingData.open(" \*\*\* PATH FOR TXT FILE \*\*\* ", fstream::in | fstream::out | fstream::app);

BearingData << (degreesBearing) << std::endl;

BearingData.close();

}

//end pisanje Bearing Data

//pisanje SlantRangeFromUSBL Data

if (LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first] == "SlantRangeFromUSBL")

{

DistanceData.open(" \*\*\* PATH FOR TXT FILE \*\*\* ", fstream::in | fstream::out | fstream::app);

realDistance = stateElement->second;

DistanceData << stateElement->second << std::endl;

DistanceData.close();

}

//end pisanje SlantRangeFromUSBL Data

//Racunanje i pisanje projecirane udaljenosti

if (LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first] == "z")

{

sourceDepth = stateElement->second;

projectedDistance\_M = sqrt( realDistance \* realDistance - sourceDepth \* sourceDepth);

projectedDistance.open(" \*\*\* PATH FOR TXT FILE \*\*\* ", fstream::in | fstream::out | fstream::app);

projectedDistance << projectedDistance\_M << std::endl;

projectedDistance.close();

}

// END racunanje i pisanje projecirane idaljenosti

\*/

// END PISANJE U DATOTEKU

//\*\*\*

}

std::cout<<std::endl<<"GPS: "<<std::endl;

for(LABUST::VEHICLES::COREDATASET::stateMap::iterator stateElement = gpsState.begin(); stateElement != gpsState.end(); stateElement++)

{

std::cout<<LABUST::VEHICLES::COREDATASET::stateNames[stateElement->first]<<"="<<stateElement->second<<std::endl;

}

}

return (EXIT\_SUCCESS);

}