SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2907

IMPLEMENTACIJA NAVIGACIJSKIH ALGORITAMA NA ARDUINO PLOČICI

Dario Kurečki

Zagreb, lipanj 2013.

Sadržaj

1. Uvod

Voda je blago bez kojeg svakidašnji život ne bi bio zamisliv. Utjecaj i važnost vode u obliku rijeka, mora i jezera je neupitna u bilo kojoj grani gospodarstva ili energetike. Ljudi svakodnevno traže najbolje načine da iskoriste potencijal vode. Te ideje dovode i do nove tehnologije, te je tako danas robotika pronašla svoj potencijal u podvodnom svijetu.

Podvodna robotika, kao jedna od relativno neistraženih disciplina, ima još puno prostora za razvoj baš zbog puno problema koje sa sobom donosi rad u morskim dubinama. Jedan od najvećih problema podvodne robotike je komunikacija. Elektromagnetski valovi koji se koriste kod GPS-a su današnji standard bežične komunikacije, ali pod vodom imaju puno manji domet nego u zraku. To predstavlja problem jer uređaji na velikim dubinama postanu neupotrebljivi. Jedno od rješenja tog problema je direktno spajanje podvodne ronilice s baznom stanicom na površini preko komunikacijskog kabela. Time nismo riješili problem u potpunosti, jer se time ograničavamo duljinom kabela što na većim dubinama predstavlja problem. Da bismo imali dobar nadzor nad ronilicom dok je pod vodom moramo znati njenu orijentaciju, položaj i dubinu kako ne bismo prešli granicu dubine nakon koje gubimo komunikaciju nad ronilicom. Također dok upravljamo ronilicom pod vodom ne možemo vidjeti njen položaj te nam treba navigacijski inercijski sustav koji će nam računati podatke o rotaciji ronilice.

Jedan takav sustav moguće je realizirati pomoću inercijskog senzora pokreta i GPS-a. Cijeli sustav se ugradi u ronilicu i pomoću inercijskog senzora pokreta možemo jednostavno odrediti orijentaciju ronilice, dok pomoću GPS-a lako odredimo udaljenost i dubinu ronilice od bazne stanice. Takav sustav i navigacijski algoritmi su trenutačno realizirani na PIC mikrokontroleru, a cilj ovog završnog rada je implementacija tih navigacijskih algoritama na Arduino pločici jer su daljnja razvijanja i nadogradnje jednostavnije na Arduino sustavu.

2. Analiza problema

Ronilice se mogu kretati blizu morske površine ili na određenim dubinama. Jedini način komunikacije ronilice i kontrolnog centra na površini je bežična komunikacija ukoliko se radi o ronilici koja vrši svoju misiju blizu površine vode, odnosno akustična komunikacija ako koristimo ronilicu na većim dubinama. U oba slučaja nemamo vizualni kontakt sa ronilicom i zbog tog problema treba u ronilicu integrirati navigacijski sustav koji se može povezati sa baznom stanicom na površini s kojom upravljamo. Navigacijski sustav nam je jedini način da lociramo trenutni položaj ronilice, njenu dubinu i nagib.

Kako bi riješili taj problem bilo je potrebno osmisliti sustav koji će cijelo vrijeme slati navigacijske podatke o položaju ronilice. U sklopu ovog završnog rada projektiran je sustav na Arduino pločici na koju je implementiran postojeći navigacijski algoritam koji će zamijeniti sustav realiziran na PIC-u.

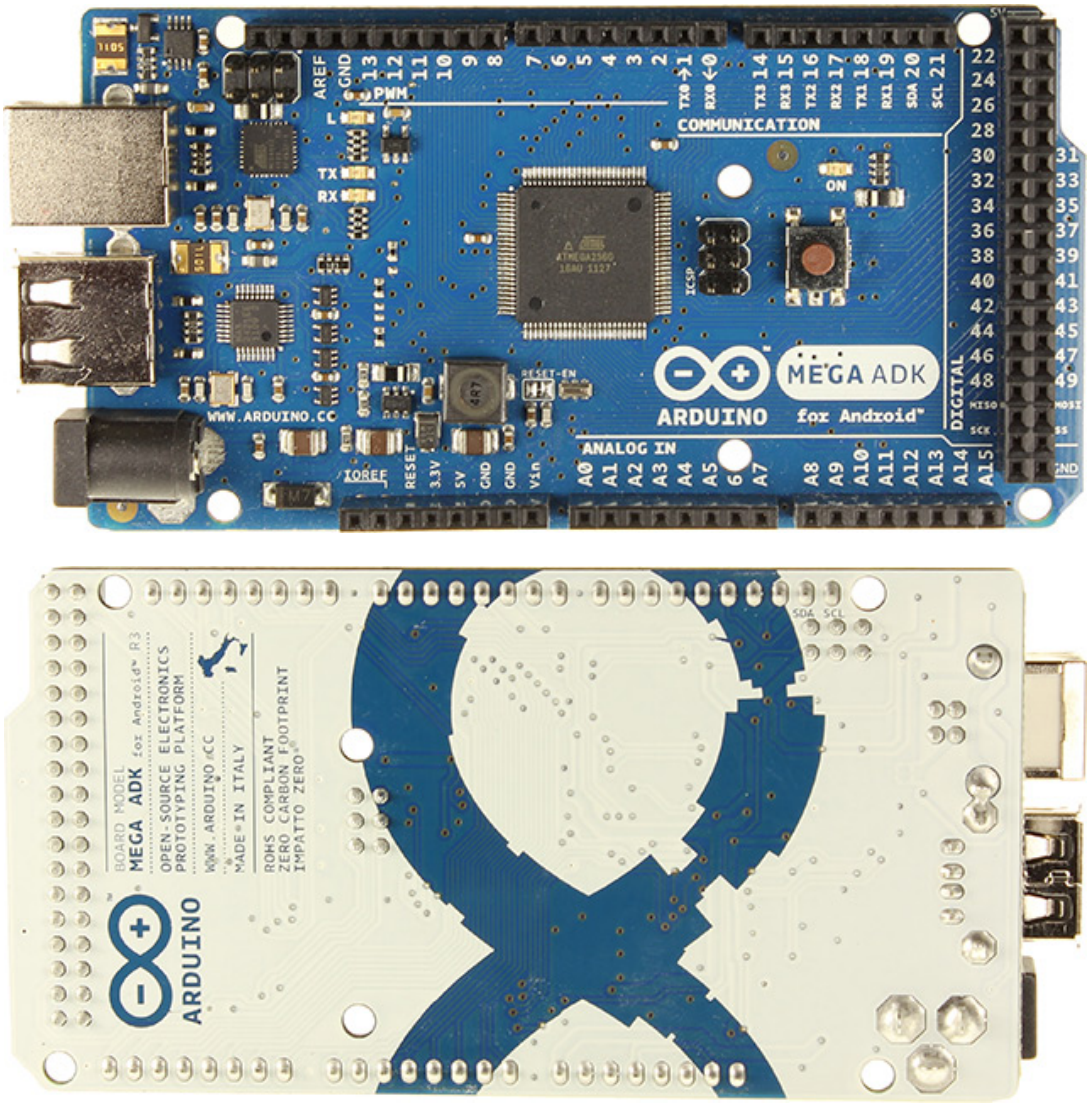
Budući da realizirani sustav koristi inercijski senzor pokreta on loše reagira na promjene magnetskog polja, jer koristi magnetometar za proračun rotacije oko z-osi. Kao primjer promjene magnetskog polja oko ronilice uzmimo slučaj kad se ronilica približi metalnoj konstrukciji broda. Zbog velike količine metalne površine magnetsko polje oko ronilice se mijenja i to utječe na proračun rotacije z-osi. Da bi se izbjegao navedeni problem u navigacijski algoritam integrirana je kalibracija senzora. Sve tri skupine podataka (akceleracija, kutna brzina, magnetsko polje) prolaze kroz Kalmanov filtar kako bi dobili još preciznija očitanja orijentacije ronilice.

2.1. Arduino

Arduino se razvio od sveučilišnog projekta kojemu je cilj bio pojednostaviti studentima izradu projekata potrebnih za učenje. Danas je Arduino platforma otvorenog tipa (eng. *open-source*) načinjena za elektroničke projekte i „oživljavanje“ okoline oko nas. Baziran je na jednostavnom hardware-u i software-u tako da je dostupan svakome uz minimalnu prilagodbu.

S vremenom su se razvile mnoge verzije Arduino pločice, a one se razlikuju po brzini rada, količini memorije i mnogim drugim dodatcima. Prednost Arduina je u tome što ga možemo spojiti na računalo preko USB porta.

U ovom završnom radu korišten je Arduino ADK Mega (Slika 2.1) baziran na ATmega2560 mikrokontroleru.



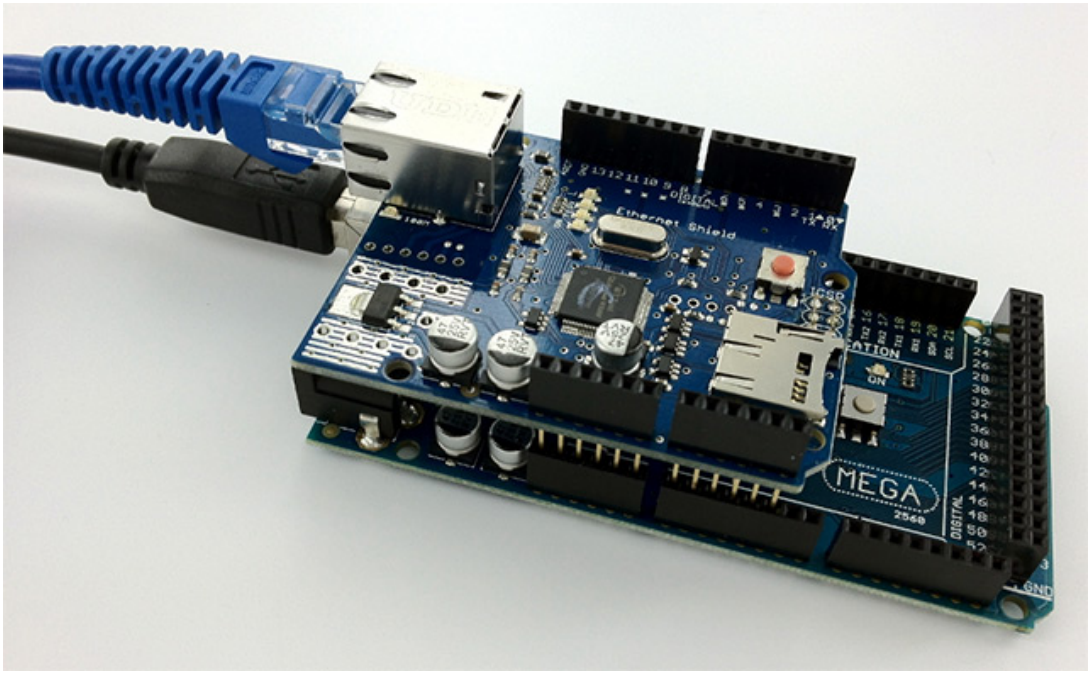
Slika 2.1 Arduino ADK Mega 2560

Arduino kao razvojno okruženje mikrokontrolera mnogo je razvijeniji i popularniji od klasičnih PIC mikrokontrolera. Razlog tome je što Arduino pločica na sebi sadrži sve potrebno za jednostavno programiranje mikrokontrolera koji se nalazi na njoj. Pločica sadrži mnoge pinove preko kojih spajamo ulazne ili izlazne sklopove. Samo programiranje i implementacija programa u Arduino mikrokontroleru je jednostavno realizirana preko USB porta, za razliku od klasičnih PIC mikrokontrolera kod kojih samostalno moramo realizirati cijeli sklop koji će biti potreban za ispravan rad PIC-a.

Programiranje PIC-a se vrši preko programatorskog modula koji se spaja na računalo, a PIC postavimo u modul te izvršimo upis željenog koda. Zatim ga vraćamo

nazad u realizirani sklop. Problematika toga je što za svaku novu programsku izmjenu moramo izvaditi PIC iz sklopa i ubaciti ga u programatorski modul kako bi izvršili novi upis programa.

Osim navedenih prednosti, dodatna prednost Arduino pločice je izuzetno velika dostupnost raznih modula koji se vrlo jednostavno spoje na Arduino te time proširuju mogućnosti sustava. Jedan od primjera je Ethernet modul (Slika 2.2) koji priključimo na Arduino Mega i time proširimo mogućnosti sustava tako da možemo raditi projekte koji imaju potrebu za direktnim spajanjem na Internet.



Slika 2.2 Arduino Ethernet modula

Karakteristike Arduino Mega pločice:

Mikrokontroler	ATMega2560
Radni napon	$5V$
Ulazni napon (preporučeno)	$7-12V$
Ulazni napon (granice)	$6-20V$
Digitalni I/O pinovi	54 (od kojih je 15 PWM izlaz)
Analogni ulazni pinovi	16
DC struja po I/O pinu	$40mA$
DC struja za 3,3 V pin	$50mA$
Flash memorija	$256KB$ od kojih je $8KB$ za bootloader
SRAM	$8KB$

EEPROM	$4KB$
Clock brzina	$16MHz$

Napajanje Arduina se vrši preko USB porta, ali može se vršiti i preko vanjskog DC izvora ili preko AC/DC adaptera. Sustav sam detektira način napajanja i prebacuje se na trenutno aktivni izvor napajanja. Preporučeni izvor napajanja je od 7 do $12V$. Zbog same stabilnosti pinova koji daju izlazni impuls od $5V$ ne smijemo sniziti napon ispod te granice, jer tada sustav ne bi bio stabilan. Ako je Arduino ispravno spojen na izvor napajanja na raspolaganju nam je sustav koji sadrži 54 pina, koje ćemo podijeliti u nekoliko skupina:

- **Napajanje**
- PWM
- Komunikacija
  - Serijska: 0 (RX) i 1 (TX); **Serijska 1: 19 (RX) i 18 (TX); Serijska 2: 17 (RX) i 16 (TX)**; Serijska 3: 15 (RX) i 14 (TX)
  - Vanjski prekidi: 2 (prekid 0), 3 (prekid 1), 18 (prekid 5), 19 (prekid 4), **20 SDA (prekid 3), 21 SCL (prekid 2)**
- Digitalni ulazi
- Analogni ulazi

Sustav koji implementiramo na Arduino pločicu koristi dvije skupine pinova - komunikaciju i napajanje. Arduino Mega u komunikacijskoj skupini pinova podržava serijsku i I^2C komunikaciju.

Programsko okruženje u kojem se Arduino programira je programski jezik C. Postoji službeno programsko okruženje od strane Arduina koje u sebi sadrži prevodioca (eng. *compiler)* i mogućnost direktnog slanja napisanog algoritma na Arduino pločicu putem USB porta.

Budući da se koristi programski jezik C tijekom pisanja programa koristimo sintaksu C jezika. Možemo implementirati sve library funkcije koje postoje u C jeziku u naš program (npr. math.h). Arduino zbog svoje velike primjene i jednostavnosti upotrebe na službenim stranicama nudi gotove library funkcije koje možemo besplatno preuzeti. One nam pomažu da što lakše realiziramo programsko rješenje.

Ukoliko koristimo serijsku komunikaciju ili I^2C komunikaciju potrebno je u naš programski kod implementirati postojeće library datoteke kako što su Wire library i

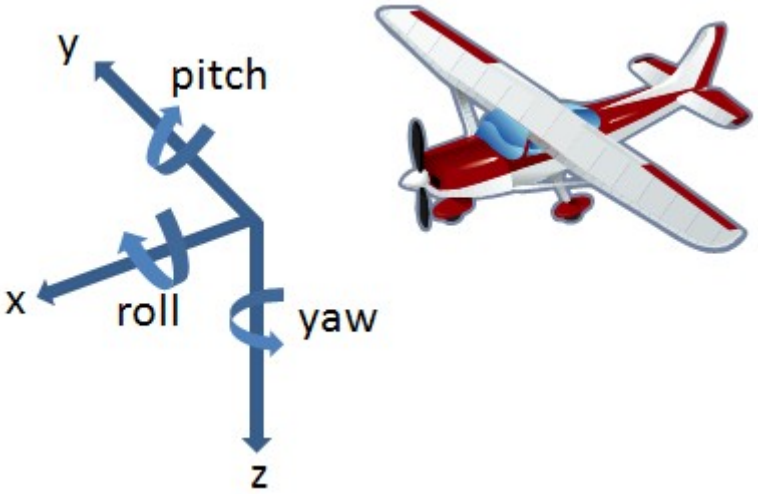
SoftwareSerial library kako bi sa programske strane uspjeli ostvariti vezu preko komunikacijskih pinova.

Na samoj Arduino pločici postoji RESET tipkalo pomoću kojeg možemo cijeli sustav, odnosno programski kod, pokrenuti ispočetka.

Sama kompleksnost Arduino Mega seže mnogo više u dubinu, no ta tematika neće biti obrađena u ovom radu. U opisu su navedene osnovne karakteristike Arduino pločice i elementi koji će biti upotrjebljeni u završnom radu.

2.2. Inercijski senzor pokreta (IMU - Inertial Measurement Unit)

Inercijski senzor pokreta ili IMU je električni sklop koji mjeri i generira podatke bazirane na brzini, orijentaciji i gravitacijskoj sili koristeći se akcelerometrom i žiroskopom, a ponekad i magnetometrom. IMU se najčešće koristi za navigaciju aviona, satelita, zračnih jedrilica, navođenih raketa (Slika 2.3).

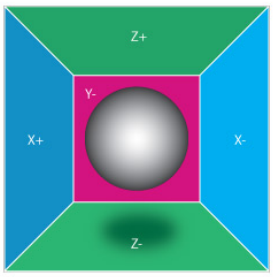


Slika 2.3 IMU kao navigacijski sustav aviona

Osnovni princip rada IMU-a se može podijeliti na tri skupine, a svaka skupina je realizirana preko zasebnog čipa i sadrži sve tri osi:

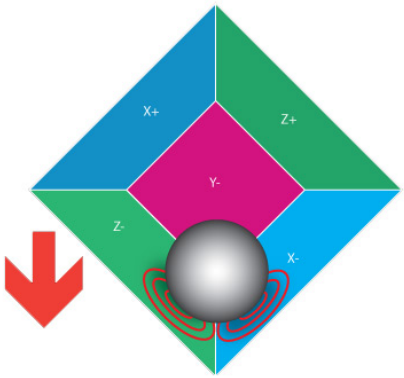
- Akcelerometar
- Žiroskop
- Magnetometar

Princip rada akcelerometra je najlakše prikazati kao kuglicu koja se nalazi u zatvorenoj kutiji (Slika 2.4).



Slika 2.4 Akcelerometar u ravnotežnom položaju

Ukoliko se IMU počne rotirati kuglica će se pomaknuti iz ravnoteže na jednu od stranica kutije zbog sile teže, te će zbog toga sila na toj stranici biti veća. Taj podatak nam otkriva na koju stranu se IMU zarotirao (Slika 2.5).



Slika 2.5 Akcelerometar u rotaciji

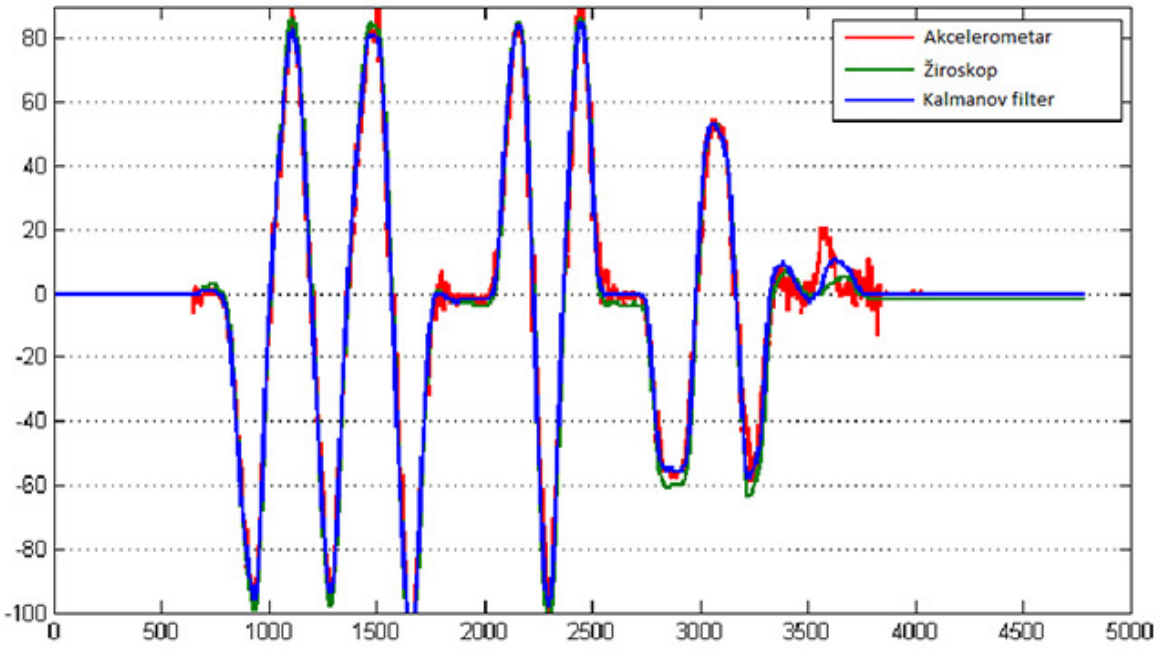
Kada bi se IMU sastojao samo od akcelerometra nastao bi veliki problem. Što bi se dogodilo kad IMU ne bi zarotirali nego samo naglo gurnuli prema naprijed bez ikakvog nagiba? Kuglica u kutiji koju smo dali kao primjer zbog svoje inercije krenula bi prema nazad jer želi ostati u stanju mirovanja. Time bi izašla iz ravnotežnog položaja i promijenila bi gravitacijsku silu na stranicama. Ovaj primjer dovodi do problema: IMU nije rotiran, a podaci pokazuju da je došlo do rotacije. Kako bi se eliminirala ovakvi problemi potrebno je u IMU implementirati još jedan sustav.

Sustav koji pomaže pri orijentaciji IMU-a je žiroskop. Žiroskop mjeri trenutačnu brzinu rotacije. Kada nagnemo IMU žiroskop će zabilježiti trenutačni porast kutne brzine.

Ukoliko bi sada došlo do naglog pomicanja IMU-a prema naprijed akcelerometar bi zabilježio promjene koje rezultiraju da je došlo do rotacije, ali u ovom slučaju žiroskop nije zabilježio nagli porast kutne brzine. Kombinacijom ova dva sustava možemo preciznije zabilježiti kada je IMU doista rotiran, a kada nije.

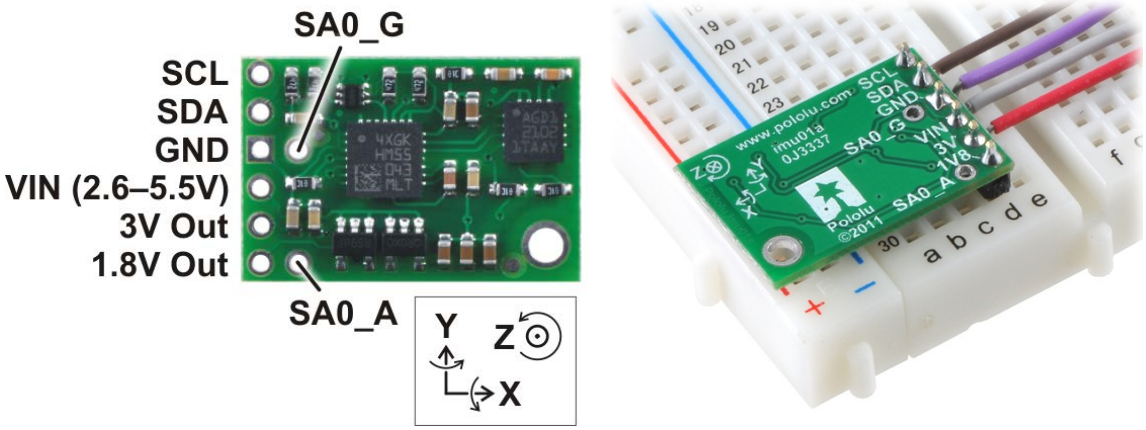
Ponekad se u IMU sustavu može pronaći i magnetometar. Magnetometar nam pomaže za preciznija mjerenja orijentacije oko z-osi. Budući da magnetometar mjeri magnetsko polje Zemlje, sustav će reagirati ako ga rotiramo oko z-osi zbog toga što će više ili manje presijecati magnetsko polje Zemlje.

Dobiveni podatci akceleracije, kutne brzine i magnetskog polja su sami po sebi poprilično „rasipni“. Ako bi koristili podatke dobivene iz IMU-a naš navigacijski sustav ne bi imao veliku preciznost, te bi slobodno mogli reći da je neupotrebljiv. Kako bi dobili veću preciznost i točnost navigacijskog sustava, podatke iz IMU-a moramo propustiti kroz Kalmanov filtar (Slika 2.6).



Slika 2.6 Usporedba mjerenja sa i bez Kalmanovog filtra

Inercijski senzor pokreta koji je korišten u ovom projektu je Pololu MinIMU-9 (Slika 2.7) koji sadrži LSM303DLH i L3G4200D senzore sa I^2C komunikacijom. LSM303DLH senzor je 3-osni akcelerometar i 3-osni magnetometar, a L3G4200D senzor je 3-osni žiroskop.



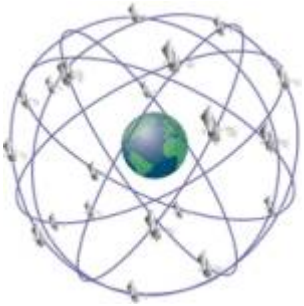
Slika 2.7 Pololu MinIMU-9

Karakteristike Pololu MinIMU-9:

Komunikacija	I^2C
Minimalni radni napon	$2,6V$
Maksimalni radni napon	$5,5V$
Osi	Pitch (x), Roll (y), Yaw (z)
Mjerni opseg	$\pm 250, \pm 500$ ili $\pm 2000°/s$ (žiroskop) $\pm 2, \pm 4$ ili $\pm 8g$ (akcelerometar) $\pm 1.3, \pm 1.9, \pm 2.5, \pm 4.0, \pm 4.7, \pm 5.6$ ili 8.1 gauss (magnetometar)
DC struja	$10mA$

2.3. Globalni pozicijski sustav (GPS - Global Positioning System)

Jeste li se ikada izgubili i poželjeli da postoji lagani način za otkrivanje puta kojim treba ići? Jeste li ikada našli savršeno mjesto za lov ili ribolov, a niste mogli zapamtiti kako do njega ponovo doći? Što napraviti kad pješačite i u jednom trenutku više ne znate kojim se smjerom treba vratiti do kampa ili auta? Sigurno ste se bar jednom morali isključiti iz prometa i pitati nekoga za pravi smjer?



GPS je sustav za određivanje pozicije na zemlji. Omogućuje pouzdano pozicioniranje, navigaciju i vremenske usluge korisnicima širom svijeta na kontinuiranoj osnovi u svim vremenskim uvjetima, danju i noću, svugdje na Zemlji ili blizu

nje, ondje gdje postoji neometan kontakt s četiri ili više satelita GPS-a.

GPS prijemnik izračunava svoju poziciju tako što precizno mjeri vrijeme signala koje šalju sateliti GPS-a visoko iznad Zemlje. Svaki satelit kontinuirano odašilje poruke koje sadrže:

- vrijeme odaslane poruke
- preciznu orbitalnu informaciju (efemeridu)
- stanje općeg sustava i grube orbite svih satelita GPS-a (almanah).

Prijemnik utvrđuje razlike u vremenu potrebnom za primanje poruka. Iz tih vremenskih razlika on utvrđuje razlike u udaljenosti do svakog satelita. Ove se razlike udaljenosti, zajedno sa satelitskim lokacijama, koriste geometrijskom trilateracijom za izračun pozicije prijemnika. Pozicija se zatim prikazuje na zaslonu, a moguć je prikaz i pokretne karte. Informacija o elevaciji također može biti uključena. Mnoge GPS jedinice također prikazuju izvedene informacije poput smjera i brzine koje se računaju iz pozicijskih promjena.

Čini se kako su tri satelita dovoljna za određivanje pozicije, budući da prostor ima tri dimenzije pa se pozicija na Zemljinoj površini može pretpostaviti. No čak i vrlo mala pogreška sata pomnožena s vrlo velikom brzinom svjetlosti, brzinom kojom se satelitski signali šire – rezultira u velikoj pozicijskoj pogrešci. Stoga prijemnici koriste četiri ili više satelita za određivanje svoje lokacije i vremena. Vrlo precizno izračunato vrijeme učinkovito je skriveno u većini primjena GPS-a kod kojih se koristi samo lokacija. Neke specijalizirane primjene GPS-a ipak koriste vrijeme, a među njih spadaju vremenski transfer, sinkronizacija prometnih signala i sinkronizacija baznih stanica za mobilne telefone.

Iako su za normalnu operaciju potrebna četiri satelita, u posebnim slučajevima može ih biti i manje. Ako je jedna varijabla već poznata, prijemnik može odrediti svoju poziciju koristeći samo tri satelita, primjerice brod ili avion mogu znati elevaciju.

U ovom završnom radu želimo realizirati ideju navigacijskog sustava. Kako smo riješili problem prikaza rotacije, postavlja se pitanje kako odrediti točan položaj ronilice i dubinu na kojoj se nalazi. Da bismo uspjeli implementirati takav sustav izabrali smo GPS prijemnik pod nazivom Locosys LS20032 (Slika 2.8). Radna frekvencija je do $10Hz$, a komunikaciju se uspostavlja RS232 serijskom vezom.

Slika 2.8 Locosys LS20032

GPS nudi mogućnost primanja velikog broja različitih vrsta poruka. Jedna od njih je i GPGGA (Global Positioning System Fix Data) koju koristimo u Locosys LS20032. Primjer jedne takve poruke je:

```
$GPGGA,121440.000,4548.0780,N,01558.2940,E,2,7,1.23,192.5,M,42.4,M,0000,0000*53
```

Odnosno u općem obliku:

```
$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh
```

Naziv	**Podaci iz primjera**	**Opis**
Oznaka vrste poruke	`$GPGGA`	Global Positioning System Fix Data
Vrijeme	`121440.000`	12:14:40
Geografska širina	`4548.0780,N`	$45°48'5"N$
Geografska dužina	`01558.2940,E`	$15°58'18"E$
0 = neispravan 1 = GPS fix 2 = DGPS fix	`2`	Poruka se odnosi na GPS fix
Broj satelita	`7`	7 satelita odašilje prema prijemniku
Horizontalna točnost podataka	`1.23`	Relativna preciznost horizontalnih podataka
Nadmorska visina	`192.5,M`	$192.5m$ iznad površine mora
Visina geoida iznad WGS84 elipsoide	`42.4,M`	$42.4m$
Checksum	`*53`	Program koristi ovaj podataka kako bi provjerio da li je primljena poruka ispravna

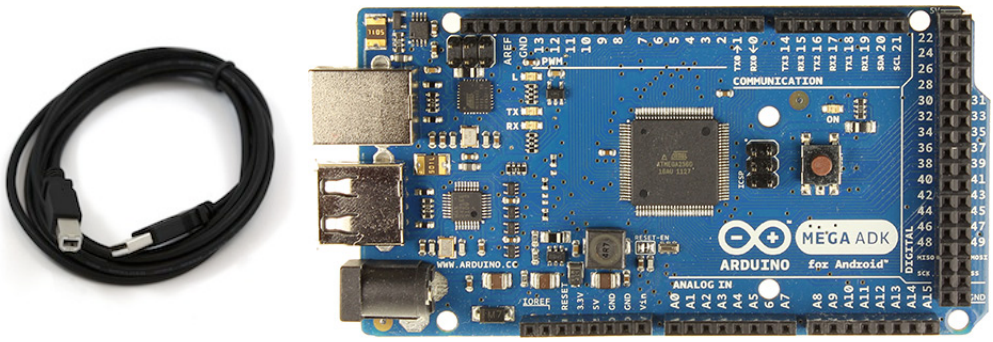
3. Povezivanje cijelog sustava

Povezivanje elektroničkih sklopova možemo podijeliti u nekoliko osnovnih dijelova:

- Povezivanje Arduino pločice sa računalom (serijska komunikacija)
- Spajanje IMU-a na Arduino pločicu
- Spajanje GPS modula na Arduino pločicu

3.1. Povezivanje Arduino pločice sa računalom (serijska komunikacija)

Prvo je potrebno spojiti Arduino pločicu na računalo preko USB kabela (Slika 3.1) preko kojeg ćemo osigurati stabilan izvor napajanja od $5V$ i također omogućiti upis napisanog programa u Arduino.

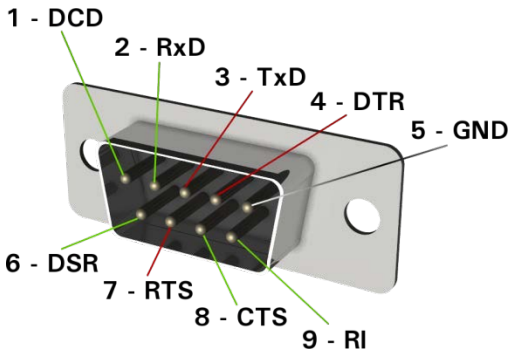


Slika 3.1 Potrebna oprema za spajanje Arduina na računalo

Kada je program upisan u Arduino USB kabel više nije potreban osim za stabilan izvor napajanja od $5V$, jer Arduino ima mogućnost spajanja vanjskog DC izvora napajanja preko $2.5mm$ ulaza, što ga čini neovisnim o računalu.

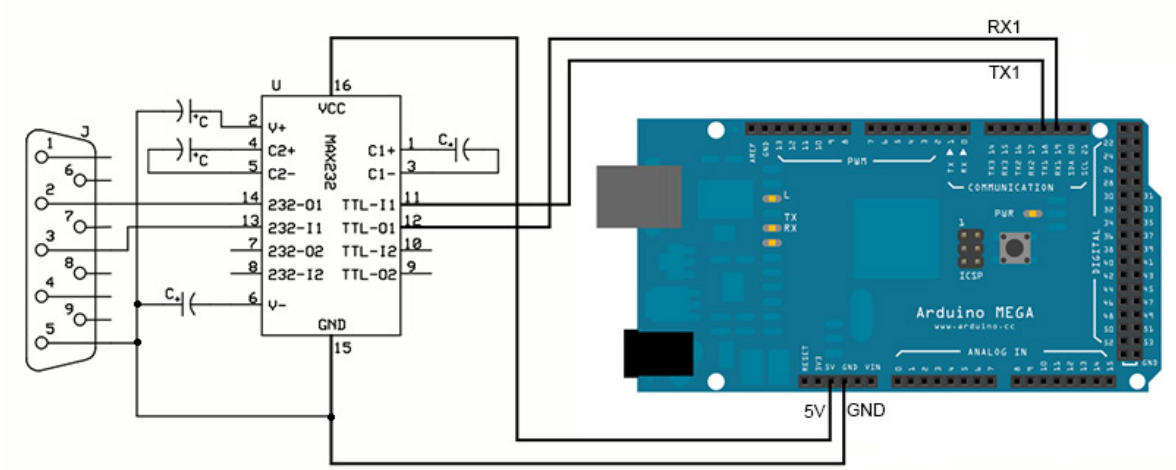
Da bi smo ostvarili serijsku komunikaciju između Arduina i računala potreban nam je USB/RS232 pretvornik (Slika 3.2). USB kraj kabela spojimo na računalo, a RS232 kraj kabela ne smijemo spojiti direktno na Arduino. Razlog tome je naponska razlika Arduina i RS232. Potrebno je znati princip spajanja RS232 na Arduino kao i njegov raspored pinova (Slika 3.3).

Slika 3.2 USB/RS232 pretvornik



Slika 3.3 RS232 raspored pinova

Naponsku razliku između Arduina i RS232 izjednačit ćemo pomoću naponskog pretvornika MAX232. RS232 spajamo na Arduino preko naponskog pretvornika (Slika 3.4).

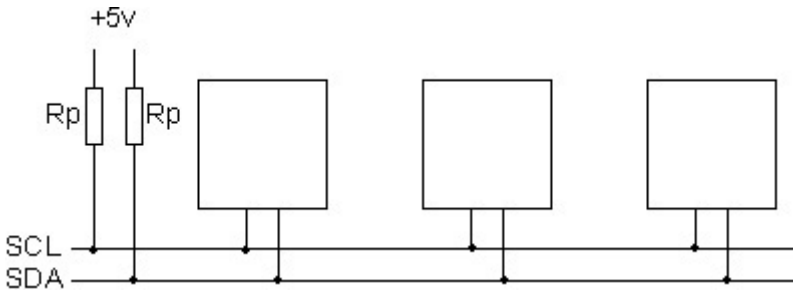


Slika 3.4 Spajanje RS232 na Arduino

Izlaze sa MAX232 (RX i TX) nije poželjno spojiti na ulaze Arduina (RX0 i TX0) zbog toga što su ti ulazi identični USB ulazima. Ako se svejedno odlučimo na spajanje baš tih ulaza onda svaki put kad želimo čitati podatke preko serijske veze moramo odspojiti USB kabel. Bolja rješenje je da ih spojimo na RX1 i TX1 jer ćemo tako moći konstantno koristiti USB komunikaciju za upis programa i serijsku RS232 komunikaciju za čitanje podataka sa Arduino pločice.

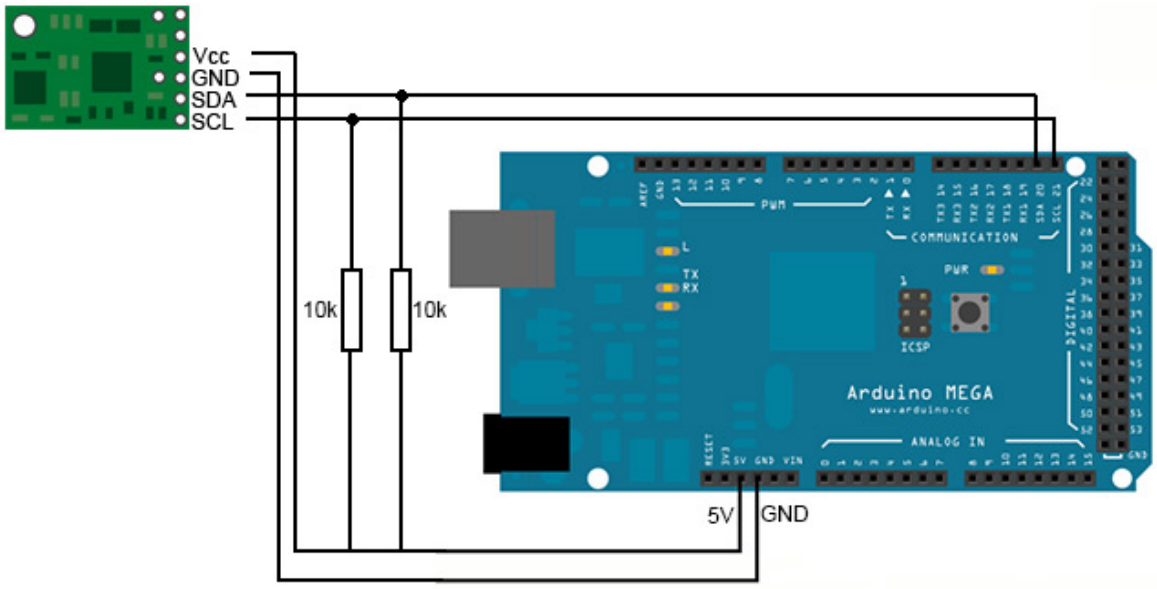
3.2. Spajanje IMU-a na Arduino pločicu

Inercijski senzor pokreta MinIMU-9 spajamo na Arduino preko I^2C sabirnice. I^2C sabirnicu karakterizira činjenica da na nju možemo spojiti više uređaja odjednom, te sa svakim od njih možemo komunicirati neovisno jedno o drugom (Slika 3.5).



Slika 3.5 Više uređaja spojeno na jednu I^2C sabirnicu

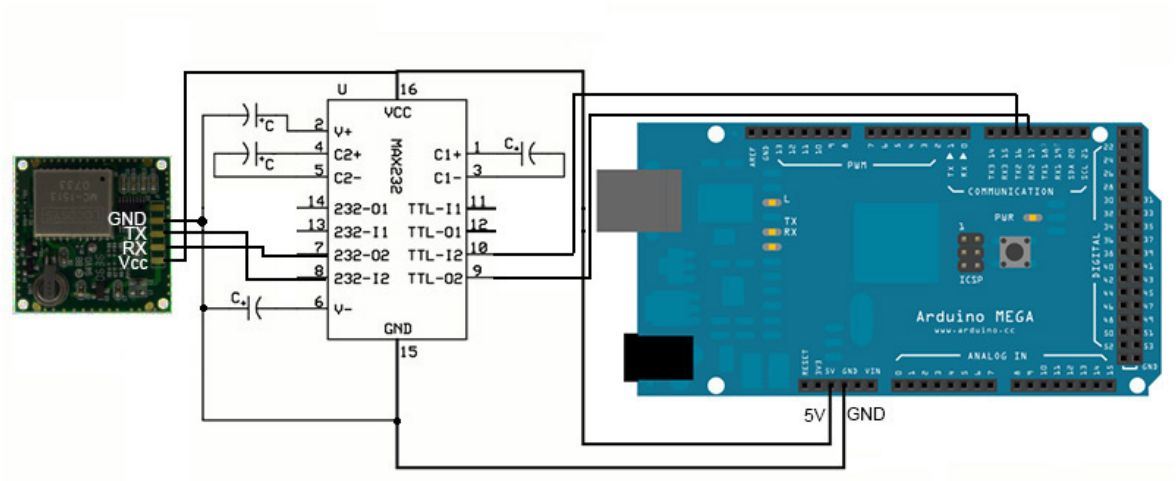
U našem slučaju koristi se samo jedan MinIMU-9 senzor kojeg ćemo spojiti na Arduino I^2C sabirnicu prema Slici 3.6.



Slika 3.6 MinIMU-9 spojen na Arduino preko I^2C sabirnice

3.3. Spajanje GPS modula na Arduino pločicu

GPS modul Locosys LS20032 spajamo preko RS232 serijske sabirnice. Kod ovog GPS modula također je potrebno napraviti naponsku pretvorbu pomoću MAX232. Budući da MAX232 podržava pretvaranje dva sustava, nismo imali potrebu za još jednim MAX232. GPS Locosys LS20032 spajamo prema Slici 3.7



Slika 3.7 GPS modul Locosys LS20032 spojen na Arduino preko RS232 sabirnice

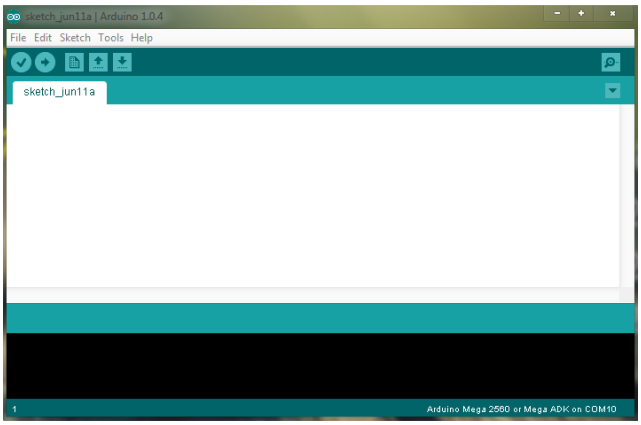
4. Programska implementacija navigacijskog algoritma

Programsku implementaciju navigacijskog algoritma zbog svoje složenosti podijelit ćemo u nekoliko segmenata:

- Programsko okruženje Arduina
- Library potrebni za algoritam
- Inicijalizacija senzora i setup funkcija
- Glavna petlja za prikupljanje i obradu podataka
- Funkcija prekida (engl. Interrupt) (GPS podaci)
- Funkcija za čitanje podataka sa IMU-a
- Funkcija za kalibraciju magnetometra

4.1. Programsko okruženje Arduina

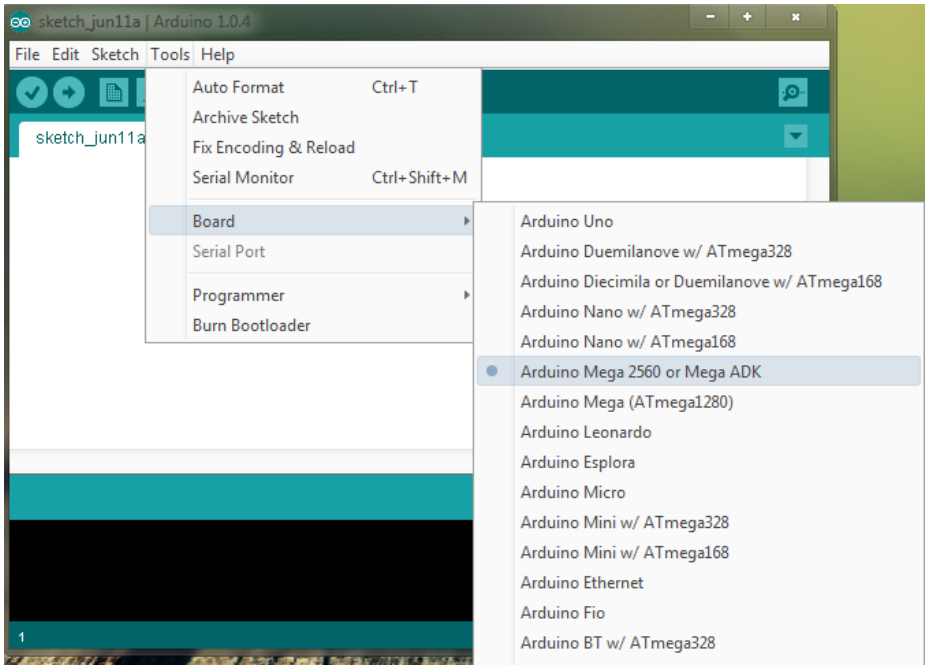
Arduino pločica je uspješno spojena na računalo, kao i senzori preko komunikacijskih portova na Arduino. Prilikom prvog spajanja Arduino pločice na računalo automatski se instaliraju svi potrebni driveri za njegov stabilan rad. Na nama je jedino da preuzmemo službeno programsko okruženje sa službene internet stranice[1]. Nakon završene instalacije programskog paketa, pokrenemo program i prikazat će nam se Arduino programsko okruženje kao na Slici 4.1.



Slika 4.1 Arduino programsko okruženje

[1] http://arduino.cc/en/Guide/Windows

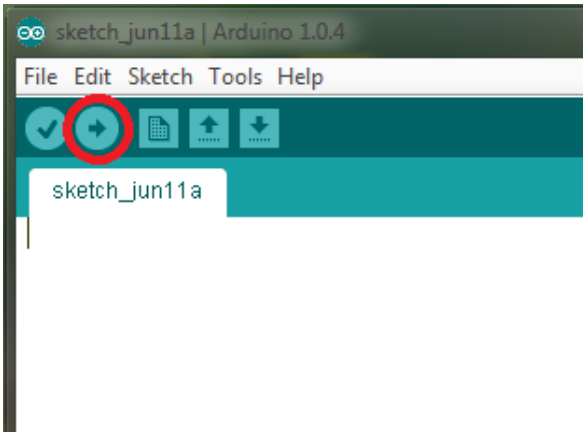
Prije samog pisanja programskog dijela potrebno je podesiti određene parametre unutar samog programskog okruženja. Kao što je navedeno postoje mnoge verzije Arduino pločica, pa je tako jedan od parametara koje treba odabrati model Arduino pločice. U našem slučaju Arduino Mega 2560 or Mega ADK (Slika 4.2).



Slika 4.2 Odabir modela Arduino pločice

Sljedeći parametar je Serial Port odnosno da bi računalo i Arduino programsko okruženje mogli ispravno komunicirati potrebno je podesiti u koji USB port smo spojili našu Arduino pločicu.

Zadnja, ali nikako manje bitna informacija vezana za Arduino programsko okruženje, je kako poslati napisani program u Arduino. Slanje programskog koda ćemo izvršiti na način da pritisnemo tipku Upload (Slika 4.3).



Slika 4.3 Slanje programskog koda na Arduino pločicu

Osnovna struktura svakog Arduino programa:

```
void setup() {

}
void loop() {

}
```

4.2. Library potrebni za algoritam

Prednost Arduino sustava je jednostavnost korištenja koja se jednako odnosi na tehnički i na programski dio. Programsku jednostavnost objašnjava programski jezik C u kojem se pišu svi programi za Arduino. Tu nema razlike između PIC-a i Arduina. Također i PIC mikrokontroleri se mogu programirati u C jeziku. Razlika je u tome što za Arduino postoji velik broj napisanih Library-a. Library su gotove napisane funkcije za određene komponente koje možemo spojiti na Arduino sustav.

Razmotrimo situaciju u našem slučaju: koristimo inercijski senzor pokreta Pololu MinIMU-9. Kako bi ga mogli ispravno koristiti, odnosno po našim potrebama, potrebno je izvršiti inicijalizaciju adresa njegovih registara. U svaki njegov registar treba upisati točno određenu vrijednost kako bi radio baš onako kako želimo. Podatke koje bi nam Pololu MinIMU-9 poslao ne možemo direktno koristiti jer svako očitanje akceleracije za određenu os ima vrijednost poslanu iz dva registra. Takve vrijednosti iz dva različita registra treba spojiti u jedan podatak koji sadrži potpunu informaciju akceleracije za određenu os.

Činjenica je da Pololu MinIMU-9 u ovom sustavu imamo zbog toga što želimo dobiti parametre koji nam govore o rotaciji ronilice. Arduino nam to pruža na jednostavan način: u program implementiramo Library koji je zadužen za svu konfiguraciju registara, spajanje registarskih podataka i proračuna. Na nama je samo da pozovemo funkciju za očitanje željenih parametara iz Pololu MinIMU-9. Prednost ovog sustava je što ne gubimo nepotrebno vrijeme za pisanje cijele konfiguracije Pololu MinIMU-9. Ukoliko netko drugi želi iskoristiti Pololu MinIMU-9 za sustav u kojem bi trebala drugačija konfiguracija sve bi trebalo pisati i konfigurirati ispočetka, dok sa Arduino Library funkcijama cijeli sustav dobiva novu razinu jednostavnosti.

Za ovaj projekt korištene su sljedeće Library funkcije:

- Wire.h
- Math.h
- L3G.h
- LSM303.h
- EEPROM.h
- Timer.h

4.2.1. Wire.h

Wire.h Library nam omogućava da uspostavimo I^2C komunikaciju sa senzorom. Ovaj Library je automatski uključen u Arduino programski paket. Nakon poziva Wire.h možemo koristiti sve njene funkcije. Funkcije koje su korištene u ovom projektu su Wire.begin() i read(). Funkcija Wire.begin() se poziva samo jednom u setup programskom dijelu i postavlja Pololu MinIMU-9 kao glavni uređaj na I^2C sabirnici. Pomoću funkcije read() preuzimamo podatke sa I^2C sabirnice.

Razlika u inicijalizaciji I^2C sabirnice između Arduina i PIC-a:

Arduino

```
#include <Wire.h>

void setup() {
  Wire.begin();
}

void loop() {

}
```

PIC

```
#include <plib/i2c.h>

void i2c_init(void)
{
    OpenI2C1(MASTER, SLEW_OFF);
    IdleI2C1();
    writeByteI2C1(ADRESA_ACC, 0x20, 0b00111111);
    writeByteI2C1(ADRESA_MAG, 0x00, 0b00010000);
    writeByteI2C1(ADRESA_MAG, 0x01, 0b00100000);
    writeByteI2C1(ADRESA_MAG, 0x02, 0b00000000);
    writeByteI2C1(ADRESA_GYR, 0x20, 0b00001111);
    writeByteI2C1(ADRESA_GYR, 0x2E, 0b00000000);
}

void writeByteI2C1(char i2c_add, char i2c_reg, char i2c_dat) {
.
.
.
}

char readByteI2C1(char i2c_add, char i2c_reg) {
.
.
.
}
```

4.2.2. Math.h

U Math.h su sadržane sve matematičke funkcije, kao što su npr. atan2(), sqrt()…

4.2.3. L3G.h i LSM303.h

L3G.h i LSM303.h Library nije automatski uključen u Arduino programsko okruženje, već ga je potrebno preuzeti sa službene stranice[2]. L3G.h Library se odnosi na Pololu MinIMU-9 i u njima su sadržane sve inicijalizacije registara, spajanje podataka iz registara i mnoge druge mogućnosti, čime se samo korištenje Pololu MinIMU-9 u programskom okruženju pojednostavljuje. Ako želimo ispisati podatke sa senzora to ćemo jednostavno učiniti sa pozivom jedne funkcije, a da prethodno nismo gubili vrijeme na pisanje konfiguracije.

4.2.4. EEPROM.h

Ovaj navigacijski sustav također koristi i EEPROM memoriju. Da bismo omogućili zapis podataka u EEPROM memoriju moramo pozvati EEPROM.h Library kako bi dobili pristup funkcijama za upis i ispis podataka iz EEPROM memorije. Funkcija je automatski uključena u Arduino programsko okruženje.

4.2.5. Timer.h

Arduino sučelje sadrži funkciju delay() pomoću koje određujemo koliko želimo da određeni kod bude na čekanju. Pomoću ove funkcije nismo mogli realizirati čitanje podataka sa Pololu MinIMU-9 u uvijek određeno vrijeme, jer funkcija kao rezultat daje zastoj programa na zadano vrijeme. Kao rješenje postoji Timer.h Library pomoću koje možemo pozivati bilo koju funkciju u programu nakon točno predefiniranog vremena. U našem slučaju pozivamo čitanje podataka sa IMU-a nakon svakih $100ms$.

```
t.every(100, imu);
```

[2] https://github.com/pololu/LSM303 i https://github.com/pololu/L3G

4.3. Inicijalizacija senzora i setup funkcija

Potrebne dodatke za Arduino programsko okruženje smo implementirali, ali su zapisi registara možda predefinirani za rad senzora na način koji nama ne odgovara. Da bi smo saznali u koji mod rada su postavljeni naši senzori, potrebno je otvoriti Library datoteke L3G.h i LSM303.h našeg Pololu MinIMU-9. U tim datotekama ćemo pronaći nazive registara i vrijednosti koje su im pridružene.

Promjene koje smo izvršili u L3G.h su:

- CTRL_REG1 (20h) registar žiroskopa postavljena na 0b00001111
  - Normalan način rada, $100Hz$ radne brzine, X, Y i Z osi uključene
- FIFO_CTRL_REG (2Eh) registar žiroskopa postavljena na 0b00000000
  - Zaobilazi FIFO buffer

Promjene koje smo izvršili u LSM303.h su:

- CTRL_REG1_A (20h) registar akcelerometra postavljen na 0b00111111
  - Normalan način rada, $1000Hz$ redne brzine
- CRA_REG_M (00h) registar magnetometra postavljen na 0b00010000
  - Normalan način rada, minimalno $15Hz$ radne brzine
- CRB_REG_M (01h) registar magnetometra postavljen na 0b00100000
  - Maksimum mjerenja ± 1.3 Gauss
- MR_REG_M (02h) registar magnetometra postavljen na 0b00000000
  - Besprekidni konverzijski način rada

Svi ostali registri su ostali predefinirani po originalnim postavkama.

Početak setup funkcije započinje sa kodom:

```
Serial1.begin(115200);  //Uspostavljanje RS232 serijske veze
                          Arduino-računalo.
Serial2.begin(115200);  //Uspostavljanje RS232 serijske veze Arduino-
                          Locosys LS20032.
Wire.begin();           //Uspostavljanje I2C komunikacije Arduino-
                          Pololu MinIMU-9.
pinMode(8, INPUT);      //Definiramo 8. pin na Arduinu kao ulazni pin,
                          pomoću ovog pina dajemo impulsni signal od 5V
                                za aktivaciju kalibracije.
compass.init();         //Vršimo inicijalizaciju žiroskopa,
gyro.init();              magnetometra i akcelerometra. Ovim naredbama
                          pozivamo funkcije iz L3G.h i LSM303.h te
                          vršimo upis definiranih vrijednosti u
                          registre.
```

```
compass.enableDefault();
gyro.enableDefault();

t.every(100, imu);              //funkcija iz skupine Timer.h, pomoću
                                  koje pozivamo funkciju imu svakih 100ms

pttr.reserve(100);              //polje od 100 članova u koje spremamo
                                  GPS poruku
gga.reserve(100);               //pomoćno polje od 100 članova u koje
                                  spremamo kopiju GPS poruke
x_0 = EEPROM.read(1) * 256;     //čitanje kalibriranih podataka
x_0 += EEPROM.read(2);          //magnetometra iz EEPROM memorije
y_0 = EEPROM.read(3) * 256;
y_0 += EEPROM.read(4);
z_0 = EEPROM.read(5) * 256;
z_0 += EEPROM.read(6);
x_max = EEPROM.read(7) * 256;
x_max += EEPROM.read(8);
y_max = EEPROM.read(9) * 256;
y_max += EEPROM.read(10);
z_max = EEPROM.read(11) * 256;
z_max += EEPROM.read(12);
modul = EEPROM.read(13) * 256;
modul += EEPROM.read(14);

x_0 /= 1000;                    //proračun sa podacima iz EEPROM-a
y_0 /= 1000;
z_0 /= 1000;
x_max /= 1000;
y_max /= 1000;
z_max /= 1000;
modul /= 1000;

x_y = (x_max - x_0) / (y_max - y_0);
x_z = (x_max - x_0) / (z_max - z_0);

Serial1.print();                //ispis svih proračunatih vrijednosti

...

compass.read();                 //aktivacija akcelerometra i
                                  magnetometra
x_acc = (int)compass.a.x;       //čitanje podataka iz akcelerometra
y_acc = (int)compass.a.y;
z_acc = (int)compass.a.z;

x_mag = (int)compass.m.x;       //čitanje podataka iz megnetometra
y_mag = (int)compass.m.y;
z_mag = (int)compass.m.z;

x_mag /= 1055;                  //podešavanje magnetometra sa podacima
y_mag /= 950;                   //iz EEPROM-a
z_mag /= 1055;
x_mag -= x_0;
y_mag -= y_0;
z_mag -= z_0;
y_mag *= x_y;
z_mag *= x_z;
```

```
x_acc /= -16384;
y_acc /= -16384;
z_acc /= -16384;

Serial1.print(x_mag,3);
Serial1.print(y_mag,3);
Serial1.print(z_mag,3);
roll = atan2(y_acc, z_acc);   //proračun roll podatka
pitch = atan2(x_acc, sqrt(y_acc * y_acc + z_acc * z_acc));
pitch *= (-1);                //proračun pitch podatka

Serial1.print(roll,3);        //ispis roll podatka
Serial1.print(pitch,3);       //ispis pitch podatka

hx = x_mag*cos(pitch)+y_mag*sin(pitch)*sin(roll)+z_mag*cos(roll)*sin(pitch);
hy = y_mag*cos(roll)-z_mag*sin(roll);
yaw = atan2(-hy, hx);         //proračun roll podatka

xrp[0][0] = roll;
xrp[1][0] = pitch;
XY = yaw;
```

4.4. Glavna petlja za prikupljanje i obradu podataka

Glavna petlja programa se izvršava brzinom $10Hz$. Podatke sa GPS modula dohvaćamo pomoću prekidne funkcije tako da pročitani bajt sa RX2 pina spremamo u polje sve dok nije pročitan zadnji podatak u GPS poruci. GPGGA poruka iz GPS modula se koristi za dobivanje GPS lokacije, UTC vremena i HDOP. Arduino paralelno prima podatke i sa IMU-a. Pomoću podataka iz IMU-a proračunavamo roll, pitch i yaw. Podaci sa akcelerometra se koriste za proračun roll i pitch komponente, a yaw je dobiven iz roll, pitch komponente i magnetometra. Za preciznija mjerenja podatke iz IMU-a filtriramo kroz Kalmanov filter sa podacima iz žiroskopa.

```
if(new_gga){
      p_c = 0;
      zar = 0;
      N = 1;
      E = 1;
      T = 0;
      hd = 0;
      do {
            if (zar == 1)
                  time[T++] = gga[p_c];
            if (zar == 2)
                  lat[N++] = gga[p_c];
            if (zar == 3) {
                  if (gga[p_c++] == 'N')
                        lat[0] = '+';
                  else
                        lat[0] = '-';
                        }
            if (zar == 4)
                  lon[E++] = gga[p_c];
```

```
            if (zar == 5) {
                  if (gga[p_c++] == 'E')
                        lon[0] = '+';
                  else
                        lon[0] = '-';
                        }
            if (zar == 8)
                  hdop[hd++] = gga[p_c];
                        if (gga[p_c] == ',')
                              zar++;
                  p_c++;
            }while(gga[p_c]);
            gga = "";
            new_gga = 0;
}
```

Navedeni isječak koda vezan je za GPS poruku. Nakon što preko prekida (eng. *interrupt)* bude pročitana i provjerena točnost GPS poruke, dolazi do glavne petlje u kojoj se sortiraju podaci iz glavnog polja u kojem je bila spremljena GPS poruka. Podaci koji su sortirani su lat, lon, hdop i time. Svaki podatak se nalazi u zasebnom polju.

```
t.update();
```

Funkcija koja se konstantno poziva svakih 100*ms*. Njena uloga je da točno u zadanom intervalu pročita podatke sa IMU-a.

```
if (isupdate) {
if (digitalRead(8))
      cal++;
else
      cal = 0;
if (cal == 10) {
      mag_cal();
      cal = 0;
}
```

Ponekad se na pinovima Arduina zna pojaviti kratka naponska smetnja koja može rezultirati aktivacijom pina 8 koji pokreće kalibraciju. Da bi izolirali takve pojave, navedeni dio koda će se pobrinuti da na pinu 8 treba biti naponski impuls u trajanju 1*s* ukoliko želimo aktivirati kalibraciju IMU-a.

```
x_mag = (int)compass.m.x;
y_mag = (int)compass.m.y;
z_mag = (int)compass.m.z;

x_acc = (int)compass.a.x;
y_acc = (int)compass.a.y;
z_acc = (int)compass.a.z;

x_gyr = (int)gyro.g.x;
y_gyr = (int)gyro.g.y;
z_gyr = (int)gyro.g.z;                                    1.dio
```

```
x_mag /= 1055;
y_mag /= 950;
z_mag /= 1055;
x_mag -= x_0;
y_mag -= y_0;
z_mag -= z_0;
```

```
y_mag *= x_y;
z_mag *= x_z;

x_acc /= -16384;
y_acc /= -16384;
z_acc /= -16384;

mm = sqrt(x_mag*x_mag+y_mag*y_mag+z_mag*z_mag);

x_gyr *= 0.00875 / 180 * M_PI;
y_gyr *= 0.00875 / 180 * M_PI;
z_gyr *= 0.00875 / 180 * M_PI;

mmm = mm/modul;

Ry = 10;
if (mmm < 0.98 || mmm > 1.02)
        Ry = 100;
if (mmm < 0.93 || mmm > 1.07)
        Ry = 500000;

roll = atan2(y_acc, z_acc);
pitch = atan2(x_acc, sqrt(y_acc * y_acc + z_acc * z_acc));
pitch *= (-1);

xrp_[0][0] = xrp[0][0]+(x_gyr + y_gyr*sin(xrp[0][0])*tan(xrp[1][0])+
z_gyr*cos(xrp[0][0])*tan(xrp[1][0]))*0.1;
xrp_[1][0] = xrp[1][0] + (y_gyr*cos(xrp[0][0]) - z_gyr*sin(xrp[0][0]))*0.1;
xy_ = XY + (y_gyr*sin(xrp[0][0])/cos(xrp[1][0]) + z_gyr*cos(xrp[0][0])/cos(xrp[1][0]))*0.1;

Frp[0][0] = 1 + (tan(xrp[1][0])*(y_gyr*cos(xrp[0][0]) - z_gyr*sin(xrp[0][0]))*0.1);
Frp[0][1] = (y_gyr*sin(xrp[0][0])+z_gyr*cos(xrp[0][0]))*0.1/cos(xrp[1][0])/cos(xrp[1][0]);
Frp[1][0] = -(y_gyr*sin(xrp[0][0]) + z_gyr*cos(xrp[0][0]))*0.1;
Frp[1][1] = 1;

t2x2(Frp, EKF2);
m2x2(Prp, EKF2, Prp_);
m2x2(Frp, Prp_, EKF2);
s2x2(EKF2, Qrp, Prp_);

yrp[0][0] = ((roll - xrp_[0][0]));
yrp[1][0] = ((pitch - xrp_[1][0]));

t2x2(Hrp,EKF2);
m2x2(Prp_, EKF2, Srp);
m2x2(Hrp, Srp, EKF2);
s2x2(EKF2, Rrp, Srp);

i2x2(Srp, Krp);
m2x2(Hrp, Krp, EKF2);
m2x2(Prp_, EKF2, Krp);

m2x1(Krp, yrp, EKF1);
s2x1(xrp_, EKF1, xrp);

m2x2(Krp, Hrp, Prp);
r2x2(Irp, Prp, EKF2);
m2x2(EKF2, Prp_, Prp);

Fy = 1;

Py_ = Fy*Py*Fy + Qy;

hx = x_mag * cos(xrp[1][0]) + y_mag * sin(xrp[1][0]) * sin(xrp[0][0])+ z_mag *
cos(xrp[0][0]) * sin (xrp[1][0]);
hy = y_mag * cos(xrp[0][0]) - z_mag * sin(xrp[0][0]);
yaw = atan2(-hy,hx);

yy = asin(sin(yaw-XY));

Sy = Hy*Py_*Hy + Ry;

Ky = Py_*Hy/Sy;

XY = xy_ + Ky*yy;

Py = (Iy - Ky*Hy)*Py_;                                                    2.dio
```

```
Serial1.write(0xFF);
Serial1.write(0xFF);
Serial1.write(0xFF);
Serial1.write(0xFF);
time[T-1] = '\0';
Serial1.println(time);
tm = atof(time);

chptr = (unsigned char *) &tm;
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
time[0] = '\0';

lat[N-1] = '\0';
Serial1.println(lat);
lt = atof(lat);
chptr = (unsigned char *) &lt;
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
lat[0] = '\0';

lon[E-1] = '\0';
Serial1.println(lon);
ln = atof(lon);
chptr = (unsigned char *) &ln;
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
lon[0] = '\0';

hdop[hd-1] = '\0';
Serial1.println(hdop);
h = atof(hdop);
chptr = (unsigned char *) &h;
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
hdop[0] = '\0';

Serial1.println(x_acc,3);
chptr = (unsigned char *) &x_acc;
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &y_acc;
Serial1.println(y_acc,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &z_acc;
Serial1.println(z_acc,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &x_gyr;
Serial1.println(x_gyr,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &y_gyr;
Serial1.println(y_gyr,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &z_gyr;
Serial1.println(z_gyr,3);
```

```
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &x_mag;
Serial1.println(x_mag,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &y_mag;
Serial1.println(y_mag,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &z_mag;
Serial1.println(z_mag,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
Serial1.println("R");
chptr = (unsigned char *) &xrp[0][0]; //rad
Serial1.println(xrp[0][0],3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
Serial1.println("P");
chptr = (unsigned char *) &xrp[1][0]; //rad
Serial1.println(xrp[1][0],3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
Serial1.println("Y");
chptr = (unsigned char *) &XY; //rad
Serial1.println(XY,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &modul;
Serial1.println(modul,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &Ry;
Serial1.println(Ry,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &mmm;
Serial1.println(mmm,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
chptr = (unsigned char *) &mm;
Serial1.println(mm,3);
for(i=0; i<4;i++) {
        Xor ^= *chptr;
        //Serial1.write(*chptr++);
}
Serial1.write(Xor);
Xor = 0;
isupdate = 0;                                                    3.dio
```

U ovom dijelu programa se izvršava glavni algoritam koji smo podijelili u tri dijela. Prvi dio je čitanje svih podataka sa IMU senzora. Drugi dio koda je proračun roll, pitch i

yaw komponenti kroz Kalmanov filter. Zadnji dio služi za pretvorbu i ispis podataka u binarni niz znamenaka.

4.5. Funkcija prekida (engl. Interrupt) (GPS podaci)

Arduino podržava vrlo jednostavan način pozivanja prekidne funkcije zbog toga što je već implementirana i povezana sa serijskim portovima. Funkcija kojom se poziva prekid na određenom serijskom portu ima oblik void SerialEvent(). GPS se nalazi na drugom serijskom portu pa će naša funkcija biti zapisana void SerialEvent2().

```
void serialEvent2() {
      while (Serial2.available()) {
            char udat = (char)Serial2.read();

            switch(udat){
                  case('$'):
                        pttr = "";
                        new_msg = 1;
                        xili = 0;
                  break;
                  case('*'):
                        if(new_msg){
                              new_msg = 0;
                              while(!Serial2.available());
                              c1 = (char)Serial2.read();
                              while(!Serial2.available());
                              c2 = (char)Serial2.read();

                              if (c1 >= '0' && c1 <= '9')
                                    c1 -= 48;
                              if (c1 >= 'A' && c2 <= 'F')
                                    c1 -= 55;
                              if (c2 >= '0' && c2 <= '9')
                                    c2 -= 48;
                              if (c2 >= 'A' && c2 <= 'F')
                                    c2 -= 55;
                              if (c1*16+c2 == xili) {
                                    if (!pttr.indexOf("GPGGA")) {
                                          gga = pttr;
                                          new_gga = 1;
                                    }
                              }
                        }
                  break;
                  default:
                        if(new_msg){
                              pttr += udat;
                              xili ^= udat;
                        }
            }
      }
}
```

Prekidna funkcija čita poruku koja se nalazi na izlazu serijskog porta GPS-a i provjerava da li je prvi znak $. Ukoliko se radi o novoj poruci svaki bajt se pretvara, a zatim xor-a sa prethodnim. Cijela xor-ana poruka mora biti jednaka vrijednosti na kraju poruke (vrijednost poslije oznake puta * tzv. checksum). Vrijednosti bi se trebale podudarati, pa ako se podudaraju primljena poruka se prosljeđuje u glavni program.

4.6. Funkcija za čitanje podataka sa IMU-a

Čitanje podataka sa IMU-a većim dijelom realizirana je preko gotovih funkcija koje su implementirane u L3G.h i LSM303.h.

```
void imu(){
        gyro.read();
        compass.read();
        isupdate = 1;
}
```

Svakim pozivom ove funkcije pročitamo podatke sa akcelerometra, magnetometra i žiroskopa za sve tri osi.

4.7. Funkcija za kalibraciju magnetometra

Kalibraciju magnetometra pokrećemo pozitivnim bridom napona $5V$ na 8. Pin Arduina. U procesu kalibracije prvo kalibriramo x-y ravninu, a zatim x-z ravninu. Zatim se računa modul X, Y i Z osi.

Proces kalibracije uzima 100 mjerenja X i Y osi, tokom mjerenja IMU je potrebno rotirati u zadanoj ravnini. Isti postupak se ponavlja za X i Z osi. Zadnji korak kalibracije je izračun modula magnetskih polja za sve tri osi, kako bi se mogla detektirati promjena magnetskog polja oko IMU-a sa obzirom na izračunate vrijednosti. Kalibrirani podaci se spremaju u EEPROM kako se ne bi morala vršiti kalibracija kod svakog paljenja.

```
void mag_cal(void) {
        x_min = 10000;
        x_max = -10000;
        y_min = 10000;
        y_max = -10000;
        z_min = 10000;
        z_max = -10000;
        Serial1.print("\r\nKalibracija x-y ravnine magnetometra za:");
        delay(1000);
        Serial1.print("\r\n5");
        delay(1000);
        Serial1.print("\r\n4");
        delay(1000);
        Serial1.print("\r\n3");
        delay(1000);
        Serial1.print("\r\n2");
```

```
        delay(1000);
        Serial1.print("\r\n1");
        delay(1000);
        Serial1.print("\r\n");

        for (k=0; k<100; k++) {
                delay(50);
                compass.read();
                x_mag = (int)compass.m.x;
                y_mag = (int)compass.m.y;
                z_mag = (int)compass.m.z;

                x_mag /= 1055;
                y_mag /= 950;
                z_mag /= 1055;

                Serial1.print(x_mag,3);
                Serial1.print("\r\n");
                Serial1.print(y_mag,3);
                Serial1.print("\r\n");

                xy[k%5] = x_mag;
                yyy[k%5] = y_mag;

                for (i=0; i<5; i++) {
                        x[i] = xy[i];
                        y[i] = yyy[i];
                }

                for(i=0; i<5-1; i++) {
                        for(j=i+1; j<5; j++) {
                                if(x[j] < x[i]) {
                                        temp = x[i];
                                        x[i] = x[j];
                                        x[j] = temp;
                                }
                        }
                }
                for(i=0; i<5-1; i++) {
                        for(j=i+1; j<5; j++) {
                                if(y[j] < y[i]) {
                                        temp = y[i];
                                        y[i] = y[j];
                                        y[j] = temp;
                                }
                        }
                }
                if (x[2] < x_min)
                        x_min = x[2];
                if (x[2] > x_max)
                        x_max = x[2];
                if (y[2] < y_min)
                        y_min = y[2];
                if (y[2] > y_max)
                        y_max = y[2];

                Serial1.print(x_min,3);
                Serial1.print("\r\n");
                Serial1.print(x_max,3);
                Serial1.print("\r\n");
                Serial1.print(y_min,3);
                Serial1.print("\r\n");
                Serial1.print(y_max,3);
                Serial1.print("\r\n");
                Serial1.print(k);
                Serial1.print("\r\n\r\n");
                delay(50);
        }

        x_0 = x_max - ((x_max - x_min) / 2);
        y_0 = y_max - ((y_max - y_min) / 2);
        x_y = (x_max - x_0) / (y_max - y_0);

        x_0 *= 1000;
        y_0 *= 1000;
        x_max *= 1000;
        y_max *= 1000;
```

```
        EEPROM.write(1, (int)x_0 >> 8);
        EEPROM.write(2, ((int)x_0)%256);
        EEPROM.write(3, (int)y_0 >> 8);
        EEPROM.write(4, ((int)y_0)%256);
        EEPROM.write(7, (int)x_max >> 8);
        EEPROM.write(8, ((int)x_max)%256);
        EEPROM.write(9, (int)y_max >> 8);
        EEPROM.write(10, ((int)y_max)%256);

        Serial1.print("\r\n");
        Serial1.print("\r\n");
        Serial1.print(x_0,3);
        Serial1.print("\r\n");
        Serial1.print(y_0,3);
        Serial1.print("\r\n");
        Serial1.print(x_y,3);
        Serial1.print("\r\n");
        Serial1.print("\r\n");

        x_0 /= 1000;
        y_0 /= 1000;

        x_min = 10000;
        x_max = -10000;

        while(!digitalRead(8));
        Serial1.print("\r\nKalibracija x-z ravnine magnetometra za:");
        delay(1000);
        Serial1.print("\r\n5");
        delay(1000);
        Serial1.print("\r\n4");
        delay(1000);
        Serial1.print("\r\n3");
        delay(1000);
        Serial1.print("\r\n2");
        delay(1000);
        Serial1.print("\r\n1");
        delay(1000);
        Serial1.print("\r\n");

        for (k=0; k<100; k++) {
                delay(50);
                compass.read();
                x_mag = (int)compass.m.x;
                y_mag = (int)compass.m.y;
                z_mag = (int)compass.m.z;

                x_mag /= 1055;
                y_mag /= 950;
                z_mag /= 1055;

                Serial1.print(x_mag,3);
                Serial1.print("\r\n");
                Serial1.print(z_mag,3);
                Serial1.print("\r\n");

                xz[k%5] = x_mag;
                zz[k%5] = z_mag;

                for (i=0; i<5; i++) {
                        x[i] = xz[i];
                        z[i] = zz[i];
                }

                for(i=0; i<5-1; i++) {
                        for(j=i+1; j<5; j++) {
                                if(x[j] < x[i]) {
                                        temp = x[i];
                                        x[i] = x[j];
                                        x[j] = temp;
                                }
                        }
                }
                for(i=0; i<5-1; i++) {
                        for(j=i+1; j<5; j++) {
                                if(z[j] < z[i]) {
```

```
                                    temp = z[i];
                                    z[i] = z[j];
                                    z[j] = temp;
                            }
                    }
            }
            if (x[2] < x_min)
                    x_min = x[2];
            if (x[2] > x_max)
                    x_max = x[2];
            if (z[2] < z_min)
                    z_min = z[2];
            if (z[2] > z_max)
                    z_max = z[2];

    Serial1.print(x_min,3);
    Serial1.print("\r\n");
    Serial1.print(x_max,3);
    Serial1.print("\r\n");
    Serial1.print(z_min,3);
    Serial1.print("\r\n");
    Serial1.print(z_max,3);
    Serial1.print("\r\n");
    Serial1.print(k);
    Serial1.print("\r\n\r\n");
    delay(50);
    }

    z_0 = z_max - ((z_max - z_min) / 2);
    x_z = (x_max - x_0) / (z_max - z_0);

    while(!digitalRead(8));
    Serial1.print("\r\nKalibracija modula magnetometra za:");
    delay(1000);
    Serial1.print("\r\n5");
    delay(1000);
    Serial1.print("\r\n4");
    delay(1000);
    Serial1.print("\r\n3");
    delay(1000);
    Serial1.print("\r\n2");
    delay(1000);
    Serial1.print("\r\n1");
    delay(1000);
    Serial1.print("\r\n");

    x_mag_m = 0;
    y_mag_m = 0;
    z_mag_m = 0;

    for (k=0; k<50; k++) {
            delay(50);
            compass.read();
            x_mag = (int)compass.m.x;
            y_mag = (int)compass.m.y;
            z_mag = (int)compass.m.z;
            x_mag /= 1055;
            y_mag /= 950;
            z_mag /= 1055;
            x_mag -= x_0;
            y_mag -= y_0;
            z_mag -= z_0;
            y_mag *= x_y;
            z_mag *= x_z;
            x_mag_m += x_mag;
            y_mag_m += y_mag;
            z_mag_m += z_mag;

            delay(50);
    }

    x_mag_m /= 50;
    y_mag_m /= 50;
    z_mag_m /= 50;
    modul = sqrt(x_mag_m*x_mag_m+y_mag_m*y_mag_m+z_mag_m*z_mag_m);

    z_0 *= 1000;
```

```
        z_max *= 1000;
        modul *= 1000;

        Serial1.print(modul,3);
        Serial1.print("\r\n");
        delay(2000);
        EEPROM.write(5, (int)z_0 >> 8);
        EEPROM.write(6, ((int)z_0)%256);
        EEPROM.write(11, (int)z_max >> 8);
        EEPROM.write(12, ((int)z_max)%256);
        EEPROM.write(13, (int)modul >> 8);
        EEPROM.write(14, ((int)modul)%256);

        Serial1.print("\r\n");
        Serial1.print("\r\n");
        Serial1.print(x_0,3);
        Serial1.print("\r\n");
        Serial1.print(z_0,3);
        Serial1.print("\r\n");
        Serial1.print(x_z,3);
        Serial1.print("\r\n");
        Serial1.print("\r\n");
        delay(2000);

        x_0 = EEPROM.read(1) * 256;
        x_0 += EEPROM.read(2);
        y_0 = EEPROM.read(3) * 256;
        y_0 += EEPROM.read(4);
        z_0 = EEPROM.read(5) * 256;
        z_0 += EEPROM.read(6);
        x_max = EEPROM.read(7) * 256;
        x_max += EEPROM.read(8);
        y_max = EEPROM.read(9) * 256;
        y_max += EEPROM.read(10);
        z_max = EEPROM.read(11) * 256;
        z_max += EEPROM.read(12);
        modul = EEPROM.read(13) * 256;
        modul += EEPROM.read(14);

        x_0 /= 1000;
        y_0 /= 1000;
        z_0 /= 1000;
        x_max /= 1000;
        y_max /= 1000;
        z_max /= 1000;
        modul /= 1000;

        x_y = (x_max - x_0) / (y_max - y_0);
        x_z = (x_max - x_0) / (z_max - z_0);

        Serial1.print("\r\n");
        Serial1.print(x_0,3);
        Serial1.print("\r\n");
        Serial1.print(y_0,3);
        Serial1.print("\r\n");
        Serial1.print(z_0,3);
        Serial1.print("\r\n");
        Serial1.print(x_y,3);
        Serial1.print("\r\n");
        Serial1.print(x_z,3);
        Serial1.print("\r\n");
        Serial1.print(x_max,3);
        Serial1.print("\r\n");
        Serial1.print(y_max,3);
        Serial1.print("\r\n");
        Serial1.print(z_max,3);
        Serial1.print("\r\n");
        Serial1.print(modul,3);
        Serial1.print("\r\n");
        delay(5000);
}
```

5. Razmatranje i rezultati

5.1. Konačan opis rada

Da bi cijeli navigacijski sustav radio, mora se spojiti izvor napajanja na Arduino pločicu u rasponu $6-12V$. Ispravnost Arduino pločice prepoznat ćemo po paljenju led-diode pod nazivom ON. Ukoliko smo navigacijski sustav spojili po prvi put na ronilicu obavezno je izvršiti kalibraciju. Kalibraciju ćemo izvršiti dovođenjem kratkog impulsa na pin 8 Arduino pločice. Tijekom kalibracije potrebno je rotirati sustav u x-y ravnini, a zatim isti postupak ponoviti i za x-z ravninu. Zadnji korak kalibracije je da sustav mirno postavimo u x-y ravninu zbog izračuna modula magnetskih polja.

Nakon što su podešeni svi podaci koji su potrebni za ispravan rad ovog navigacijskog sustava možemo ga implementirati u ronilicu. Podaci koje ćemo dobivati sa ronilice su njena orijentacija, geografski položaj i vrijeme.

5.2. Poboljšanje projekta u budućnosti

Sklop projekta je realiziran na protoboard pločici. Kao što je navedeno Arduino karakterizira jednostavnost zbog toga što se mogu projektirati moduli koji se zatim jednostavno mogu priključiti na pinove Arduino pločice. Također postoje već projektirani GPS i IMU moduli koji bi se jednostavno spojili na Arduino i tako činili kompaktnu cjelinu bez ikakve potrebe za projektiranjem novog sustava. U ovom projektu koristili smo GPS i IMU kakav je korišten na prijašnjem PIC sustavu, kako bi implementacija algoritama bila što jednostavnija. Ukoliko bi išli poboljšavati sustav, sigurno bi bilo puno lakše i kompatibilnije sa Arduinom da koristimo gotove module predviđene za njega.

Implementacija algoritma se pokušala izvesti na način da se što manje mora mijenjati kod programa, dok je s jedne strane to pojednostavilo problem, s druge je bilo komplikacija. Ukoliko bi se koristili navedeni moduli koji su predviđeni za Arduino također bi upotrijebili gotove napisane Library-e koji bi definirali GPS i IMU, pa bi čitanje, provjera i ispisivanje podataka iz senzora bila još jednostavnija. Kalmanov filter također je implementiran pomoću Library paketa, tako da su preinake u sklopu i algoritmu uvijek otvorene i nužne.

6. Zaključak

U ovom završnom radu je predstavljena implementacija navigacijskog algoritma na Arduino pločicu. Na početku rada je analiziran problem navigacije i orijentacije ronilica pod vodom i podvodnih sustava općenito. Zatim je ponuđeno osnovno rješenje problema koji kao takav pokriva većinu problematike sustava. Predstavljen je postojeći sustav koji treba realizirati na Arduino pločici. Opisani su sklopovi korišteni za razvoj projekta, kao i programska rješenja. Cijeli sklop je povezan sa ArduinoMega2560 pločicom koji je bilo potrebno programirati sa modificiranim algoritmom iz PIC mikrokontrolera. Sustav je uspješno sastavljen kao i implementacija algoritma.

Sklop predstavlja jedinstven i funkcionalan navigacijski sustav na koji ne utječu nikakve veće smetnje i ne utječe na rad ronilice, a pruža korisniku jeftinu i sigurnu navigaciju. Sklop je u budućnosti otvoren novim idejama i prijedlozima.

7. Literatura

1. Getting Started w/ Arduino on Windows, http://arduino.cc/en/Guide/Windows, 2013
2. Global Positioning System, http://hr.wikipedia.org/wiki/Global_Positioning_System, 2013
3. Inertial measurement unit, http://en.wikipedia.org/wiki/Inertial_measurement_unit, 2013

8. Sažetak

U ovom završnom radu je projektiran sustav koji služi kao navigacija bespilotnim ronilicama. Taj navigacijski sustav šalje podatke o orijentaciji ronilice u prostoru. Budući da se u sustavu nalaze dvije skupine navigacijskih senzora sa točnošću možemo dobiti podatke o nagibu i rotaciji ronilice (inercijski senzor pokreta) odnosno položaj u prostoru (GPS).

Predstavljeno je rješenje sklopa koje je implementirano na Arduino pločici koja služi kao osnovna logička jedinica, po potrebi autonoman sustav napajanja, GPS koji služi za dobivanje koordinata lokacije sustava i inercijski senzor pokreta pomoću kojeg saznajemo orijentaciju sustava.

Navigacijski algoritam kojim je isprogramiran Arduino bio je napisan za sustav koji se bazirao na PIC mikrokontroleru, te je navedeni kod bilo potrebno modificirati i implementirati u sustav napravljen na Arduino pločici. Cijeli kod je napisan i kompajliran u programskom paketu „Arduino“.

Na kraju rada su ponuđene ideje za buduće poboljšanje sustava.

9. Abstract

As part of this final thesis navigation system for autonomous underwater vehicle was developed. Navigation system transmits orientation data of underwater vehicle. Since system is comprised of two groups of navigation sensor it can provide accurate data about inclination, rotation (internal motion detector) and position in space (GPS).

Presented solution is a switch which is implemented on Arduino. It serves as basic logical unit, autonomous power source as needed, GPS which is used for retrieving location coordinate and inertial motion detector which can detect system orientation.

Navigation algorithm, which is used by Arduino, originally was developed for system based on PIC micro controller hence it had to be modified and implement it for system based on Arduino. All programming code was written and compiled in programming pack „Arduino“.

At the end of this thesis it is discussed about future improvements of system.