UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

SIMULÁTOR ROBOTICKÝCH PLACHETNÍC

2011

Pavol Hudec

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

SIMULÁTOR ROBOTICKÝCH PLACHETNÍC

Bakalárska práca

Študijný program : Aplikovaná informatika
Študijný odbor: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školitel: Mgr. Pavel Petrovič, PhD.

Evidenčné číslo: a3f2c7d6-b970-4b80-bf6e-700b14c6d2ac

Bratislava 2011

Pavol Hudec

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Pavol Hudec
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Simulátor robotických plachetníc

Cieľ: Naši partneri z Austrian Association for Innovative Computer Science (INNOC) sú intenzívne zapojení do organizácie Majstrovstiev sveta robotických plachetníc - v roku 2008 zorganizovali majstrovstvá v Rakúsku. Autonómne plachetnice sú lode, ktoré sa dokážu pohybovať na jazerách a na mori po zadanej trajektórii, stráviť na vode niekoľko dní a pod. Lode sú poháňané iba silou vetra. Tímy sa teraz pripravujú na plavbu krížom cez Atlantický oceán a potrebovali by testovať svoje riadiace programy v simulácii, ktorý by simuloval správanie lode za rozličných poveternostných podmienok. Výsledkom projektu by mal byť funkčný simulátor, prácu je možné rozdeliť na viacero podtém. Téma je vhodná aj na diplomovú prácu. INNOC má svoj klub vo Viedni, v Rakúsku a jeho členovia navštevujú Bratislavu. Študent bude úzko spolupracovať a komunikovať s členmi tímu INNOC. (pozn. s členmi INNOC sme zapojení do spoločného medzinárodného projektu Centrobot). Požiadavky: dobrá komunikácia v angličtine (nemčine)

Vedúci: Mgr. Pavel Petrovič, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 01.10.2010

Dátum schválenia: 25.10.2010

doc. RNDr. Mária Markošová, PhD.
garant študijného programu

.....
študent

.....
vedúci

ČESTNÉ VYHLÁSENIE

Vyhlasujem, že som bakalársku prácu s názvom „Simulátor robotických plachetníc“ spracoval samostatne, pod odborným vedením vedúceho bakalárskej práce, na základe vlastných teoretických poznatkov a s použitím uvádzanej literatúry.

V Bratislave dňa 30.5.2011 ….....

POĎAKOVANIE

Ďakujem vedúcemu mojej bakalárskej práce, Mgr. Pavlovi Petrovičovi PhD, za venovaný čas, ochotu, rady a pripomienky pri vypracovávaní tejto bakalárskej práce.

Abstrakt

HUDEC, Pavol: Simulátor robotických plachetníc [Bakalárska práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej informatiky. Vedúci bakalárskej práce: Mgr. Pavel Petrovič, PhD. Bratislava: FMFI UK, 2011. Spoločnosť INNOC vlastní robotickú plachetnicu a podieľa sa na celosvetových súťažiach robotických plachetníc. Svoje riadiace algoritmy testuje na mori, čo je nepraktické, časovo náročné a často krát aj nedostupné. Cieľom bakalárskej práce bolo vytvoriť simulátor pre túto robotickú plachetnicu, aby mohli svoje riešenia testovať rýchlejšie a častejšie. Simulátor je zaintegrovaný do ich riešenia. Loď používa blackboard architektúru (tabuľová architektúra). Jednotlivé moduly komunikujú medzi sebou len prostredníctvom tabule, na ktorú môžu písať a čítať z nej. Simulátor simuluje plachetnicu pomocou neurónovej siete, ktorá sa naučí potrebné informácie z log súboru z reálnej plavby na mori.

Kľúčové slová: simulácia, neurónová sieť, plachetnica, blackboard

Abstract

HUDEC, Pavol: Robotic sailboats simulator [Bachelor thesis]. Comenius University in Bratislava. Faculty Of Mathematics Physics And Informatics; Department Of Applied Informatics. Supervisor: Mgr. Pavel Petrovic, PhD. Bratislava: FMFI UK, 2011. The INNOC company has a robotic sailboat and participates in robotic sailboat worldwide competitions. Their control algorithms tested at sea, witch was impractical, time consuming and often even unavailable. Aim of this thesis was to develop a simulator for this robotic sailboat in order to test their solutions faster and more frequently. The simulator is integrated within their solutions. Their ship uses a blackboard architecture. The individual modules are communicating with each other only through the blackboard on which they can write and read from it. Simulator simulates sailing with help of a neural network witch learns the necessary information from the log file witch is from real voyage at sea.

Key words: simulation, neural network, sailboat, blackboard

Obsah

1 Úvod

Spoločnosť INNOC je nezávislý výskumný inštitút so sídlom vo Viedni. Vedie medzinárodne známy výskum a vývoj v oblasti inovatívnych počítačových technológií, a to najmä v oblasti umelej inteligencie a robotiky. Zúčastňuje sa Majstrovstiev sveta autonómnych robotických plachetníc. Slovom autonómna je myslené, že plachetnica sa sama dokáže pohybovať na mori, s cieľom dostať sa do vopred určeného cieľa, čo možno najrýchlejšie. Plachetnica obsahuje aj motory, ale tie sú určené výlučne len na pohyb plachiet a kormidla. Poháňaná je len silou vetra.

Cestovať kvôli každej zmene v riadení lode, by bolo veľmi namáhavé a neefektívne. Testovanie riadenia plachetnice v reálnom prostredí je obmedzované aj počasím. Cieľom bakalárskej práce je vytvoriť funkčný simulátor robotickej plachetnice pre túto spoločnosť. Simulátor, ktorý simuluje správanie plachetnice za rôznych poveternostných podmienok by zjednodušil testovanie riadenia plachetnice. Naši partneri z INNOC by mohli testovať svoje algoritmy tu. Znamená to, že by mohli testovať aj počas ročných období, kedy to v reálnych podmienkach nie je možné. Riadenie lode by mohlo byť testované častejšie a stúpla by efektivita práce.

Autonómna robotická plachetnica môže mať viacero reálnych využití. Môže byť použitá ako plavidlo na prepravu tovaru. Človek by len zadal cieľ, kam má loď tovar priniesť, a sama by tam bez akejkoľvek ľudskej pomoci prišla. Keďže je táto plachetnica energeticky sebestačná, náklady na prevádzku by boli minimálne. Ďalšie možné využitie tejto plachetnice je výskum pre pasívne akustické monitorovanie morských cicavcov. O spôsobe života niektorých morských cicavcov vie ľudstvo málo, a plachetnica by mohla pomôcť pri ich skúmaní.

2 Prehľad problematiky

2.1 Hardware AVS Roboat

AVS Roboat je autonómna robotická plachetnica, ktorá patrí k najlepším na svete. Je vyvíjaná výskumným tímom spoločnosťou INNOC od roku 2006. Základom AVS Roboat je komerčná plachetnica, ktorú navrhol Jan Herman Ling. Loď bola pôvodne vytvorená pre deti na učenie plachtenia. Loď je dlhá 3.75 m a váži 60 kg. Sú v nej zabudované ťažké časti, ako batérie a multisenzory. Vrátane batérie má loď celkovú hmotnosť 300 kg. Plochy hlavnej a prednej plachty majú spolu 4,5 m2. Je vybavená solárnymi panelmi ktoré poskytujú 285 W energie pri najlepšom žiarení slnka a metanolovými palivovými článkami, ktoré poskytujú 65 W ako záložný zdroj energie. Roboat má trojstupňový komunikačný systém , spájajúci WLAN, UMTS / GPRS a satelitný komunikačný systém, ktorý umožňuje kontinuálny real-time prístup z brehu. Tento komunikačný systém možno využiť napríklad pre sledovanie a navigáciu lode z veliaceho centra na brehu. Kormidlo a plachty, sú samostatne kontrolované vstupnými údajmi z rôznych senzorov (GPS, kompas, atď), ktoré sa analyzujú na palubnom PC s operačným systémom Linux.

2.1.1 Senzory

Loď dokáže zisťovať potrebné informácie k plavbe z vonkajšieho okolia prostredníctvom senzorov. Senzory plachetnice dodávajú real-time informácie o aktuálnom smer a rýchlosti vetra. Sú merané náklon plachetnice (do bokov, aj dopredu – dozadu), geografická poloha a smer otočenia. Toto sú minimálne údaje potrebné pre uspokojivé správanie sa autonómnej plachetnice. Pri krátkodobom smerovaní je použitý radarový systém na detekciu pohybu prekážky.

2.2 Software AVS Roboat

2.2.1 Celková architektúra

Autonómne plachetnice musia vykonávať rad zložitých úloh k dosiahnutiu vopred definovaného cieľa. Pri plánovaní trasy je potrebné vziať do úvahy dlhodobé poveternostné podmienky a aj statické prekážky, ako sú napríklad ostrovy. Plachetnica

pracuje v prostredí, ktoré sa niekedy môže meniť, preto musí rýchlo reagovať na meniace sa podmienky. Mobilné autonómne robotické systémy sú zvyčajne rozdelené do samostatných vrstiev, kde každá rieši časť problému.

Sú väčšinou rozdelené do dvoch vrstiev. Existujú dve hlavné architektúry: Top-down plánovač a bottom-up reaktívny systém. V Top-down plánovači sa údaje so senzorov a vedomosti o prostredí použijú na vytvorenie modelu sveta v ktorom sa používa plánovanie. Plánovacie mechanizmy vytvoria podrobný plán pre robota na dosiahnutie vopred stanoveného cieľa. Tieto systémy sú vhodné na statické prostredie, pretože generovanie plánu je zvyčajne časovo náročné a tento systém nemôže rýchlo reagovať v dynamicky sa meniacom prostredí. Bottom-up reaktívny systém pripojí merané údaje snímačov priamo s mechanickými časťami robota, preto môže rýchlo reagovať na zmeny vo svete s nečakanými a pohybujúcimi sa prekážkami. Vzhľadom k tomu, že robot reaguje len na okamžité informácie, bez globálnych znalostí, nemusí dosiahnuť globálne optimálne riešenie a často chýba schopnosť vykonávať zložité operácie. Tieto dva prístupy sa môžu skombinovať- hybridná architektúra. Táto architektúra sa používa v ASV Roboat.

2.2.2 Hlavné časti software

Celý software možno rozdeliť na tri časti:

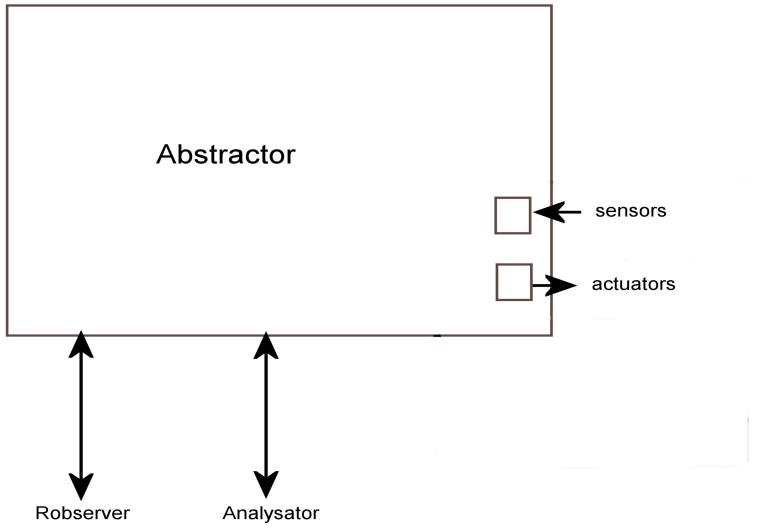
- Abstractor
- Analysator
- Robserver

Robserver ide len na PC (win/mac), Abstractor a Analysator idú na lodi (Linux, kompilovateľné aj pod Windows)

Abstractor

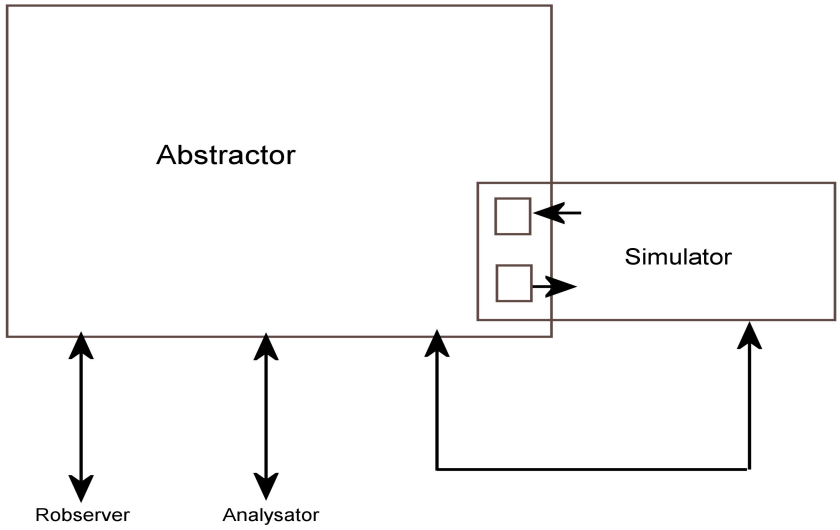
Abstractor je program písaný v jazyku Java, ktorý beží priamo na lodi. Slúži ako blackboard. Zbiera údaje zo senzorov a transformuje surové dáta do sémanticky použiteľných hodnôt. Predspracovanie, ako je tlmenie, škálovanie, transformácie jednotiek sú vykonávané na tejto úrovni. Má prístup k údajom o stave batérie, komunikácie, prostredia, smerovania, natočenia, naklonenia, o prekážkach, aktuálnej pozícií, pozícii

kormidla a plachiet, rýchlosti, cieľoch a vetre.



Obrázok 1. Prepojenie senzorov a aktivátorov so software cez abstractor.

Ak abstraktor nebeží na lodi, ale na počítači, potrebné údaje môže zbierať zo simulátora.



Obrázok 2. Prepojenie simulátora so software cez abstractor.

Analysator

Analysator zbiera potrebné informácie z abstractora. Analysator je program písaný v C++, ktorý riadi celú loď. Určuje kam má loď smerovať, požadovanú polohu kormidla a plachty, plánuje trasu.

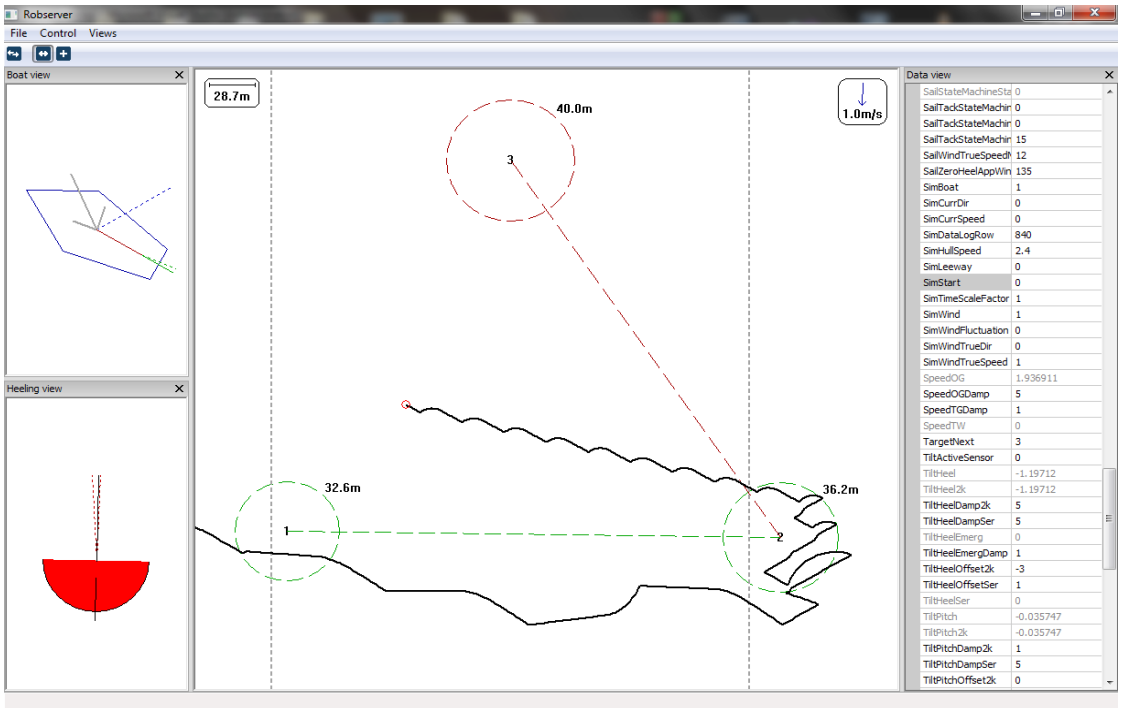
Robserver

Na zobrazenie plavby lode slúži robserver. Je to program písaný v C++, ktorý dokáže zobrazovať stav a pohyb plachetnice ak sa simuluje na počítači, ale aj ak pláva na vode. V plachetnici sú zabudované komunikačné moduly ako wifi, GPRS-UMTS, a irídiový satelit. To umožňuje vymieňanie údajov s počítačom.

Robserver zobrazuje trasu, ktorú plachetnica prešla, smer natočenia, veľkosť naklonenia, požadované smerovanie plachetnice, natočenie plachty a kormidla, smer a rýchlosť vetra a pod.

Loď môže počas bežných operácií posielať hodnoty senzorov do robservra. Robserver zobrazuje prijaté údaje ako pozícia alebo informácie o stratégií lode. Plachetnica je navrhnutá pre úplne autonómnu prevádzku. Avšak ľudský operátor musí preddefinovať strategické ciele. Keď ich udáva, nemá priamy vplyv na plánovanie cesty alebo vykonávanie manévrov. Takisto je možné posielať údaje o prekážkach alebo nový požadovaný smer.

Cez robserver sa dá konfigurovať správanie simulovaného prostredia zmenou rýchlosti vetra, zmenou sily unášania vetrom do strán a pod.



Obrázok 3. Robserver.

2.2.3 Blackboardova (tabuľová) architektúra

Blackboardova architektúra je úlohovo nezávislá architektúra pre integráciu riešenia problémov. Úlohovo nezávislou myslime, že je použiteľná pre široký rozsah úloh ako triedenie, diagnostika, opravy, výpočty a pod. Integrovane riešenia problémov, ako systémy na báze blackboard architektúry, sa pokúšajú prekonať obmedzenia jedného expertného systému zamestnaním dvoch alebo viacerých riešiacich modulov.

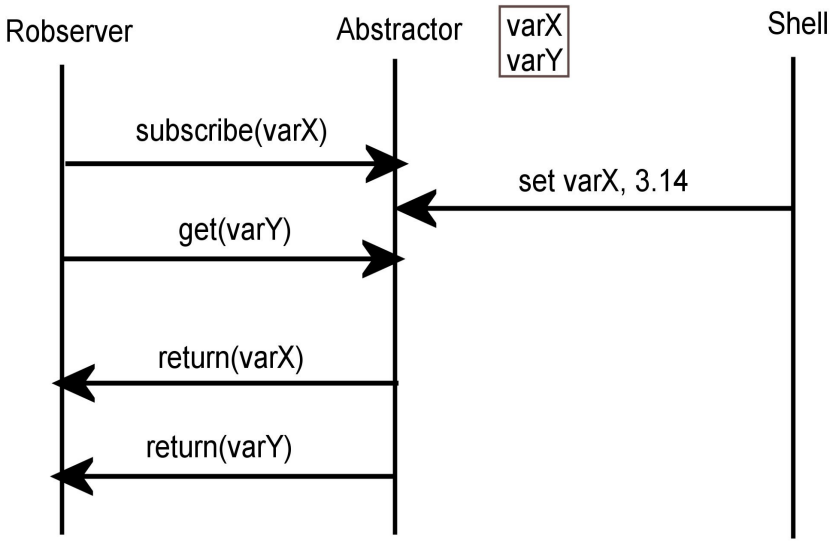
V blackboardovej architektúre sady riešiacich modulov (knowledge source) zdieľajú spoločnú globálnu databázu. Riešiace moduly reagujú na zmeny na tabuli a tiež v nej vykonávajú zmeny. Môžu robiť zmeny iba v tabuli, preto komunikujú a spolupracujú len cez tabuľu. Každý riešiaci modul reaguje iba na určitú množinu tried. V prípade potreby blackboard drží stav riešenia problému, kým riešiace moduly modifikujú blackboard.

Takáto architektúru používa aj AVS Roboat. V abstractore sú uložené všetky údaje, či už namerané a spracované zo senzorov, nastavené operátorom, alebo vypočítané. K abstractoru sa prostredníctvom socketov môže pripojiť ľubovolné množstvo programov. S abstractorom sa komunikuje pomocou príkazov:

- get – požiadavka na poslanie hodnoty premennej
- set – požiadavka na nastavenie hodnoty premennej
- subscribe - požiadavka na odoberanie premennej. Ak je premenná odoberaná, bude sa posielať podľa určitých pravidiel naspäť. Jedna premenná môže byť viac krát odoberaná.
- unsubscribe – požiadavka na zrušenie odoberania

Asynchrónnosť

Pri očakávaní vrátenia premennej môže nastať situácia, kedy abstractor vráti premenné v inom poradí ako boli na ne vykonávané dotazy.



Obrázok 4. Príklad asynchrónnosti v abstractore.

Na obrázku 4. je znázornený príklad situácie vracania premenných, ktorá môže nastať. V abstractore sú zadeklarované premenné varX a varY. Najprv sa nastaví odoberanie premennej varX, čo znamená, že pri každej zmene premennej varX sa pošle premenná robservru. Robserver pošle dotaz na vyžiadanie premennej varY. Ale medzi tým sa premenná varX zmenila. Znamená to, že abstractor má teraz poslať robservru varX aj varY a môže ich poslať v ľubovolnom poradí. Pri vrátení premennej sa okrem jej hodnoty pošle aj meno a typ premennej, takže robserver vie kedy sa ktoré premenné poslali.

Komunikácia

Na to, aby si jednotlivé časti vedeli posielať údaje, musia používať nejaký spôsob komunikácie a tiež komunikačný protokol. Tieto časti komunikujú medzi sebou prostredníctvom socketov. V komunikácií sa používajú tieto packety

- GET (0x00) – Pošle naspäť return packet, vždy len raz, socketu, ktorý poslal tento packet
- SET (0x01) – Nastaví hodnotu premennej. Ak je premenná odoberaná, pošle naspäť return packety všetkým socketom ktoré ju odoberajú. Return packet pošle aj keď sa hodnota premennej nezmení

- SUBSCRIBE (0x02) – Nastaví odoberanie premennej a pošle naspäť jeden return packet. Ak premenná ešte nemala nastavené odoberanie z daného socketu, nastaví sa jej odoberanie. Po tejto operácií menenie hodnoty danej premennej prostredníctvom SET spôsobí spätné poslanie return packetu socketu, ktorý vykonal SET. Ak už má nastavené odoberanie z daného packetu, pridá sa jej ešte jedno odoberanie. Potom pri operácií SET na danú premennú socketu príde toľko return packetov, koľko krát je premenná odoberaná.
- UNSUBSCRIBE (0x03) – Zruší odoberanie premennej, nepošle naspäť nič, ani keď premenná nie je odoberaná

Premenné

Id premennej sa skladá z dvoch Bytov

1. Byte – id vlastnosti, ktorej podkategória je daná premenná (napr. Vietor)
2. Byte – id premennej:
   („_“ - ľubovolné hexadecimálne číslo)

- 0_ - jednotlivé hodnoty (napr. rýchlosť, smer)
- 1_ - nastavenia konfigurácie (napríklad tlmenie, max)
- 2_ - alternatívne hodnoty (napr. čísla, podľa GPS)
- F_ - automatická kalibrácia (napr. 0-žiadna)

Dátový typ

- Skalár
  - 0x00 (8-bit, signed byte)
  - 0x01 (32-bit, signed long)
  - 0x02 (32-bit, float)
- Vektor
  - 0x03 (2x Skalár signed byte)
  - 0x04 (2x Skalár signed long)
  - 0x05 (2x Skalár float)
- String
  - 0x06 (1. Byte dĺžka + 8-bitové znaky)
- Boolean
  - 0x07 (0=False, 1=True)
- Príkaz
  - 0x08 Príkaz (nemá žiadne dátové časti)

- Špeciálne typy
    - 0x10 Polygon (1. Byte – počet, ostatné Byty - Vektor float)
    - 0x11 Indexovaný polygon (1. Byte – index, ostatné Byty - Polygon)
    - 0x12 Cieľ (pätica: dve súradnice, polomer, prebývanie, cestovný parameter - každý float => 5*4=20 Byte)
    - 0x13 Indexovaný cieľ (1. Byte – index, ostatné Byty - Cieľ)

Všetky skalárne typy sú vnútorne double, ale môžu byť prezentované ako iné typy.

Packety

- GET

1. Byte	2. + 3. Byte	4. Byte	5. Byte
GET	Premenná	Typ	Index

Packet GET sa skladá z 5 bytov. Prvý byte určuje druh akcie, čiže GET. Ten má hodnotu vždy 0x00. V druhom a treťom byte je uložené ID premennej, ktorej hodnotu chceme pomocou GET získať. Štvrtý byte udáva akého typu chceme mať vrátenú premennú. Piaty byte je index. Používa sa, len ak premenná je cieľ alebo prekážka. V takom prípade sa indexuje od 1. Nula znamená návrat všetkých cieľov alebo prekážok. Ak nejde o prekážku ani cieľ, tento údaj je ignorovaný.

- SET

1. Byte	2. + 3. Byte	4. Byte	5. Byte	6. - N. Byte
SET	Premenná	Typ	Dĺžka	Data

Prvý byte packetu SET určuje druh akcie – SET. Má hodnotu vždy 0x01. V druhom a treťom byte je uložené ID premennej, ktorej chceme nastaviť hodnotu. V štvrtom byte je typ hodnoty, ktorú chceme poslať. Piaty byte je dĺžka. Tento byte nie je nastavený na 1, iba v prípade, že premenná je cieľ. Dĺžka určí, koľko cieľov sa nastaví. Ostatné byty sú dáta veľkosti (sizeOf(Typ)*Dĺžka) a predstavujú hodnotu, ktorá sa premennej nastaví.

- SUBSCRIBE

1. Byte	2. + 3. Byte	Každé 4 Byty (Dĺžka krát)		
SUBSCRIBE	Dĺžka	Premenná (2 Byty)	Typ	Rýchlosť aktualizácie

Prvý byte packetu určuje druh akcie, čiže SUBSCRIBE. Má hodnotu vždy 0x02. Druhý a tretí byte označujú dĺžku. Je to vlastne počet premenných, ktorím sa má vykonať SUBSCRIBE. Ďalej nasledujú 4 byty, ktorých význam sa opakuje dĺžka krát. Prvé dva z týchto štyroch bytov znamenajú ID premennej. Tretí byte určuje aký typ premennej sa má posielať naspäť v return packete. Posledný z týchto štyroch bytov je rýchlosť aktualizácie. Ak je hodnota tohto bytu nula, return packet sa odošle pri SET packete danej premennej. Ak je hodnota iná, return packet sa odošle každých (rýchlosť odosielania) sekúnd. Momentálne je táto časť zakomentovaná a nepoužíva sa, takže má zmysel posielať len packety s rýchlosťou aktualizácie nastavenou na nula.

- UNSUBSCRIBE

1. Byte	2. + 3. Byte	Každé 3 Byty (Dĺžka krát)	
UNSUBSCRIBE	Dĺžka	Premenná	Typ

Prvý byte packetu určuje druh akcie – UNSUBSCRIBE. Má hodnotu vždy 0x03. Druhý a tretí byte udávajú dĺžku. Dĺžka je počet premenných, ktorím sa má zrušiť subscribe. Ďalej nasledujú trojice bytov dĺžka krát. V prvých dvoch bytoch je uložené ID premennej, ktorej sa má zrušiť subscribe. V poslednom z týchto bytov je typ tejto premennej.

- RETURN PACKET

1. + 2. Byte	3. Byte	4. Byte	5. - N. Byte
Premenná	Typ	Dĺžka	Data

Prvé dva byty return packetu udávajú ID premennej. Tretí byte je typ premennej, ktorá sa posiela v tomto packete. Štvrtý byte je dĺžka. Dĺžka je nastavená vždy na 1, ak premenná, ktorá je v packete, nie je viac cieľov alebo prekážok. Ak je, dĺžka potom bude počet týchto cieľov alebo prekážok. Ostatné byty sú hodnota premennej. Veľkosť týchto

bytov je (sizeOf(Typ)*dĺžka).

- ERROR PACKET

1. + 2. Byte	3. Byte	4. Byte	5. - N. Byte
0xFFFF	0x06	1	Chybové hlásenie ako reťazec

Error packet sa vracia pri rôznych chybách. Prvé dva byty sú vždy 0xFFFF, tretí je vždy 0x06 (kód pre string), štvrtý byte je vždy 1 a zvyšné byty predstavujú chybové hlásenie.

Súbory

Software používa niekoľko súborov na konfiguráciu a nastavenia. Ich popis a vysvetlenie niektorých častí:

- AbstraktorVariablen.txt – je to konfiguračný textový súbor, v ktorom sú uložené tieto údaje o premenných:

ID	dvoj-bytové hexadecimálne číslo
Dátový typ	jedno-bytové číslo
Dĺžka	
Meno premennej	
Prednastavená hodnota	
Jednotka	String
Vizualizácia	(N – nie, O..prehľad, D - detailná)
Analysator	(N - nie, Y - ano)
Shell	(N - nie, Y - ano)
Typ logu	(L - log, E - udalosť, N - nie)
Len na čítanie	(R – len na čítanie, W- aj na zápis)
Textový popis premennej	

- AbstraktorVariablen.xls – v tomto súbore sú údaje, ktoré sú aj v AbstraktorVariablen.txt, len v prehľadnej, ľudsky čitateľnej forme.

- settings - súbor so začiatočnou konfiguráciou. Sú tu uložení informácie ako cesta k súboru s premennými, číslo portu, a premenné Abstractora. Tieto údaje sú uložené pri vypnutí.

- testing.201103071441.log – Pri každom spustení abstractoru sa vytvorí súbor s menom testing+dátum+čas spustenia . V tomto súbore sú uložené údaje z každej sekundy plavby (simulovanej aj skutočnej).

2.2.4 Riadenie plachetnice

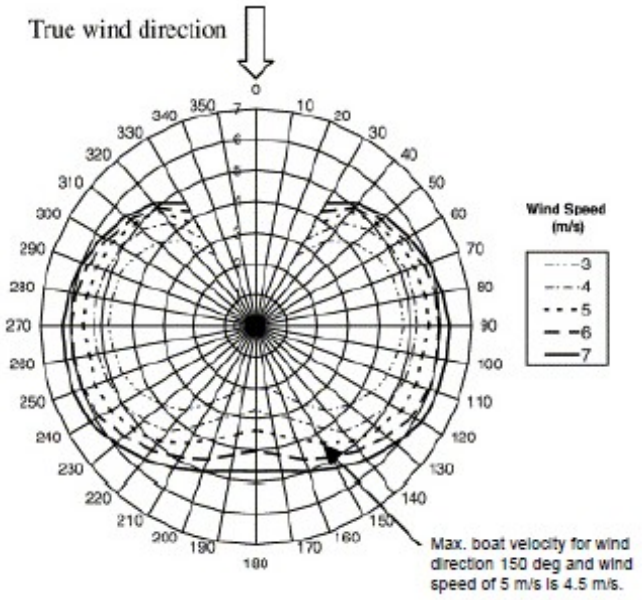
Riadiaci systém je rozdelený do štyroch vrstiev. Tieto vrstvy sú zodpovedné za:

- strategické dlhodobé smerovanie
- krátke nasmerovanie
- vykonávanie manévrov
- núdzové reflexy

Všetky štyri vrstvy sú vykonávané paralelne. Každá vrstva má prístup k údajom senzorových snímačov cez údaje abstraktora. Spodné vrstvy sú implementované v abstractore, vrchné v analysatore.

Polárny diagram

Skutočná rýchlosť plachetnice závisí od rýchlosti vetra, a na uhle medzi loďou a smerom vetra: kým oproti vetru nie je možné získať skoro žiadnu rýchlosť, maximálna rýchlosť je zvyčajne získaná z vetra zo zadnej strany +- 120 stupňov. Táto závislosť môže byť vykreslená ako špecifický polárny diagram lode.



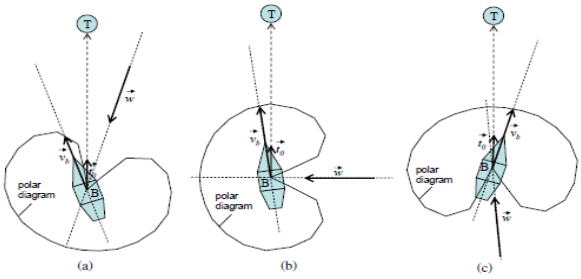
Obrázok 5. Príklad polárneho diagramu. [3]

„Rýchlosť lode v_b je teda určená rýchlosťou vetra w_{abs} a uhlom medzi smerom vetra a smerom lode:

$$|\vec{v}_b| = f\left(|\vec{w}_{abs}|, |\varphi(\vec{v}_b) - \varphi(\vec{w}_{abs})|\right)$$ “ [3]

Dosiahnutie stanoveného cieľa

Loď sa pri svojej plavbe snaží dosahovať vopred stanovené ciele. Pri určovaní smeru a plánovaní celej trasy k cieľu sa využíva algoritmus na smerovanie. Tento algoritmus určí najlepší smer, po ktorom sa má plachetnica vydať. Na to, aby sa loď dostala z aktuálnej pozície B do cieľového bodu T, sa musia brať do úvahy aj smer k cieľu aj smer vetra. „Cieľom smerovacieho algoritmu je znížiť vzdialenosť k cieľu tak rýchlo, ako je to možné.“ [3]



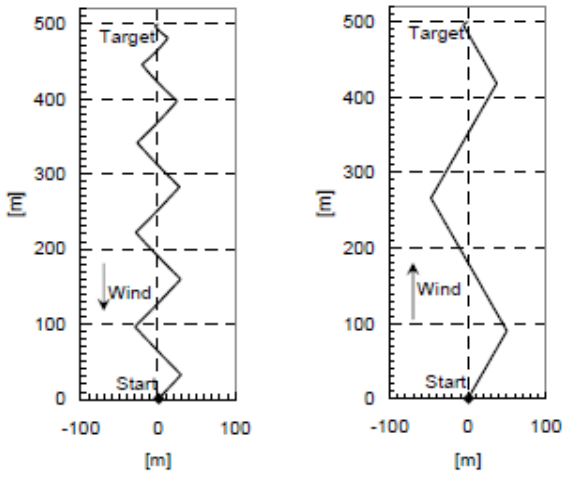
Obrázok 6. Možné konštelácie lode, cieľa a vetra. [3]

Dlhodobé smerovanie

Vrchná vrstva riadiaceho systému má na starosti hlavnú stratégiu smerovania plachetnice. Smerovanie lode môže byť považované za postup, kedy je optimálna trať určená pre konkrétne plavidlo, založené na predpovedi počasia a morských prúdov. Algoritmus smerovania určuje optimálnu hrubú trasu s ohľadom na špecifické správanie lode, predpokladané počasie a správanie mora. Trasa je rozdelená do mnohých krátkych úsekov. Ďalšie koordináty cieľa sú odovzdávané do nižšej vrstvy.

Krátkodobé smerovanie

Táto vrstva je pod vrstvou dlhodobého smerovania. Riadi loď k menším cieľom, určených dlhodobým smerovaním. Niektoré kurzy sú splavné, ale pomerne neefektívne. Aby bolo možné riadiť plachetnicu na konkrétny cieľ, musia byť vopred stanovené navigovateľné cesty. Tieto obmedzenia treba vziať do úvahy. Preto trasa môže obsahovať viac oddielov, ktoré sú prepojené rôznymi manévrami.



Obrázok 7. Príklady kľučkovania na 500 m, oproti vetru (vľavo) a po vetre (vpravo) založené na smerovacej metóde navrhnutej Stezlerom a Pröllom (2007). [4]

Vzhľadom k tomu, že poveternostné podmienky sa často menia a presné predpovede počasia nie sú k dispozícii, na účely určenia optimálneho smeru boli brané do úvahy len miestne poveternostné podmienky. Krátkodobé smerovanie vôbec nepoužíva predpoveď počasia. Reaguje na zmeny poveternostných podmienok v reálnom čase pravidelným prepočítaním optimálneho riadenia. Statické prekážky, ako sú ostrovy, rovnako ako

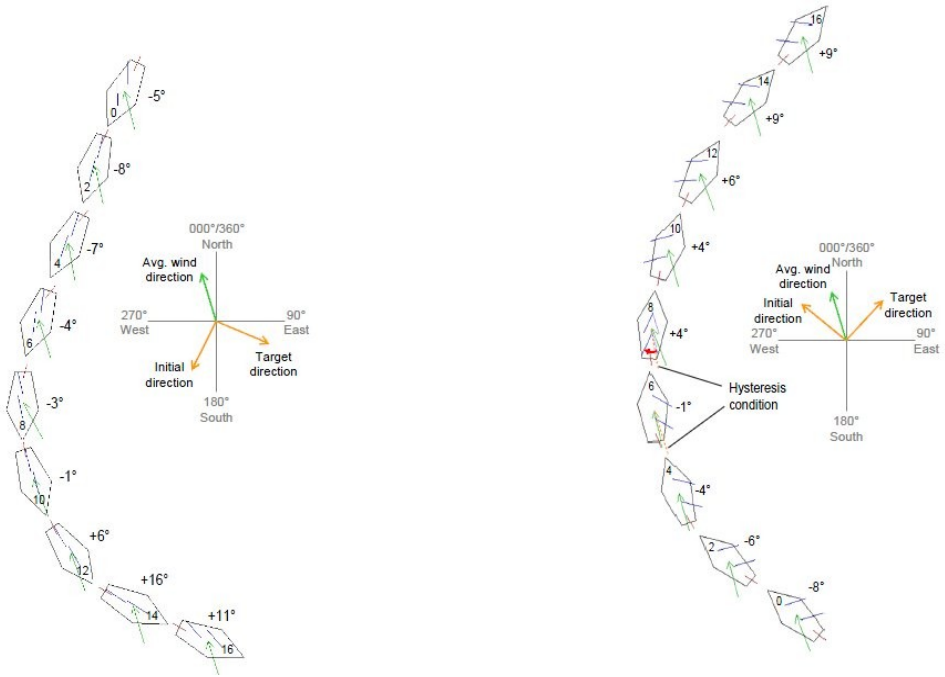
dynamické prekážky ako ostatné lode musia byť zohľadnené tu.

Hysterézia a parameter kľučkovania:

V praxi námorník kľučkuje okolo smeru k cieľu ak je cieľ vnútri uhla, kde žiadne priama navigácia nie je možná. Takýto princíp používa aj AVS Roboat. Nasleduje lokálne optimum v_b (v blízkosti nedávneho kurzu) po určitú dobu, kým globálne optimum v'_b nie je výrazne lepšie ako v_b. V tomto bode sa loď otočí podľa globálneho optima v'_b, ktoré bude nasledovať kým alternatívny kurz nebude výrazne lepší ako ďalšie otočenie a toto sa opakuje.

Vykonávanie manévrov

Táto vrstva je pod vrstvou krátkodobého smerovania. Ak sa skutočný smer lode odchyľuje od požadovaného smeru, ktorý určí krátkodobé smerovanie, systém upraví polohy kormidla tak, aby loď mala požadovaný kurz. Paralelne sa kontrolný systém snaží vytvoriť pohon využitím vetra fúkajúceho do plachiet. Aj zmena smeru vetra pri plachtení môže spôsobiť manéver.



Obrázok 8. Otáčanie: pozícia kormidla a plachty podľa smeru k cieľu, smeru vetra a počiatočného kurzu. [4]

Obrázok vľavo

0 začiatočný kurz

2 začiatočný kurz

4 Krátke smerovanie kurzu určí smer, kade má loď ísť a začína otáčať kormidlom.

6 Natočenie kormidla sa zvyšuje.

8 Vzhľadom k otočeniu lode sa sklon kormidla ďalej nezvyšuje, aj keď je loď stále ešte ďaleko od požadovaného smeru.

10 Loď je otočená priamo oproti vetru. Náklon klesne pod požadovanú hodnotu kvôli slabému bočnému vetru. Otočením plachty skúša dosiahnuť požadovaný náklon. Aby sa predišlo nadmernému pôsobeniu kormidla, je už znova v centrálnej polohe.

12 Vzhľadom k zotrvačnosti sa loď otáča požadovaným smerom.

14 Zvýši sa sila bočného vetra, čiže sa zvýši aj náklon.

16 Nový požadovaný smer je dosiahnutý. Plachta a kormidlo sa otočili, aby bol dosiahnutý požadovaný náklon.

Obrázok vpravo

0 začiatočný kurz

2 Krátke smerovanie kurzu určí smer, kade má loď ísť a začína otáčať kormidlom.

4 Natočenie kormidla sa zvyšuje.

6 Loď sa už obrátila pomocou vetra. Plachty sú stále na pravej strane, pretože hysterézia podmienka nie je zatiaľ splnená.

8 Teraz sa loď otočila výrazne cez smer vetra a hysterézna podmienka je splnená. Preto sa plachty otočili na druhú stranu.

10 Plachty sa viac natočili, lebo náklon je menší od požadovaného. Kormidlo znižuje sklon, pretože cieľový smer je takmer dosiahnutý.

12, 14 Vzhľadom k zotrvačnosti sa loď otáča požadovaným smerom.

16 Nový požadovaný smer je dosiahnutý. Plachta a kormidlo sa otočili, aby bol dosiahnutý požadovaný náklon.

Vyhýbanie sa prekážkam

Dôležitý problém je vyriešenie detekcie statických prekážok a vyhýbanie sa im. Statické prekážky môžu byť vopred preddefinované na mape ako základ pre nasmerovanie systému. Kombináciu viacerých techník, ako sú termálne zobrazovanie, radar, kamera a

automatický identifikačný systém je možné zistiť prítomnosť dynamických prekážok. Výskum v tejto oblasti bol vykonaný pre autonómne podvodné vozidlá. Vyhýbanie sa prekážkam je iné pre plachetnice, lebo sa musia pohybovať v smere závislom na poveternostných podmienkach.
Tento prístup implementovaný v kontrolnom systéme bol vyvíjaný v INNOC (2006). Algoritmus umožňuje autonómnej plachetnici úspešne oboplávať rôzne veľké prekážky za rôznych poveternostných podmienok.

Rozšírené určovanie trasy pre vyhýbanie sa prekážkam

Existuje celková maximálna vzdialenosť, ktorú nazývame "bezpečný horizont r_{max}". Segmenty za ňou môžu byť ignorované. Smery vedúce k prekážkam v bezpečnom horizonte dostanú trest. Tento trest sa zvyšuje čím bližšie sa prekážka nachádza. Veľmi blízke prekážky, ktorých vzdialenosť hrozí, že poklesnú pod určitú hodnotu r_{min}, označí zodpovedajúce smery ako také, ktorými nemožno ísť (nesplavné).

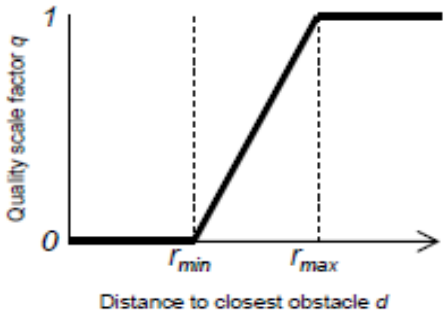
Rozlišujeme dva prípady:

- Nesplavné kurzy, ktoré nemožno navigovať kvôli smeru vetra, alebo preto, že prekážka je príliš blízko. Tieto kurzy nie sú zvažované pri výpočte optimálneho kurzu.

- Splavné kurzy. Vzhľadom na blízkosť prekážky (vzdialenosť $d_b < r_{max}$), sú kurzy dynamicky menené. „Trasy vedúce k prekážke dostanú trest q_b ako váhu k v_t. q_b sa vypočíta podľa rovnice:

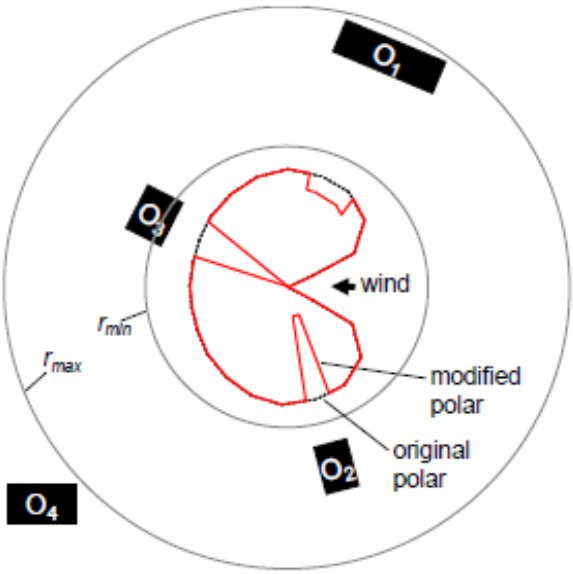
$$q_b = \min\left(1, \max\left(0, \frac{d_b - r_{min}}{r_{max} - r_{min}}\right)\right)$$ " [1]

  Je to vlastne lineárne škálovanie medzi 0 a 1 v rozsahu $r_{min} - r_{max}$.
  Mimo tohto rozsahu je to 0 alebo 1.



Obrázok 9. Lineárne škálovanie polárneho diagramu v závislosti na vzdialenosti prekážok. [1]

Vysoký trest znamená malé q_v (blízko 0), nízky znamená hodnotu blízku k 1.
Toto v_t umožňuje algoritmu optimalizácie kurzu porovnávať hodnoty rôznych kurzov, kde sú zahrnuté prekážky.



Obrázok 10. Vplyv prekážok na polárny diagram. [1]

Na obrázku 10. sú znázornené dynamické zmeny polárneho diagramu. Keďže do úvahy sú brané iba prekážky v bezpečnom horizonte r_{max}, O_4 sa ignoruje. Smer vektora k prekážke O_1 dostane iba malý trest, pretože sa nachádza blízko r_{max}. Smer k prekážke O_2 dostane oveľa väčší trest. Prekážka O_3 porušuje r_{min}, a preto smer k nej bude nesplavný.

2.3 Podobné riešenia

2.3.1 Pôvodný simulátor

Autori AVS Roboat vyvinuli simulátor pre plavbu plachetnice. Tento simulátor sa ale nespráva podľa reálnych fyzikálnych zákonov a loď v simulátore sa teda neplaví ako v skutočnom prostredí.

Analýza pôvodného simulátora

Simulátor je súčasť abstractora. Čas striedaní sa cyklov je nepriamo závislý na premennej SimTimeScaleFactor, ktorá je implicitne nastavená na 1 a čas medzi jednotlivými cyklami trvá sekundu. Pomocou SimTimeScaleFactor sa dá zrýchľovať / spomaľovať rýchlosť simulácie.

Na začiatku sa načítajú údaje z vybraného log súboru. Premenná SimBoat určuje, či brať údaje z tohto log súboru, alebo simulovať plavbu lode podľa prostredia. Údaje z log súboru sa tu používajú, len ak nie je nastavená táto premenná. V takom prípade sa priamo nastavujú cyklicky vlastnosti lode, ako pozícia x a y, rýchlosť, kurz, pozícia kormidla a smerovanie. Tieto vlastnosti sa potom simulujú. Umožňuje to pozerať si už uskutočnenú plavbu.

Ak je SimBoat nastavená, používa sa simulácia plavby. Podľa vzorcov a konštánt sa nastavuje rýchlosť. Nové súradnice lode vznikajú sčítaním starých súradníc, rýchlosti lode podľa kurzu a bočného unášania lode Kurz sa nastavuje priamo zo smerovania.

2.3.2 Portugalský simulátor

Keďže súťaž s robotickými plachetnicami prebieha na celom svete, súťažiaci sú z rôznych krajín. Naši kolegovia z INNOC s jedným tímom z Portugalska nadviazali kontakt, komunikujú a vymieňajú si ním informácie. Portugalský tím vlastní tiež svoju plachetnicu a aj simulátor. Poskytli nám jeden zdrojový súbor tohto simulátora, z ktorého môžme zistiť, na akom princípe pracuje. Žiaľ, nemôžme ho otestovať, ani nijak ohodnotiť, pretože máme len tento jeden zdrojový súbor a dva konfiguračné súbory.

súbor *sail.conf* – konfiguračný súbor, je tu uložená adresa IP, čísla portov, časy na rôzne účely, maximálna a minimálna rýchlosť lode a pod.

Súbor *sailforcetable.dat* – sily plachty v závislosti na smere vetra.

súbor *fast-sim.c* – zdrojový súbor so simulátorom. Simulátor používa ako vstupy iba skutočný smer a rýchlosť vetra, rýchlosť pohánania plachty a pozíciu kormidla (tiež aj začiatočnú pozíciu a smer lode) výstupy sú: zjavný smer a rýchlosť vetra (akyvietor sa zdá z pohľadu lode), latitude/longitude, kurz, rýchlosť, uhly smerovania

Hlavná simulácia beží každých 100ms. Používajú sa nejaké triky na aproximáciu na 100 ms. Dátové premenné sú typu float s menom s_*. Veľa parametrov sú fixné konštanty v kóde a boli nastavené porovnávaním správania simulovanej plachetnice a reálnej lode.

Na začiatku sa načítajú konfiguračné súbory. Vypočítajú sa nové súradnice pripočítaním rýchlosti k starým a násobením konštantami. Z týchto súradníc sa dostanú GPS súradnice a pridá sa k nim šum. Vypočíta sa zdanlivý vietor zo skutočného. Šum sa pridá aj kurzu, náklonom a vetru. Uhol medzi plachtou a zjavným smerom vetra určuje úplnú silu plachty. Vypočíta sa použitím vetra a vzorcov. Sila je rozložená na silu ťahania, ktorá ťahá dopredu loď s silu náklonu, ktorá ťahá loď doboku. Oba paralelne k vode.

Ako vidieť, v simulácií sa používa veľa vzorcov, fyziky, odpozorovaných vlastností a konštánt lode a pridávanie šumu veličinám. Nedá sa zhodnotiť ako dobre simulátor funguje, pretože sme nemali prístup k celému zdroju.

3 Špecifikácia

Celý software používa komunikáciu prostredníctvom socketov cez blackboard so špecifickým protokolom. Pre lepši pochopenie blackboardovej architektúry sme vytvorili malú aplikáciu, ktorá komunikuje s abstractorom, ktorý slúži ako blackboard, takýmto protokolom. Museli sme preto vytvoriť aj novú premennú, s ktorou budeme pracovať.

3.1 Popis vytvorenia novej premennej:

- Modifikácia súboru s premennými

  Novú premennú je potrebné pridať do súboru s premennými, ktorého cesta je uložená v súbore settings, ktorý používa abstractor. Tu sme premennej nastavili ID triedy, v ktorej sa bude nachádzať, ID premennej, typ, dĺžku, meno, prednastavenú hodnotu, jednotku, vizualizáciu, prístup z analysatora a zo shellu, logovanie, a možnosť prepisovať ju.
- Vytvorenie triedy s novým, ešte nepoužitým ID

  Každá, už definovaná trieda má svoje jedinečné ID.
- Vytvorenie premennej v triede

  V novovytvorenej triede sme vytvorili premennú, ktorej typ bol, aký sme uviedli do premennej v súbore s premennými. V triede sme vytvorili aj ID premennej, ako v tomto súbore.
- Inicializácia triedy

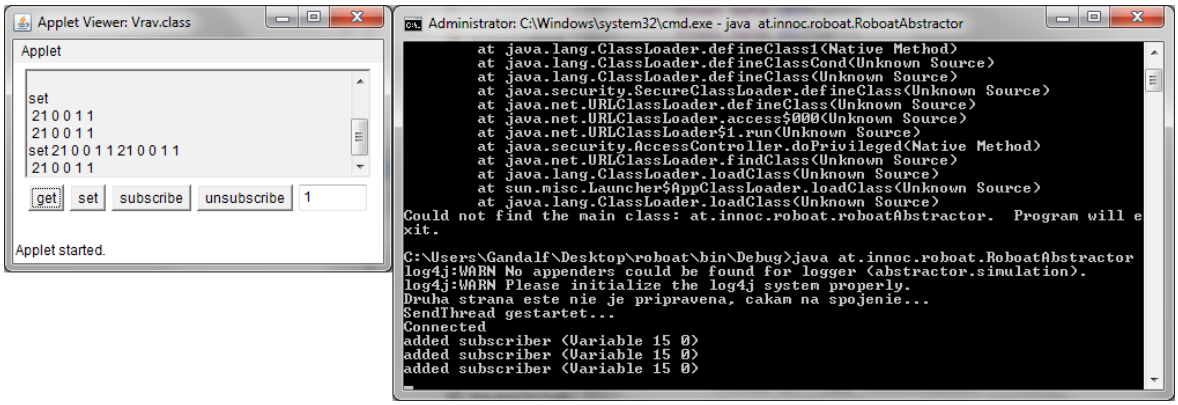
  Trieda sa inicializuje v konštruktore abstractora.
- Registrácia triedy

  Do triedy packet sme zapísali ID našej triedy.

Toto boli minimálne požiadavky na pridanie premennej, ktoré musia byť splnené.

3.1.1 Komunikácia s aplikáciou

Vytvorili sme najjednoduchšiu možnú aplikáciu pre komunikáciu s abstractorom, ktorá používa základné štyri operácie na prácu s blackboardom: get, set, subscribe, unsubscribe. Snažili sme sa, aby obsahovala len tie najdôležitejšie veci a bola čo možno najprehľadnejšia. Obsahuje štyri tlačidlá základné operácie a jedno textové pole pre zadávanie hodnoty čísla.



Obrázok 11. Aplikácia komunikujúca z abstractorom.

Aplikácia si posiela sockety s abstractorom podľa definovaného protokolu. Pri operácií set sa vezme číselná hodnota z textového poľa. Pre jednoduchosť sa nekontroluje správnosť výrazu. Inde sa textové pole nevyužíva. Ako vidieť na obrázku 11., aplikácia pracuje podľa popisu blackboardu.

Aplikácia čaká v nekonečnom cykle na nové správy. Akonáhle dostane novú správu, napíše ju do text area ako kladné celé čísla oddelené medzerami.

Port na pripojenie k abstractoru:

```
private static final int port = 2628;
```

Konštanty používané v programe:

```
final byte MOJA_TRIEDA=0x15;
final byte MOJA_PREM=0x00;
final byte GET=0x00;
final byte SET=0x01;
final byte SUBSCR=0x02;
final byte UNSUBSCR=0x03;
final byte TYPE=0x00;
```

Funkcia `posliSpravu(byte[] bts)` odosiela byty pomocou

```
wr.write(bts, 0, bts.length);
wr.flush();
```

Packety si vymieňa s abstractorom podľa popisu komunikačného protokolu.

- GET

Abstractoru pošleme packet, na žiadosť o zaslanie premennej.

Poslanie packetu:

```
byte[] a = {GET ,MOJA_TRIEDA, MOJA_PREM, TYPE,0x00};
posliSpravu(a);
```

Premenná je identifikovaná podľa bytov `MOJA_TRIEDA + MOJA_PREM`. Chceme aby nám bola zaslaná hodnota typu byte - `TYPE=0x00`. Index môže mať ľubovolnú hodnotu, pretože sa v tomto prípade bude ignorovať.

Reakcia:

Následne nám príde return packet z abstractora: 21 0 0 1 3, kde

$21_D=15_H$ – ID triedy

$0_D=0_H$ – ID premennej

$1_D=1_H$ – dĺžka v bytoch

$3_D=3_H$ – hodnota premennej

- SET

Packet na žiadosť o nastavenie hodnoty premennej.

Poslanie packetu:

```
byte[] a = {SET, MOJA_TRIEDA, MOJA_PREM, TYPE, 0x01,
            Byte.valueOf(hodnota.getText())};
posliSpravu(a);
```

Premenná je identifikovaná podľa bytov `MOJA_TRIEDA + MOJA_PREM`. Premennej chceme nastaviť hodnotu typu byte - `TYPE=0x00`. Dĺžka je 1. Hodnota, ktorá sa má nastaviť sa vezme z textového poľa.

Reakcia:

Ak premenná nebola odoberaná, naspäť sa nepošle return packet. Ak bola odoberaná, príde toľko return packetov, koľkokrát bola odoberaná z nášho socketu.

Príklad: Premennej sme dva krát nastavili odoberanie a potom sme poslali SET packet. Naspäť nám prišli packety 21 0 0 1 1 a 21 0 0 1 1 – dva krát:

$21_D=15_H$ – ID triedy

$0_D=0_H$ – ID premennej

$1_D=1_H$ – dĺžka v bytoch

$1_D=1_H$ – hodnota premennej

- SUBSCRIBE

Packet na žiadosť o odoberanie premennej.

Poslanie packetu:

```
byte[] a = {SUBSCR, 0x00, 0x01, MOJA_TRIEDA, MOJA_PREM, TYPE, 0x00};
posliSpravu(a);
```

Chceme nastaviť odoberanie jednej premennej, takže dĺžka bude 1 (`0x00*256 + 0x01 = 1`). Premenná je identifikovaná podľa bytov `MOJA_TRIEDA + MOJA_PREM`. Chceme aby sa nám zasielala hodnota premennej typu byte - `TYPE=0x00`. Rýchlosť aktualizácie nastavíme na 0 - chceme, aby sa nám premenná posielala pri jej menení.

Reakcia:

Následne príde return packet, ako po príkaze GET.

- UNSUBSCRIBE

Packet na žiadosť o zrušenie odoberania.

Poslanie packetu:

```
byte[] a = {UNSUBSCR, 0x00, 0x01, MOJA_TRIEDA, MOJA_PREM, TYPE};
posliSpravu(a);
```

Chceme zrušiť odoberanie jednej premennej, takže dĺžka bude 1 (`0x00*256 + 0x01 = 1`). Premenná je identifikovaná podľa bytov `MOJA_TRIEDA + MOJA_PREM`. Typ je byte.

Reakcia:

Rozlišujeme 3 prípady, ak bola premenná z nášho socketu odoberaná:

- práve raz - odoberaná už nebude.
- viac ako raz – bude odoberaná z nášho socketu o jeden krát menej
- nebola odoberaná – nezmení sa nič

V žiadnom prípade sa neposiela späť return packet.

3.2 Návrh riešenia

3.2.1 Analýza možností

Simuláciu plachetnice je možné urobiť viacerými spôsobmi.

- **Použitie fyzikálnych vzorcov**

  Ako u našich partnerov z Portugalska, simulácia môže byť založená na fyzikálnych vzorcoch. Pri takomto spôsobe by sme potrebovali vedieť viac o konštantách lode, ako plocha plachty, kormidla a pod. Navyše tento spôsob vyžaduje veľa znalostí fyziky, ktoré nemáme.

- **Použitie existujúcich údajov z reálnych plavieb**

  Plachetnica počas svojich plavieb ukladá informácie o aktuálnom stave prostredia a celej plachetnice. Tieto údaje zapisuje do log súboru. Jeden taký súbor z dlhej plavby máme k dispozícií.

Kvôli slabším znalostiam fyziky, neznámym konštantám lode a prístupným údajom z plavieb sme si vybrali možnosť naprogramovať simulátor pomocou existujúcich logov z plavieb. Simulátor bude využívať neurónovú sieť, ktorá je už naprogramovaná v openCV v jazyku C++. Na trénovanie sa používa back propagation algoritmus.

Použitie back propagation algoritmu

Neurónová sieť pozostáva z viacerých vrstiev: vstupná vrstva, výstupná vrstva a niekoľko skrytých vrstiev. Každá vrstva pozostáva z neurónov, ktoré dostanú vstup z neurónov z vrstvy priamo pod ňou a pošlú ich výstup neurónom vo vrstve priamo nad ňou. Každý neurón z vrstvy N je prepojený z každým neurónom vrstvy N+1. Medzi neurónmi v

jednej vrstve nie sú žiadne prepojenia. Zo vstupov sa šíria údaje cez všetky vrstvy až po výstupnú. Tam sa porovná so želanými hodnotami. Obvykle je vo výstupnej vrstve chyba v každom z neurónov. Algoritmus sa snaží priviesť tieto chyby na nulu. Back propagation algoritmus môžeme rozdeliť na dve časti, ktoré sa opakujú, kým chyba nie je dostatočne malá:

1. Šírenie

   Dopredné šírenie trénovacieho vstupu cez neurónovú sieť cieľom vytvoriť šírenie aktivačného výstupu.

   Spätné šírenie propagácie aktivačného výstupu cez neurónovú sieť. Pomocou trénovacieho cieľa sa vytvára delta pre všetky výstupné a skryté neuróny.
2. Zmena váhy

   Každej váhe sa vyráta sklon vynásobením výstupnej delty a aktivačného vstupu.

   Sklonu váhy udáva, kde chyba rastie.

Spojenie neurónovej siete a simulátora

Keďže simulátor je písaný v jazyku java a neurónová sieť v C++, aplikácie potrebujú medzi sebou nejakým spôsobom komunikovať. Spôsobov komunikácie je viacero:

- presmerovanie vstupu a výstupu cez pipe
- komunikácia cez sockety
- Java Native Interface (jni) – interface, ktorý dovoľuje volať natívne metódy z Java aplikácie.

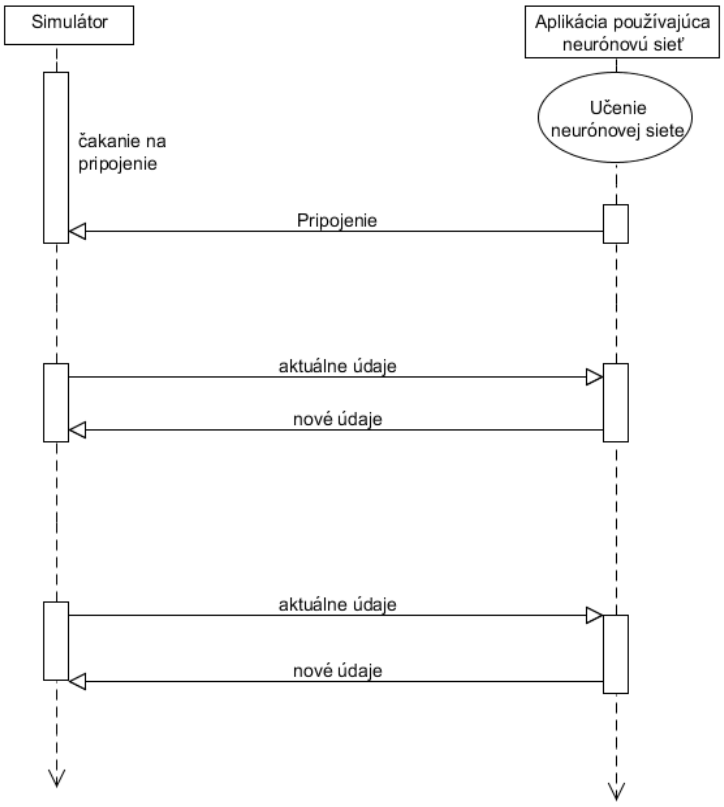
Z týchto možností sme si vybrali komunikáciu cez sockety.

Dôvody:

- aplikácie môžu bežať na rôznych PC
- s komunikáciou cez sockety máme skúsenosti
- použiteľné na ľubovoľnej platforme

Aplikácia používajúca neurónovú sieť sa najprv natrénuje z údajov zapísaných v súbore. Simulátor čaká, kým sa aplikácia nepripojí. Aplikácia sa pripojí, a keď sa dá

simulátoru povel na chod, začne sa medzi nimi komunikácia. Simulátor posiela svoje staré údaje neurónovej sieti, a tá následne pošle simulátoru nové, aktuálne údaje.

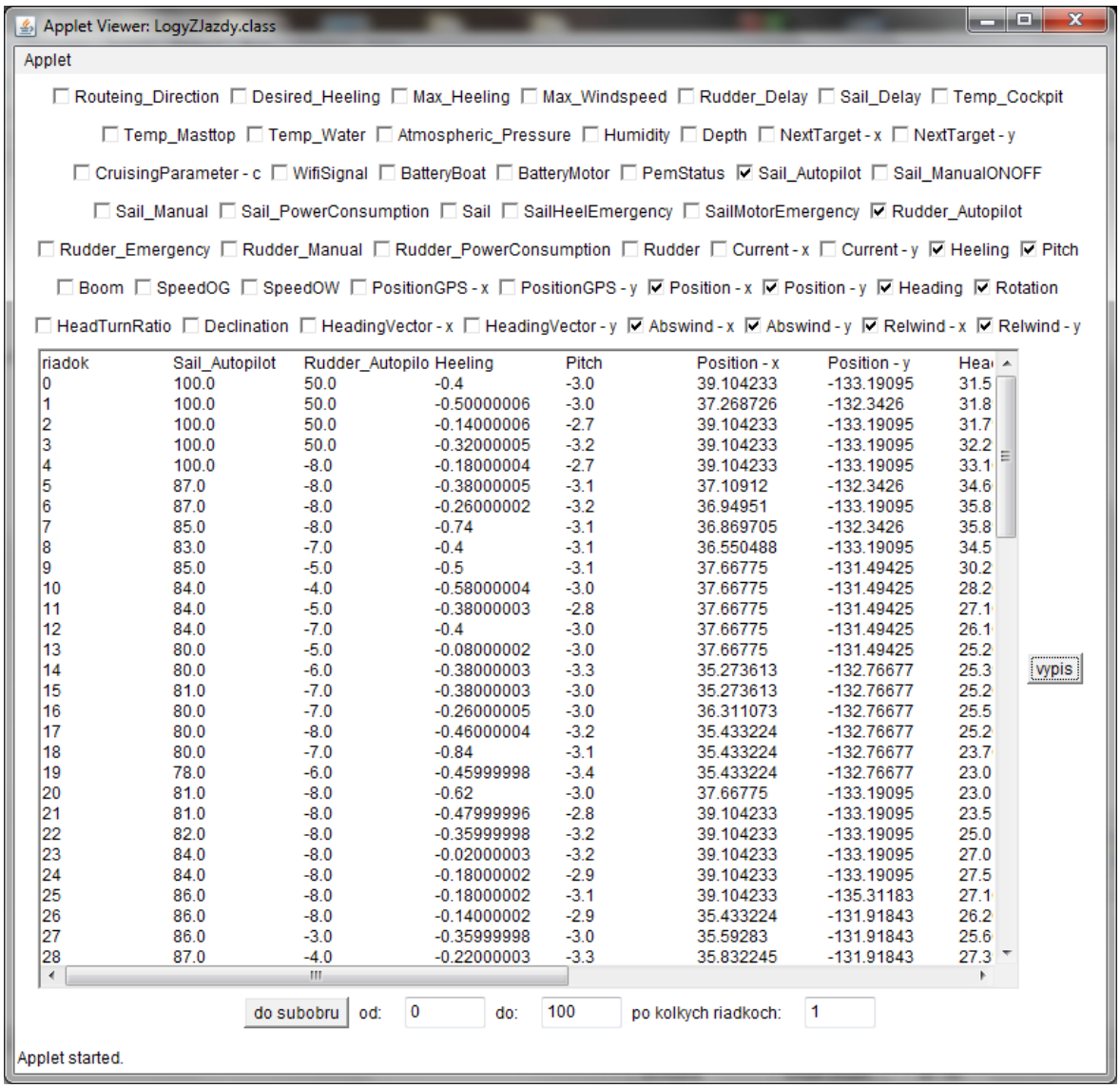


Obrázok 12. Prepojenie simulátora s neurónovou sieťou.

3.3 Implementácia

3.3.1 Spracovanie log súboru

V log súbore sú uložené pre človeka ťažko čitateľne údaje. Vytvorili sme preto aplikáciu, s rôznymi nastaveniami, ktorá zobrazí prehľadne tieto hodnoty. Pomocou tejto aplikácie môžeme zistiť, ktoré hodnoty sú pre nás dôležité, a ktoré nepotrebné.



Obrázok 13. Aplikácia na spracovanie log súboru.

Zaškrtnutím chceck boxov si vyberáme údaje, ktoré chceme mať zobrazené. Celý log súbor je dosť dlhý a zobrazenie všetkých logov by trvalo dlho, preto väčšinou nezobrazujeme všetky logy, ale môžeme si vybrať poradové číslo začiatočného a konečného logu pre určenie intervalu, ktorý chceme zobraziť. Je možné určiť aj číslo, po koľkých logoch sa nám budú zobrazovať údaje. Aplikácia umožňuje informácie ukladať priamo do súboru.

3.3.2 Trénovanie neurónovej siete

Princíp riadenia

V našej aplikácií na spracovanie log súboru sme zistili, ktoré údaje sa v log súbore nachádzajú a ktoré sú vhodné na určovanie potrebných vlastností lode. Vhodné údaje boli ako vstupné riadiace údaje:

- natočenie plachty
- pozícia kormidla

merané vstupné údaje:

- náklon do strany
- náklon dozadu
- rýchlosť
- kurz
- otočenie
- absolútny vietor x,y
- relatívny vietor x,y

výstupné údaje:

- náklon do strany
- náklon dozadu
- rýchlosť
- kurz
- otočenie

Súradnice pozície lode boli priveľmi nepresné a zašumené, tak sme s nimi nemohli pracovať.

Na trénovanie neurónovej siete sa používa súbor vygenerovaný aplikáciou na spracovanie log súboru. Ako vstup berie neurónová sieť dva logy idúce za sebou (22 čísiel) a ako výstup vracia päť horeuvedených hodnôt. Pri vstupoch neurónovej siete upravujeme kurz. Hodnota kurzu je 0 – 360. Môže nastať situácia, keď sa loď bude otáčať cez os, a zo vstupu, kde je veľké číslo, bude výstup s malým číslom. Aby nenastávali veľké skoky, upravujeme túto hodnotu tak, aby boli čísla bližšie k sebe. To znamená, že ak má nastať takýto skok, k malému číslu prirátame 360, alebo z veľkého odrátame 360. Z toho vyplýva,

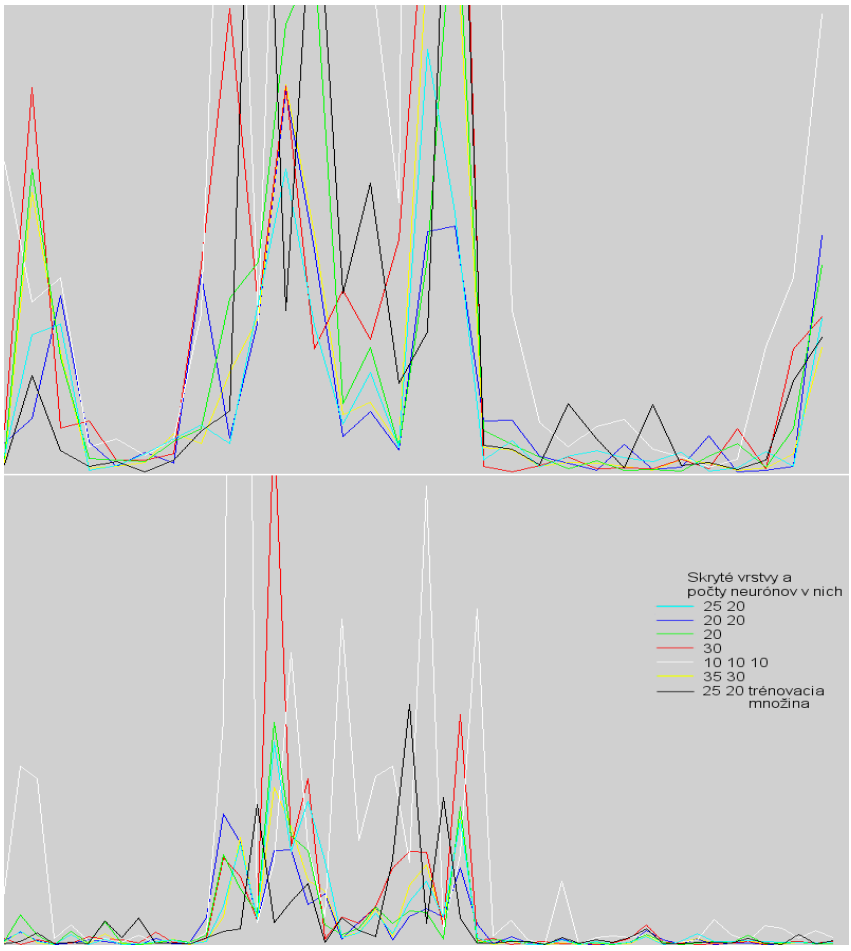
že hodnoty presiahnu hranice 0 – 360, ale odstránime nežiadúce skoky. Hodnoty mimo hraníc potom upravujeme, kde je potrebné.

Trénovanie siete

Pre dosiahnutie najlepšieho natrénovania neurónovej siete sme skúšali rôzny počet skrytých vrstiev a neurónov. Z logov sme si vyrobili množinu na trénovanie a množinu na testovanie neurónovej siete.

Obrázok 14. Znázornenie trénovacej a testovacej množiny pre neurónovú sieť.

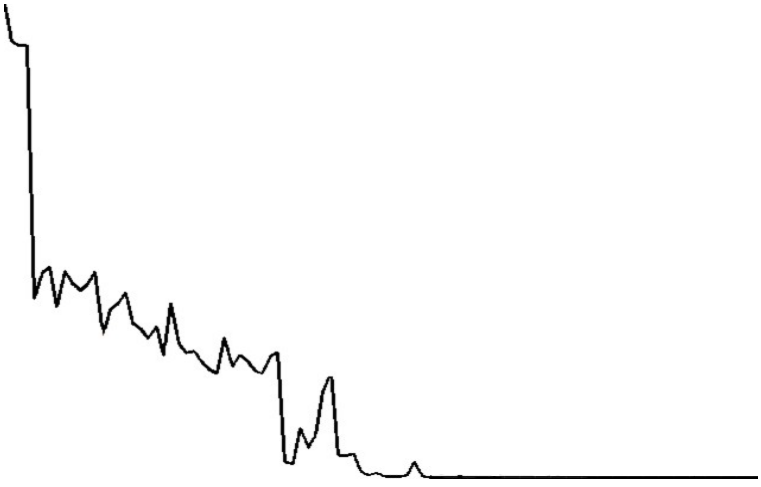
Pri testovaní sme potom správnosť rátali kvadratickou chybou. Podľa týchto chýb sme si vykreslovali grafy, zisťovali priemernú chybu a tak zisťovali ako dobre je sieť natrénovaná.



Obrázok 15. Meranie všetkých chýb na testovacej množine.

Z týchto grafov vidieť, že najväčšie chyby z týchto majú sieť s tromi vrstvami po 10 neurónov a s jednou vrstvou s 30 neurónmi. Ostatné siete boli približne rovnaké, snažili sme sa vybrať rôzne rozpoloženie vrstiev, s čo najlepšími výsledkami.

Ďalej sme trénovali sieť, kým nieje kvadratická chyba dostatočne malá.



Obrázok 16. Meranie priemerných chýb testovacej množiny po každom novom učení siete.

Trénovaciu množinu sme skúšali rozdeľovať na rôzne veľké úseky, ktoré sa má neurónová sieť učiť. Výsledky ukázali, že najmenšia chyba nastáva, keď sa neurónová sieť naučí všetky údaje naraz.

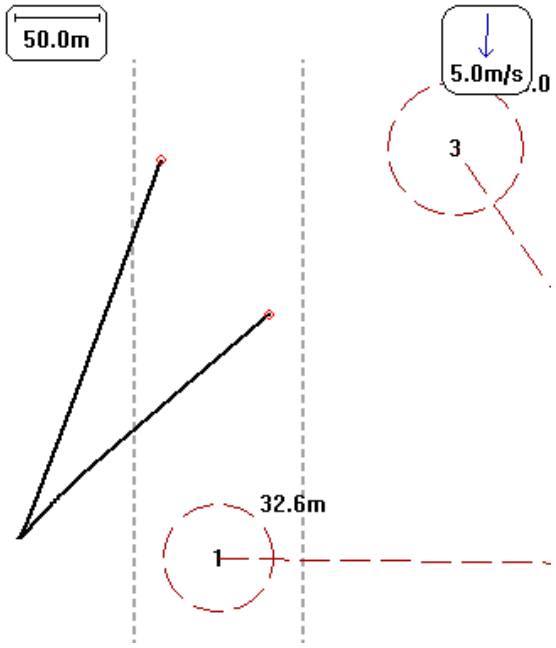
Integrácia neurónovej siete do stabilného riešenia

Simulátor sme implementovali v novom unite, ktorý je dedičom pôvodného simulátora a spúšťa sa namiesto neho. Logovanie a vyrátanie pozície plachetnice pomocou kurzu a rýchlosti zostali zo starého simulátora. Simulátor posiela cez sockety údaje o lodi neurónovej sieti. Keďže sieť je natrénovaná z dvoch po sebe idúcich logov, aj tu potrebuje ešte staršie údaje lode, než aktuálne, ktoré si simulátor pamätá. Po poslaní správy čaká simulátor na spätnú správu od aplikácie s neurónovou sieťou. Po jej prijatí sa nastavia nové hodnoty.

4 Výsledky

4.1 Úpravy kvôli log súboru

K dispozícii sme mali len jeden log súbor, z približne 10 hodinovej plavby. Veľa údajov tu bolo zašumených. Pri testovaní sme odpozorovali, že niektoré hodnoty v simulátore majú iný rozsah, ako v log súbore. Napríklad hodnota pozície kormidla je v aplikácií ohraničená <-15, 15>, zatiaľčo v log súbore <-100, 100>. Túto nezhodu sme riešili vynásobením konštantou.

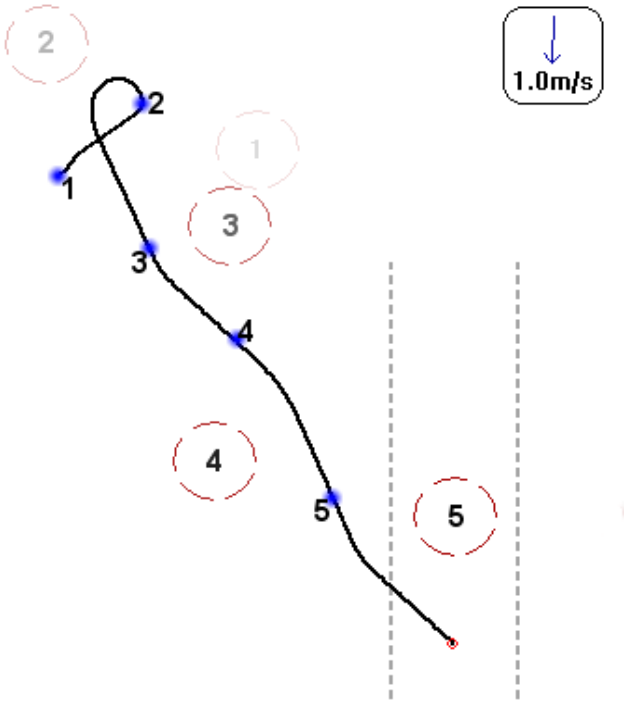


Obrázok 17. Úprava pozície kormidla.

Loď na obrázku 17. by mala ísť smerom k cieľu 1. Loď, ktorá ide na obrázku viac doprava, má upravované údaje o kormidle, ktoré sa posielajú neurónovej sieti. Vidieť, že reaguje trochu viac na kormidlo, ale aj tak nejde požadovaným smerom (priamo na cieľ 1).

4.2 Pozorovanie reakcie na zmenu hodnôt lode

Ak sa pozícia cieľa zmení, plachetnica natočí plachtu a kormidlo potrebnými smermi, a smeruje k novým súradniciam cieľa. Pri takomto presúvaní môžeme zistiť ako simulátor reaguje na zmenu pozície plachty a kormidla.

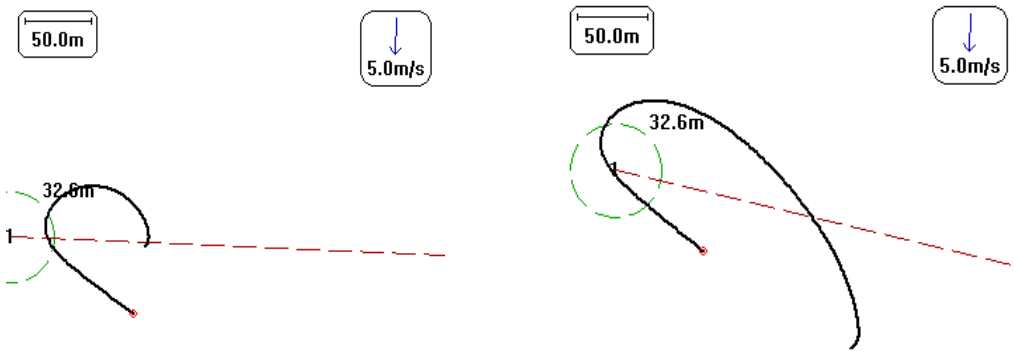


Obrázok 18. Presúvanie cieľa počas plavby.

Čísla pri modrých bodoch označujú, kedy sa pozícia daného cieľa zmenila. Od bodu 1, loď ide približne k prvému cieľu, ale ide priveľmi nahor. Od bodu 2 sa zmení smer plavby, smerom k druhému cieľu, ale odvráti sa od neho. Pri bodoch 3, 4, 5 plachetnica reaguje na zmenu pozície cieľa miernym zmenením smeru svojho kurzu.

4.3 Dosiahnutie cieľa

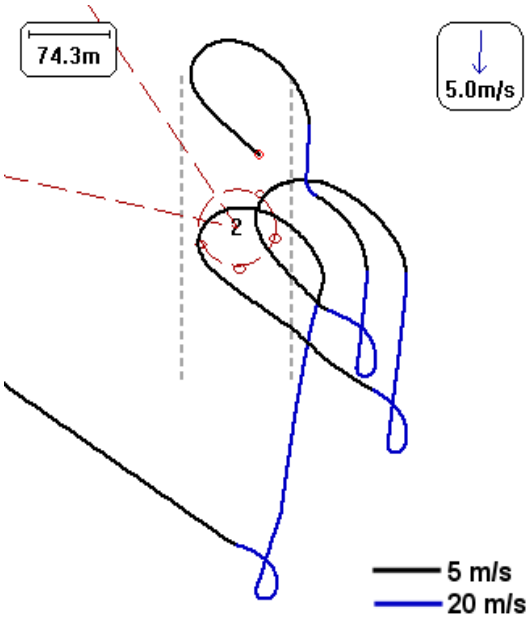
V niektorých prípadoch loď príde do cieľa. Umiestnili sme cieľ západne od pozície lode, ktorá bola otočená na východ. Loď sa počas plavby otáčala k cieľu, a aj do neho došla



Obrázok 19. Dosiahnutie cieľa.

4.4 Vplyv sily vetra

Plachetnica menila smer plavby podľa toho, ako silno fúkal vietor.



Obrázok 20. Vplyv sily vetra na simulovanú plavbu plachetnice.

Kým pri vetre s rýchlosťou 5 m/s loď smerovala juhovýchodne, pri 20 m/s smerovala severne. Požadovaný smer, ani pozície cieľov sa nemenili.

5 Záver

Výsledkom bakalárskej práce je funkčný simulátor robotickej plachetnice. Mal by uľahčiť prácu pri tvorbe algoritmov našim partnerom zo spoločnosti INNOC, ktorí túto plachetnicu vlastnia. S týmto simulátorom by nemuseli chodiť do terénu zakaždým, keď chcú otestovať nové správanie plachetnice.

Simulátor zo stavu prostredia a plachetnice simuluje správanie plachetnice. Zdá sa, že simulácia plavby nie je úplne reálna. Loď v často krát pláva smerom približne k cieľu, ale stáva sa, že ho potom minie a naberie kurz, ktorý si udrží, a už sa nezmení. Potom sa loď už len vzďaľuje od cieľa.

K dispozícii sme mali len jeden log súbor zo skutočnej plavby na mori. V tomto súbore boli navyše dosť zašumené niektoré hodnoty. Niektoré údaje, ako maximálna pozícia kormidla sa nezhodovali s existujúcou aplikáciou. Počas plavby, keď sa vytváral tento súbor, nemusel fúkať vietor všetkými možnými smermi, ktoré mohli nastať v simulácií. Platí to aj pre rýchlosť vetra, kurz, smerovanie, natočenie plachiet, pozíciu kormidla, a aj pre ich kombinácie. Napríklad keď nenastala situácia, kedy by loď plávala proti vetru, a v simulátore má nastať takáto situácia, simulátor to bude pravdepodobne dosť nepresne simulovať.

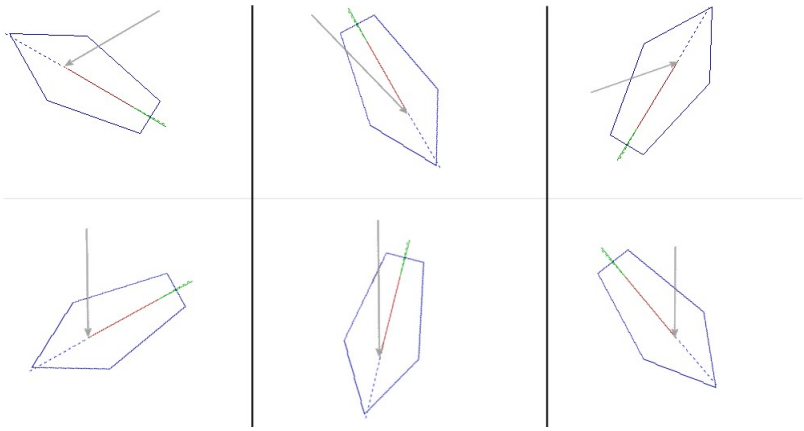
Cieľ bakalárskej práce sme splnili. Simulátor reaguje na vlastnosti lode a prostredia. Smer lode sa viditeľne mení pri zmene natočenia plachty a pozície kormidla. Loď sa veľakrát aj dostane do cieľa, a ak nie, zvykne sa k nemu približovať.

5.1 Možné vylepšenia

Možné vylepšenia vidíme vo vytvorení nových logov zo skutočných plavieb na mori. Presnejšie údaje môžu veľmi pomôcť pri simulovaní plachetnice. Neurónová sieť sa môže potom naučiť aj viac údajov, a simulácia môže byť presnejšia.

Keďže je ťažké zariadiť, aby počas skutočnej plavby na mori smer vetra dosiahol každú možnú hodnotu, ďalším vylepšením by mohlo byť upravenie vlastností lode

vzhľadom na vietor. To znamená, že pri učení by sme súradnicovú sústavu otáčali tak, aby mal vietor vždy rovnaký smer. Pri simulácií by sa potom dodatočne upravovali hodnoty späť.



Obrázok 21. Otočenie súradnicovej sústavy za účelom dosiahnutia rovnakého smeru vetra.

Prílohy

Príloha č.1: CD so zdrojovými kódmi alpikácie a elektronickou verziou bakalárskej práce

Zoznam použitej literatúry

[1] Stelzer, R. – Jafarmadar, K. – Hassler, H. – Charwot, R. 2009, *A Reactive Approach to Obstacle Avoidance in Autonomous Sailing,* Proceedings of International Robotic Sailing Conference, Kingston, Ontario, Canada http://www.cci.dmu.ac.uk/administrator/components/com_jresearch/assets/publications/1274961145.pdf

[2] Stelzer, R. – Jafarmadar, K. 2009 *Communication Architecture for Autonomous Sailboats,* Proceedings of International Robotic Sailing Conference, Matosinhos, Portugal http://www.cci.dmu.ac.uk/administrator/components/com_jresearch/assets/publications/1266686522.pdf

[3] Stelzer, R. – Pröll T. *2008 Autonomous Sailboat Navigation for Short Course Racing,* Elsevier Journal of Robotics and Autonomous Systems, Vol. 56 (7) http://www.cci.dmu.ac.uk/administrator/components/com_jresearch/assets/publications/1266686445.pdf

[4] Stelzer, R. – Pröll T. - John R. I. *2007 Fuzzy Logic Control System for Autonomous Sailboat,* Proceedings of IEEE International Conference on Fuzzy Systems, London, UK http://www.cci.dmu.ac.uk/administrator/components/com_jresearch/assets/publications/1266686696.pdf

[5]Stelzer, R. – Jafarmadar, K. *2007 A Layered System Architecture to Control an Autonomous Sailboat,* Proceedings of TAROS 2007, Aberystwyth, UK http://www.cci.dmu.ac.uk/administrator/components/com_jresearch/assets/publications/1266686769.pdf

[6] Artificial neural networks, 2010, http://www.learnartificialneuralnetworks.com/backpropagation.html

[7] Dr. Hunt. J. *2002 Blackboard Architecture,* http://www.agent.ai/doc/upload/200402/hunt02_1.pdf

[8] Metzner Ch. - Cortez L. - Chacín D., 2005, *Using a Blackboard Architecture in a Web Application,* http://2005papers.iisit.org/I58f73Metz.pdf