Helsingin yliopisto
Tietojenkäsittelytieteen laitos

Cum laude laboratorio
Ohjelmistotuotantoprojekti Nobel

TESTAUSDOKUMENTTI

Ilkka Forsblom, Risto-Matti Kääriäinen, Jussi Lahti,
Maarit Saaresto & Jarkko Salminen

versio 1.0

Ohjaaja:	**Teemu Kurppa**
Tilaaja:	**Tapio Elomaa**
jätetty:	3.5.2001

Sisältö

1 Luokkien rakenteellinen ja toiminnallinen testaus

1.1 Itseään testaavat oliot

Ohjelmavirheiden löytämiseksi olioluokista tehdään itseään testaavia. Tämä tarkoittaa olioiden funktioissa tehtäviä automaattisia tarkistuksia, jotka testaavat olion tilan oikeellisuutta attribuuttien perusteella. Jokaiseen luokkaan koodataan invariant-funktio, jonka tehtävänä on palauttaa totuusarvo olion tilan eheydestä. Esimerkiksi taulukko-olion invariant-funktio tarkistaisi, ettei taulukon koko ole negatiivinen. Kaikkien mutkikkaampien funktioiden, joiden laskennassa saattaa olla virheitä, pitää lopuksi tarkistaa invariantin voimassaolo. Myös muiden laskennan oikeellisuutta testaavien jälkiehtojen käyttö on järkevää. Jos funktiolle voidaan määritellä järkevä esiehto, se on hyvä varmistaa. Yleisesti algoritmien eri askeleisiin voidaan sisällyttää tarkistuksia.

Näiden rakenteellisten tarkistuksien toteutuksessa käytetään esikääntäjän makroja. Seuraava header määrittelee makrot invarianttien ja muiden ehtojen testaamiseen.

SelfTestingObject.h:

```
//estetään moninkertainen includointi
#ifndef SELFTESTINGOBJECT_H
#define SELFTESTINGOBJECT_H

//käytetään assert-makroja
#include <cassert>
#include <string>
#include <stdexcept>
#include <cstdio>

namespace NobelNS {

//käyttäjä perii abstraktin SelfTestingObject-luokan
class SelfTestingObject {
public:
//käyttäjä toteuttaa abstraktin rajapinnan
        virtual bool invariant() const = 0;
        virtual ~SelfTestingObject() {}
};

class InvariantException : public std::runtime_error {
public:
        InvariantException(std::string s, int
line) : std::runtime_error(s+InvariantException::toString(line))
{}
        virtual ~InvariantException() {}
private:
        static std::string toString(int i) {
                char buf[11];
                snprintf(buf, 11, "%d", i);
                return std::string(buf);
```

```
        }
};

}

//kun NDEBUG on määritelty, testausvaiheen assertit korvataan poikkeuksilla
#define NDEBUG

//kun OPTIMIZATION on määritelty, kaikki tarkistukset poistetaan
//käytetään vain, jos on pakko optimoida
/*#define OPTIMIZATION*/

//jos debuggaus on päällä käytetään assertteja
#ifndef NDEBUG
        #define INV() assert(this->invariant())
        #define TEST(boolean, exception) assert(boolean)
        #define INVDB() assert(this->invariant())
        #define TESTDB(boolean) assert(boolean)
//muuten heitetään poikkeuksia
#else
//jos ei optimoida
        #ifndef OPTIMIZATION
                #define INV() if (!this->invariant()) throw
NobelNS::InvariantException(std::string()+"Class in
variant failure! File: "+__FILE__+" line: ", __LINE__)
                #define TEST(boolean, exception) if (!(boolean)) throw (exception)
                #define INVDB()
                #define TESTDB(boolean)
//jos optimoidaan, tarkistukset poistetaan
        #else
                #define INV()
                #define TEST(boolean, exception)
                #define INVDB()
                #define TESTDB(boolean)
        #endif
#endif

/*Makroja INV() ja TEST(boolean, exception) käytetään esim. jälki-
ja esiehtoina. Niiden käyttö on suositeltavaa.
INVDB() ja TESTDB(boolean) sopivat aikaa vaativiin tarkistuksiin.*/

#endif
```

Käyttäjäluokka perii abstraktin `SelfTestingObject`–luokan ja toteuttaa `invariant`-funktion. Tämän jälkeen `INV()`–makroja käytetään funktioissa jälkiehtoina testaamaan olion invarianttia. `TEST(boolean, exception)`–makroa voi käyttää esiehtojen, jälkiehtojen ja muiden laskennan vaiheiden tarkistukseen. Kun `NDEBUG`–direktiiviä ei ole määritelty, esikääntäjä generoi makroista `assert`–kutsuja. Jos assertin parametri on tosi, ohjelman suoritus jatkuu normaalisti. Jos se on epätosi, tulostetaan jäljitettävä virheilmoitus `standarderror`–virtaan ja ohjelman suoritus terminoidaan. Kun `NDEBUG` on määritelty, asserttien sijasta koodiin generoidaan poikkeuslauseita. Testausvaiheessa käytetään `assert`–makroja, ja ylläpitovaiheessa heitetään poikkeuksia. Tätä muutosta varten pelkkä `NDEBUG`–direktiivin määrittely `SelfTestingObject`–

headerissa riittää.

Kommentointien poistaminen `OPTIMIZATION` makrosta aiheuttaa kaikkien tarkistusten hylkäämisen, mikä ei kuitenkaan ole suotavaa, sillä ohjelmistoon on tarkistus jättää poikkeuksia heittäviä ehtolauseita. `OPTIMIZAION`–makroa voi käyttää, jos tarkistuksista kertyy liikaa yleisrasitetta. `INVDB()`– ja `TESTDB(boolean)`–makroilla voi erikseen määritellä aikaa vaativat testit, joita käytetään vain testausvaiheessa. Erillisen testaus-headerin ansiosta ehtolauseita ei tarvitse poistaa eikä muokata manuaalisesti.

Idea makrojen käytössä on, että testausvaiheessa virheen löytyessä assert-kutsu terminoi heti ohjelman suorituksen. Ylläpitovaiheessa tarkistukset voivat edelleen löytää virheitä, mutta tällöin on järkevää heittää poikkeus eikä kaataa ohjelmaa. Järjestelmä voidaan ohjelmoida varautumaan näihin poikkeuksiin, ilmoittamaan niistä ja mahdollisesti toipumaan niistä. Tällöin järjestelmän luotettavuus kasvaa. Makrojen avulla virheitä löytyy helpommin, ja ne voidaan paikallistaa kätevästi myös ylläpitovaiheessa. Paljon pahempaa on, jos järjestelmä jatkaa toimintaa epäeheässä tilassa tai kaatuu yllättäen eikä tapahtuneesta virheestä saada mitään tietoa.

1.2 Testiajurit

Luokille tehdään testiajurit, jotka suorittavat rakenteelliset peruspolkutestaukset. Testit käyvät läpi kaikki olioluokkien koodirivit. Testiaineistot koodataan suoraan ajuriin, mikä on teknisesti helpoin ja käytännöllisin ratkaisu. Toiminnallisuutta testataan antamalla jokaiselle funktiolle perussyötteet, hieman virheelliset ja täysin virheelliset syötteet. Virheiden löytämiseksi on syytä keskittyä erikoisiin syötteisiin. Funktioiden pitää palauttaa syötteitä vastaavat oikeat tulosteet. Ajurit käyttävät assert-makroja tulosten vertailuun. Ne voivat halutessaan testata olioiden tilan oikeellisuutta kutsumalla `invariant`–funktiota, vaikka oliot tarkastavat tilansa automaattisesti.

1.3 Nobel

Kaikki Nobelin luokat ovat varsin yksinkertaisia toiminnallisuudeltaan, eikä useimpia niistä ole sen takia ole testattu ollenkaan, ja testatuista luokistakin on käyty läpi lähinnä vain joidenkin metodien tietyt erikoistapaukset. Täysin testaamattomia ovat `BlockException`, `Length`, `Pose`, `RobotAcceleration` ja `RobotVelocity`.

1.3.1 Angle

Tapaus Luodaan `Angle` konstruktorin kutsulla `Angle(-180, Deg)`.

Testaa Normalisoinnin toimivuus.

Tulos Luotu kulma on suuruudeltaan 180 astetta.

1.3.2 Vector

Tapaus Luodaan Vectoreita komponenteilla (5,5), (-5,-5) ja (5,-5) (pituudet senttimetrejä). Kutsutaan metodeita `getLength()` ja `getAngle()`.

Testaa Metodien `getLength()` ja `getAngle()` oikeellisuutta eri perustapauksissa.

Tulos `getLength()` palauttaa kaikissa tapauksissa saman pituuden. `getAngle()` palauttaa 45 astetta, -125 astetta ja -45 astetta, vastaavasti.

Tapaus Luodaan Vector komponentteinaan (-5,0) (senttimetreissä). Kutsutaan `getAngle()`–metodia.

Testaa `getAngle()`–metodin toimivuus erikoistapauksessa.

Tulos Palautettu kulma on 180 astetta.

Tapaus Luodaan nollavektori (0,0), kutsutaan `getAngle()`–metodia

Testaa `getAngle()`–metodin toimivuus erikoistapauksessa.

Tulos Palautettu kulma on 0 astetta.

1.4 Input

1.4.1 Input

Luokan Input testauksessa on käyetty testiajuria `Input/testdriver4.cc`. Syötteet `Input`–oliolle on uudelleenohjattu tiedostosta.

Tapaus Ei syötteitä.

Testaa Luku stdinistä ei saa blokata prosessia, vaikka syötteitä ei ole annettu.

Tulos Onnistui.

Tapaus Syötteenä yksi 'q'-merkki.

Testaa funktion `isHaltSet()` pitää palauttaa true.

Tulos Onnistui.

Tapaus Syötteenä yksi 't'-merkki.

Testaa funktion `isHaltSet()` pitää palauttaa false.

Tulos Onnistui.

Tapaus Syötteenä merkkijono "xxxxt".

Testaa funktion `isHaltSet()` pitää palauttaa false.

Tulos Onnistui.

Tapaus Syötteenä merkkijono "qxxxxx".

Testaa funktion `isHaltSet()` pitää palauttaa true.

Tulos Onnistui.

Tapaus Syötteenä merkkijono “xxqxxx”.

Testaa funktion `isHaltSet()` pitää palauttaa true.

Tulos Onnistui.

Tapaus Syötteenä merkkijono “xxxxxq”.

Testaa funktion `isHaltSet()` pitää palauttaa true.

Tulos Onnistui.

Tapaus Syötteenä merkkijono “xxqqqxxx”.

Testaa funktion `isHaltSet()` pitää palauttaa true.

Tulos Onnistui.

Tapaus Syötteenä merkkijono, jossa on ensin yli sata ’x’-merkkiä ja lopussa ’q’.

Testaa Funktion isHaltSet pitää palauttaa ensimmäisellä kutsulla false, koska syöte luetaan sadan merkin kokoiseen puskuriin eikä ‘q’-merkki mahdu mukaan.

Tulos Onnistui.

1.5 Map

1.5.1 GridCell

GridCell-luokalle on tehty peruspolkutestaus (`Map/testdriver1.cc`).

1.5.2 Grid ja GridIterator

`Grid`– ja `GridIterator`–luokkia on testattu ajurilla `Map/testdriver2.cc` ja parilla testikartalla. Luokille ei ole tehty täydellistä peruspolkutestausta.

1.5.3 Map

`Map`–luokkaa on testattu ajurilla `driver3.cc` ja muutamalla testikartalla. Mapille ei ole tehty täydellistä peruspolkutestausta.

1.6 Movement

Movementille ei ole tehty modulitestausta, jossa luokkia olisi testattu yksinään. Movement–osajärjestelmää on testattu yhtenä kokonaisuutena robotilla ajetuilla käytännön testeillä, joissa osajärjestelmän on todettu toimivan oikein, eli robotti on käyttäytynyt oletetulla tavalla testitilanteissa.

1.7 RobotControl

1.7.1 RobotControl

Luokalle RobotControl ei ole tehty erillistä modulitestausta. Luokkaa on testattu integrointivaiheen testeillä.

1.8 Localization

1.8.1 Localization

Testiti tehtiin testiajurin ja kahden tyngän avulla. Tyngät korvasivat `Landmarks`– ja `LandmarkRecognition`–luokkien ilmentymät. Erilaiset testitilanteet simuloitiin muokkaamalla tynkien funktioiden paluuarvoja.

Tapaus 3 maamerkkiä eskenään 120 kulmissa, jokainen 100 cm etäisyydellä robotista.

Testaa triangulaatioalgoritmia.

Tulos Paikannukseski saatiin kolmion keskipiste ja suunnaksi setettu poikkeama x-akselista.

Tapaus 3 maamerkkiä kulmissa 0, 179 ja -90, jokainen 100 cm etäisyydellä robotista.

Testaa triangulaatioalgoritmia.

Tulos Paikannukseski saatiin kolmion keskipiste ja suunnaksi setettu poikkeama x-akselista.

Tapaus 3 maamerkkiä kulmissa 0, 160 ja -90, joista ensimmäinen 10000 cm etäisyydellä kaksi 100 cm etäisyydellä robotista.

Testaa triangulaatioalgoritmia.

Tulos Paikannukseksi saatiin kolmion keskipiste ja suunnaksi asetettu poikkeama x-akselista.

Tapaus 3 maamerkkiä kulmissa 0, 60 ja 120 , jokainen 100 cm etäisyydellä robotista.

Testaa Maamerkkien valinta-algoritmi: testattavien tutkittavien kolmikoiden valintaa.

Tulos LocalizationException: “Landmark picking failure.”

Tapaus 25 maamerkkiä ‘A’–‘Z’ kattaen tasaisesti 360 astetta, jokainen 100 cm etäisyydellä robtista.

Testaa Maamerkkien valinta-algoritmi: parhaan kolmikon valintaa testatuista kolmikoista.

Tulos Paikannukseksi saatiin maamerkkien muodotaman ympytän keskipiste ja suunnaksi asetettu poikkeama x-akselista.

Tapaus Havaittiin alle 3 maamerkkiä.

Testaa Maamerkkien valinta-algoritmi: aloitusehto.

Tulos LocalizationException: “Not enough landmarks.”

Tapaus Yhtään maamerkkiä ei havaittu.

Testaa Maamerkkien valinta-algoritmi: aloitusehto.

Tulos LocalizationException: “Not enough landmarks left for triangulation.”

Tapaus Havaittiin 5 maamerkkiä, joista 2 samoja.

Testaa Moninkertaisten ja siten mahdollisesti väärien maamerkkien poisto tutkittavien maamerkkien joukosta.

Tulos Paikannus saatiin oikeellisesti kolmesta toisistaan eroavasta maamerkistä.

Tapaus 5 maamerkkiä, joista 3 samoja.

Testaa Ajon keskeytys, mikäli dublikaattien poiston jälkeen jäljellä liian vähän maamekkejä paikan määritystä varten.

Tulos LocalizationException: “Not enough landmarks left for triangulation.”

Tapaus Nähdään maamerkit ‘A’, ‘B’ ja ‘C’. Sijainnit tiedetään maamerkeille ‘B’, ‘C’ ja ‘D’.

Testaa Testaa tilannetta, jossa maamerkkitiedoston maamerkiltä puuttuu sijaintitiedot syystä taikka toisesta. **Huomaa:** Tämän tilanteen ei tulisi esiintyä valmiissa ojelmistossa.

Tulos LocalizationException: “Unplaced landmark (A) detected: Terminating triangulation.”

1.8.2 Landmarks

Luokan Landmarks testauksessa on käytetty testiajuria `Localization/TestLandmarks.cc`.

Tapaus Perustesti: palauttaako Landmarks maamerkin sijainnin?

Testaa funktion getLandmarkLocation toiminta

Tulos Testi onnistui: funktio palautti merkin sijainnin

Tapaus Syntaksivirhe syötetiedostossa: yhden merkin y-koordinaatti puuttuu

Testaa konstruktorin toiminta

Tulos Testi onnistui: konstruktori hyväksyy syötteen syntaksivirheeseen asti ja hylkää loput.

Tapaus Syntaksivirhe syötetiedostossa: yhden merkin y-koordinaatti onkin merkki eikä double.

Testaa konstruktorin toiminta

Tulos Testi onnistui: konstruktori hyväksyy syötteen syntaksivirheeseen asti ja hylkää loput.

Tapaus Syntaksivirhe syötetiedostossa: yhden merkin y-koordinaatin keskelle merkitään välilyönti.

Testaa konstruktorin toiminta

Tulos Testi onnistui: konstruktori hyväksyy syötteen syntaksivirheeseen asti ja hylkää loput.

1.8.3 LandmarkRecognition

LandmarkRecognition-luokan toiminta on varsin yksinkertaista. Se tunnistaa halutut segmentit annetuilla maamerkkejä vastaavilla hahmontunnistuskoneilla. Testaus liittyy ainoastaan hahmontunnistuskeoneiden käyttöön. Lisäksi testataan luokan ominaisuus yhdistää saman maamerkin kahdentumisen kahdessa vierekkäisessä limittäisessä kuvassa yhdistäminen.

Tapaus Maamerkkitiedostoon merkitään yhden koneen tiedostonimi väärin (`letter_U.svm` → `leter_U.svm`).

Testaa Testaa konstruktoria, jonka pitää tarkistaa löytyykö kaikki annetut koneet levyltä.

Tulos Oikea tulos eli LandmarksFileNotFoundException.

Tapaus Määritellään n kpl hahmontunnistuskoneita.

Testaa Testaa recognize()-metodia, jonka pitää käydä tunnistuksessa kaikki n kpl koneita läpi

Tulos Kaikki n kpl koneita käytiin tunnistuksessa läpi.

Tapaus Asetetaan tunnistettava maamerkki kuvien rajakohtaan, jolloin maamerkki näkyy kahdessa eri kuvassa.

Testaa Kahden maamerkin yhdistämistä ja niiden kulmien keskiarvon laskemista uudelle yhdistetylle maamerkille.

Tulos Tulos halutun mukainen.

1.8.4 Segmentation

`Segmentation`–luokka on varsin yksinkertainen toiminnallisuudeltaan, eikä sille siksi ole suoritettu modulitestausta.

1.8.5 BasicSegmentation

Basicsegmentationin testaaminen muista olioista riippumatta olisi ollut hyvin työlästä, joten sille ei ole suoritettu modulitestausta.

1.8.6 View

View-luokka ainoastaan ottaa kahdeksan kuvaa.

Tapaus Ajetaan yksi paikanmääritys.

Testaa Kaikki kahdeksan kuvaa otetaan onnistuneesti.

Tulos Kaikki kahdeksan kuvaa otettu.

Tapaus Ajetaan kolme paikanmääritystä samalla ohjelman ajolla.

Testaa Kameran kääntymien optimointi so. ensin kuvataan vasemmalta oikealle, sitten oikealta vasemmalle ja lopuksi jälleen vasemmalta oikealle.

Tulos Ok

Tapaus Ajetaan yksi paikanmääritys.

Testaa Toimiiko kuvien tallennus levylle kun ajotiedostossa on `WRITE_YUV` –parametri on y.

Tulos Kuvat tallennettu.

Tapaus Ajetaan yksi paikanmääritys

Testaa Kuvia ei tallenneta kun `WRITE_YUV` –parametri on n.

Tulos Ei tallennetu

1.9 External

1.9.1 External

1.9.2 Settings

Tapaus Jätetään ajotiedostoon asetusrivin loppuun ylimääräisiä välilyöntejä.

Testaa Settingsin pitäisi karsia välilyönnit pois asetusten arvoista.

Tulos Onnistui.

1.9.3 Output

Tapaus Asetetaan ajotiedostoon lokitulostusparametrit nollaksi

Testaa Lokitiedostoon tulostetaan kaikki eri osajärjestelmien tulostukset

Tulos Kaikki tulostettu.

Tapaus Asetetaan ajotiedostoon ruudulle tulostus -parametrit nollaksi

Testaa Ruudulle tulostetaan kaikki eri osajärjestelmien tulostukset

Tulos Kaikki tulostettu.

Tapaus Asetetaan ajotiedostoon lokitulostusparametrit numeroksi viisi.

Testaa Lokitiedostoon ei tulosteta mitään.

Tulos Ei tulostusta.

Tapaus Asetetaan ajotiedostoon ruudulle tulostus -parametrit numeroksi viisi.

Testaa Ruudulle ei tulosteta mitään.

Tulos Ei tulostusta

2 Integrointitestaus

2.1 Osajärjestelmien testaus

2.1.1 Paikanmääritysosajärjestelmän testit

Testit tehdään testiajurilla `loc_ajuri.cc`, joka käyttää ainoastaan paikanmääritysosajärjestelmää.

Tapaus Testi 1

Testaa Maamerkkitiedostoon merkitään yhden koneen tiedostonimi väärin (`letter_U.svm` → `leter_U.svm`). Ohjelman pitää päättyä virheilmoitukseen, jossa sanotaan, että kyseistä tiedostoa ei löydy.

Tulos Ohjelma päättyy vasta panoraman kuvauksen jälkeen.

Tapaus Testi 2

Testaa Maamerkkitiedostoon merkitään vain 2 merkkiä sijainteineen. Ohjelman pitää päättyä virheilmoitukseen, jossa selitetään, että paikanmääritykseen vaaditaan vähintään kaksi maamerkkiä.

Tulos Ohjelman suoritus päättyy vasta, kun satunnaisvektorit on suoritettu.

Tapaus Testi 3

Testaa Maamerkkitiedostoon ei merkitä yhtään mitään. Järjestelmän pitää ilmoittaa, että maamerkit puuttuvat.

Tulos Ohjelman suoritus päättyy vasta, kun satunnaisvektorit on suoritettu.

Tapaus Testi 4

Testaa Robotin ympärillä on kolme eri maamerkkiä. Robotti on merkkien muodostaman kolmion sisällä. Merkit ovat yhden metrin päässä robotista 120:n asteen välein. Testi on onnistunut, jos robotti pystyy määrittämään suuntansa ja sijaintinsa.

Tulos Onnistui.

Tapaus Testi 5

Testaa Kuten tapaus 4, mutta kahdella samalla maamerkillä. Testi on onnistunut: ohjelma ilmoittaa, että maamerkkejä ei ole tarpeeksi.

Tulos Onnistui.

Tapaus Testi 6

Testaa Kuten tapaus 4, mutta vain kahdella maamerkillä. Ohjelman pitää ilmoittaa, että merkkejä ei ole tarpeeksi.

Tulos Onnistui.

Tapaus Testi 7

Testaa Kuten tapaus 4, mutta kolmas maamerkki on huomattavasti kauenpana robotista kuin muut merkit. Kolmas merkki asetetaan etäisyyksille 2, 3, 4 ja 5 m. Testi on onnistunut, jos kolmas merkki pystytään tunnistamaan vähintään kolmen metrin etäisyydeltä.

Tulos Testi onnistui. Merkki pystyttiin tunnistamaan vielä 6,3:n metrin päästä.

Tapaus Testi 8

Testaa osaako ohjelma valita useista merkeistä sopivan kolmikon siten, että robotti on merkkien muodostaman kolmion sisällä? Muuten kuten tapaus 4, mutta viidellä maamerkillä.

Tulos Onnistui.

Tapaus Testi 9

Testaa Osaako ohjelma karsia duplikaattimerkit pois, koska jokainen merkki saa esiintyä ympäristössä vain kerran. Kun sama merkki on havaittu kaksi kertaa samassa panoramassa, järjestelmä ei voi tietää kumpi merkki pitää hyväksyä. Dublikaattien havaitseminen johtuu tukivektorikoneiden satunnaisesti tunnistamista roskasegmenteistä, jotka eivät ole todellisia maamerkkejä. Muuten kuten tapaus 8, mutta kahdella samalla maamerkillä. Paikanmäärityksen pitäisi onnistua, koska merkkejä jää vielä kolme jäljelle.

Tulos Onnistui.

Tapaus Testi 10

Testaa Kuten testi 8, mutta kolmella samalla maamerkillä. Järjestelmän pitää ilmoittaa, että havaittuja maamerkkejä ei ole tarpeeksi.

Tulos Onnistui.

Tapaus Testi 11

Testaa havaitseeko järjestelmä, että robotti ei ole maamerkkikolmikon muodostaman kolmion sisäpuolella.

Kuten testi 4, mutta:

merkkien m_1 ja m_2 välinen kulma on 60 astetta

merkkien m_2 ja m_3 välinen kulma on 60 astetta

Testi on onnistunut, jos järjestelmä ilmoittaa, että paikanmääritys ei onnistu.

Tulos Onnistui.

2.1.2 Liikkumisosajärjestelmän testit

Testit suoritetaan kokonaan integroidulla ohjelmistolla. Testeissä mainittuja esteitä ei merkitä karttaan.

Varotoimenpiteet pysähtymistesteissä:

- valmiina painamaan `CONTROL-C` -näppäimiä!
- valmiina siirtämään estettä!

Tapaus Testi 1

Testaa Ajotiedostossa määritellään pysäytysetäisyys jokaiselle sonarille erikseen. On epäselvää, lasketaanko etäisyys robotin keskipisteestä vai reunasta vai jostain muualta.

- **viiden** etumaisen sonarin arvoksi asetetaan 40
- Asetetaan esteeksi reitille pahvilaatikko
- este on suoraan edessä, laatikon reuna kohtisuorassa kulkusuuntaan nähden
- Mille etäisyydelle laatikosta robotti pysähtyy? (jarrutusmatka vaikuttaa?)
- Testi on onnistunut, jos robotti ei törmää esteeseen ja merkintätapa saadaan selville. Voiko kameran johto osua esteeseen?

Tulos Testi onnistui. Etäisyys mitataan robotin reunasta. Pienin järkevä arvo etumaisille sonareille on 20 cm. Muuten tapahtuu törmäys. Jos robotti liikkuu etuperin, ja kameran suunta on taaksepäin, kameran johto voi osua esteeseen, kun pysähtymisetäisyys on 20 cm.

Tapaus Testi 2

Testaa Kuten testi 1, mutta laatikko on suoraan edessä ja laatikon reuna on vinottain (45 astetta) kulkusuuntaan nähden.

- pysäytysetäisyys reunasta laskien 20 cm
- Testi on onnistunut, jos esteeseen ei törmätä.

Tulos Testi onnistui.

Tapaus Testi 3

Testaa Kuten testi 2, mutta laatikko peittää reitin vain osittain, että ainoastaan robotin reuna on osumassa esteeseen. Laatikon reuna on kohtisuorassa reittiin nähden.

- pysäytysetäisyys reunasta laskien 20 cm
- Testi on onnistunut, jos esteeseen ei osuta.

Tulos Testi onnistui. Väliä esteeseen jäi 5 cm.

Tapaus Testi 4

Testaa Kuten testi 3, mutta laatikko sijoitetaan reitin ulkopuolelle 10 cm:n päähän.

- pysäytysetäisyys reunasta laskien 20 cm
- Testi on onnistunut, jos robotti ei ohita estettä vaarallisen läheltä - 10 cm:n etäisyydeltä, kun pysäytysetäisyyden pitäisi olla 20 cm.

Tulos Testi onnistui.

Tapaus Testi 5

Testaa Tomivatko vastapuolen sonarit, kun robotti peruuttaa?

- Robotin suunta pitää lähtöpisteessä asettaa siten, että reitti suoritetaan peruuttaen.
- Testi on onnistunut, jos vastapuolen sonarit toimivat peruutettaessa.

Tulos Testi onnistui.

Tapaus Testi 6

Testaa Toimiiko hätäpysäytys?

Kesken liikkeen painetaan `CONTROL-C`

Testi on onnistunut, jos näppäinkomennon jälkeen robotti liikku alle 10 cm

Tulos Testi onnistui.

Tapaus Testi 7

Testaa Havaitsevatko sonarit kapeat esteet? Väärässä paikassa olevaan tuoliin ei saa törmätä.

- Testikarttana voi käyttää `D427.asc`

- pysäytysetäisyydeksi robotin reunasta lukien asetetaan 20 cm (alaraja)
- Robotin reitille asetetaan tuoli: martti kulkee kohti tuolin jalkaa.
- Testi on onnistunut, jos tuolin jalkaan ei törmätä

Tulos Testi onnistui, kun tuolin jalka oli suoraan edessä tai hieman sivussa.

Tapaus Testi 8

Testaa Kuten testi 7, mutta esteenä on ihmisjalka!

Tulos Robotti törmää, jos jalkaterä on kohti robottia. Muuten robotti ehtii pysähtyä.

Tapaus Testi 9

Testaa Pystyykö robotti kulkemaan robottilabran oviaukosta?

- pysäytysetäisyys robotin reunasta lukien 20 cm
- käytetään **seitsemää** etumaista sonaria
- marginaali 15 cm
- kartta: `D427uusin.asc`
- lähtöpiste labrassa metrin päässä oviaukosta
- maali käytävällä
- asetetaan maamerkit oviaukon lähelle, että paikanmääritys onnistuu lähtöpisteessä
- Testi on onnistunut, jos robotti kulkee oviaukon läpi ilman sonarihavaintoja

Tulos Testiä ei suoritettu.

Tapaus Testi 10

Testaa Kuten testi 9, mutta käytetään **viittä** etumaista sonaria.

Tulos Testi onnistui. Viisi sonaria riittää esteiden havaitsemiseksi.

Tapaus Testi 11

Testaa Voiko pysäytysetäisyyden asettaa niin pieneksi, että robotti törmää esteisiin? Pitää varautua käyttäjän virheellisiin syötteisiin.

- asetetaan `SONAR_RANGE`–arvot nolliksi
- Testi on onnistunut, jos pysäytysetäisyys asetetaan oletusarvoisesti vähintään kymmeneen senttiin eikä esteisiin törmätä.

Tulos Testi epäonnistui. Nolla-arvot hyväsytään ja robotti törmää esteisiin.

Tapaus Testi 12

Testaa Kuten testi 11, mutta asetetaan negatiivisia arvoja

Tulos Ohjelma hyväksyy negatiiviset arvot ja robotti törmää esteisiin.

Tapaus Testi 13

Testaa Kuten testi 2, mutta laatikon kulma osoittaa kohti robottia.

Tulos Testi onnistui: väliä esteeseen jäi 2 cm.

2.1.3 Reitinlaskentaosajärjestelmän testit

Testit suoritetaan kokonaan integroidulla järjestelmällä robottilaboratoriossa. Testit käsittelevät ominaisuuksia, joita ensimmäisessä integrointivaiheessa ei ole kokeiltu.

Tapaus Testi 1

Testaa Syötteenä annetaan karttatiedosto, jossa on esteiden täydellisesti ympäröimä paikka. Kun robotti sijoitetaan tähän umpikujaan, ohjelman pitäisi paikanmäärityksen jälkeen ilmoittaa, ettei reittiä maaliin ole olemassa ja tämän jälkeen lopettaa suoritus.

Tulos Testi onnistui.

Tapaus Testi 2

Testaa Kuten testi 1, mutta esteiden välissä on pieni rako, josta robotti voisi mahtua. Marginaali asetetaan kuitenkin niin suureksi, että se peittää reitin. Jälleen ohjelman pitäisi ilmoittaa, että maalia ei voi saavuttaa.

Tulos Testi onnistui.

Tapaus Testi 3

Testaa Löytääkö robotti ulos marginaalialueelta?

- Testikartta (martissa): `~/help/koodit/D427.asc`
- Marginaaliksi esetetaan 100 cm
- Robotti sijoitetaan marginaalin sisälle
- Maamerkit on aseteltava siten, että paikanmääritys onnistuu marginaalialueella, ilman `randomWalkkia`.
- maalipiste pitää olla saavutettavissa
- sonarit eivät saa havaita seinää.
- Testi onnistunut, jos robotti lähtee suunnistamaan ulos marginaalista kohti maalipistettä. Robotin pitää liikkua optimaalista reittiä.

Tulos Testi onnistui.

Tapaus Testi 4

Testaa Robotin pitää löytää ulos marginaalista, kun sonarit ovat havainneet karttaan merkityn esteen. Käytännössä tämä voi johtua suuresta odometriavirheestä.

- pysäytysetäisyys robotin reunasta laskien 30 cm

- marginaali 100 cm
- maamerkit on asetettava siten, että paikanmääritys onnistuu esteen vieressä ensimmäisellä yrityksellä.
- Kun robotti on havainnut karttaesteen, sen pitäisi paikanmäärityksen jälkeen liikkua ulos marginaalista lyhintä reittiä (70 cm). Tämä korjausvektori ei noudata optimaalista reittiä maaliin.

Tulos Testi onnistui: robotti liikkui ensin ulos marginaalista ja lähti tämän jälkeen liikkumaan kohti maalia.

Tapaus Testi 5

Testaa Kun kartassa on yli 10 000 vapaata ruutua, käyttäjälle annetaan arvio dijkstran vaatimasta laskenta-ajasta. Onko aika-arvio oikea? Onko raja 10 000 liian pieni tai liian suuri?

- kartassa on 12 000 vapaata ruutua: ascii-tiedoston riville 400 laitetaan 30 merkiä pitkä rivi
- Testi on onnistunut, jos ohjelman käynnistyshetkestä paikanmäärityksen aloitukseen kuluu ilmoitettu aika.

Tulos Testi onnistui. Aika-arvio oli riittävän tarkka: ohjelma tulosti aika-arvioksi 1 s, ja aikaa kului 2-3 s.

Tapaus Testi 6

Testaa Kuten testi 5, mutta vapaita ruutuja on 50 000

Tulos Testi onnistui. Tulostus: 28 s. Mitattu aika: 29 s.

Tapaus Testi 7

Testaa Kuten testi 5, mutta ruutuja on 5000 00 tai enemmän. Testi on onnistunut, jos aika-arvio pitää paikkansa ja tulosteessa aika on annettu minuutteina ja sekunteina.

Tulos Aika-arvio oli riittävän tarkka. Tulostus: 2 min 40 s. Mitattu aika 2 min 5 s.

2.1.4 Ohjausosajärjestelmän testit

Testit tehdään kokonaan integroidulla järjestelmällä.

Tapaus Testi 1

Testaa Miten robotti käyttäytyy, kun sonarit havaitsevat yllättävän esteen?

- Testikartta `D427.asc`
- Vapaaehtoinen koehenkilö seisoo paikallaan robotin reitillä, kunnes ohjelma terminoi.
- Merkit asetellaan siten, että paikanmääritys onnistuu paikassa, johon robotti pysähtyi.

- pysäytysetäisyys robotin reunasta 10 cm, jonka pitäisi olla pienin mahdollinen arvo
- Testi on onnistunut, jos
  - Robotti pysähtyy törmäämättä esteeseen,
  - odottaa esteen siirtymistä parametrin WAIT verran,
  - yrittää kerran liikkua uudelleen,
  - tekee onnistuneesti paikanmäärityksen,
  - ilmoittaa, että yllättävä este on havaittu,
  - yrittää liikkua kerran uudelleen ja
  - tulostaa järkevän virheilmoituksen ja lopettaa ohjelman.

Tulos Testi onnistui.

Tapaus Testi 2

Testaa Kuten testi 1, mutta maamerkit on aseteltu siten, että paikanmääritys epäonnistuu ensimmäisellä kerralla esteen läheisyydessä.

- `RANDOMVECTOR_COUNT > 0`
- Testi on onnistunut, jos randomWalk käynnistyy ja paikanmääritystä yritetään uudelleen ajoparametrien mukaisesti.

Tulos Testi onnistui.

Tapaus Testi 3

Testaa Kuten testi 2, mutta paikanmääritys estetään yllättävän esteen jälkeen peittämällä maamerkit.

- Testi on onnistunut, jos sonarihavainnon jälkeen satunnaisvektoreita suoritetaan `RANDOMVECTOR_COUNT` kertaa ja paikanmäärityksiä `RANDOMVECTOR_COUNT+1` kertaa (kun ensimmäinen paikanmääritys esteen vieressä lasketaan mukaan). Tämän jälkeen ohjelman pitää päättyä järkevään virheilmoitukseen.

Tulos Testi onnistui.

Tapaus Testi 4

Testaa Toimiiko robotti järkevästi, kun se havaitsee yllättävän esteen marginaalialueen sisällä?

- kartta: `D427.asc`
- sonarit eivät saa havaita karttaan merkittyjä esteitä lähtöpisteessä
- lähtöpiste keskellä marginaalialuetta
- pysäytysetäisyys robotin reunasta laskien 10 cm
- marginaali: `MARGIN = 100`
- paikanmäärityksen pitää onnistua lähtöpisteessä
- este asetetaan pysäytysetäisyyden sisälle. Este ei sulje reittiä ulos marginaalista eikä estä paikanmääritystä.
- Testi on onnistunut, jos

1. robotti ilmoittaa sonarihavainnosta
2. odottaa
3. tekee paikanmäärityksen
4. ilmoittaa yllättävästä esteestä
5. yrittää liikkua uudelleen
6. ohjelma päätttyy

Tulos Testi epäonnistui. Robotti ilmoittaa havainneensa karttaan merkityn esteen ja liikkuu ulos marginaalista.

Tapaus Testi 5

Testaa Kuten testi 4, mutta yllättävä este on marginaalin ja vapaan alueen rajalla. Robotti on karttaan merkityn seinämän ja yllättävän esteen välissä, eli suora reitti ulos marginaalista on suljettu.

Tulos Testi epäonnistui. Robotti joutui ikuiseen silmukkaan: se tekee paikanmäärityksen ja ilmoittaa havainneensa karttaan merkityn esteen ja yrittää ulos marginaalista jne...

Tapaus Testi 6

Testaa Toimiiko `q / Q` kesken `randomWalkin()`?

Tulos Testi onnistui.

Tapaus Testi 7

Testaa `WAIT`–parametri asetetaan nollaksi. Jääkö odotus pois, kun sonarit havaitsevat esteen?

Tulos Testi onnistui. Odotus jää pois.

Tapaus Testi 8

Testaa Kuten testi 7, mutta `WAIT` on negatiivinen.

Tulos Testi onnistui. Robotti käyttäytyy kuten testissä 7.

Tapaus Testi 9

Testaa `ETAP_LENGTH` asetetaan nollaksi. Tarkistetaanko sijaintia ikuisessa silmukassa?

Tulos Testi onnistui. Ikuista silmukkaa ei synny. Etappitarkistus tehdään kahden metrin välein, jos syötteenä annettu arvo ei ole kelvollinen.

Tapaus Testi 10

Testaa Suoritetaanko satunnaisvektoreita, kun `RANDOMVECTOR_COUNT = 0` ja `RANDOMVECTOR_LENGTH = 100`.

- Onnistunut testi: Satunnaisvektoreita ei suoriteta, kun paikanmääritys epäonnistuu.

Tulos Testi onnistui.

2.2 Koko järjestelmän integrointitestit

Tapaus Testi 1

Testaa Pysäytysetäisyys ei saa olla marginaalia suurempi. Sen sijaan marginaalin olisi järkevä olla vähintään viisi senttiä suurempi kuin pysäytysetäisyys.

- `MARGIN = 20 cm`
- pysäytysetäisyys 40 cm
- Testi on onnistunut: käyttäjälle annetaan virheilmoitus ja lopetetaan, tai järjestelmä asettaa arvot uudelleen, että marginaali on vähintään viisi senttiä suurempi.

Tulos Ei onnistunut. Arvoja ei rajoiteta millään tavalla.

Tapaus Testi 2

Testaa Kuten testi 1, mutta

- `MARGIN = 10`
- pysäytysetäisyys 0 (virheellinen syöte, jota ei pidä hyväksyä)
- Testi on onnistunut: kuten testi 1
- Mitkä ovat sopivat alarajat marginaalille ja pysäytysetäisyydelle?

Tulos Ei onnistunut. Arvoja ei rajoiteta.

Tapaus Testi 3

Testaa Ajotiedostosta otetaan kokonaan pois merkintä `LANDMARK_FILE = x`. Järjestelmän pitää ilmoittaa, että parametri `LANDMARK_FILE` puuttuu.

Tulos Testi onnistui.

Tapaus Testi 4

Testaa Kuten testi 3, mutta ajotiedostossa on merkintä `LANDMARK_FILE =`, mutta parametrin arvo puuttuu. Järjestelmän pitää ilmoittaa, että parametrin `LANDMARK_FILE` arvo puuttuu.

Tulos Testi onnistui.

Tapaus Testi 5

Testaa Kuten testi 3, mutta parametrille `SCALE`.

Tulos Testi onnistui.

Tapaus Testi 6

Testaa Kuten testi 4, mutta parametrille `SCALE`.

Tulos Testi ei onnistunut: Arvo asetettiin nollaksi ja ilmoitettiin, että arvon pitää olla vähintään 1.

Tapaus Testi 7

Testaa Parametrin `SCALE` arvoksi merkitään 0.5. Testi on onnistunut, jos järjestelmä ilmoittaa, että parametrin pienin mahdollinen arvo on yksi sentti. Tämän jälkeen ohjelma lopetetaan.

Tulos Testi onnistui.

Tapaus Testi 8

Testaa Järjestelmä ei saa kaatua, kun `double`-arvoisen parametrin arvoksi asetetaan merkki.

- Merkitään ajotiedostoon `SCALE = c`.
- Testi on onnistunut, jos järjestelmä ei kaadu, vaan antaa virheilmoituksen, että muuttujan SCALE arvo on virheellinen, ja lopettaa ohjelman.

Tulos Testi ei onnistunut: Arvo asetettiin nollaksi ja ilmoitettiin, että arvon pitää olla vähintään 1.

Tapaus Testi 9

Testaa Maalipiste ei saa olla marginaalissa.

- Merkitään ajotiedostoon maaliksi marginaaliin kuuluva piste.
- Testi on onnistunut: Ohjelma päättyy virheilmoitukseen, että piste (x,y) kuuluu marginaaliin.

Tulos Onnistui.

Tapaus Testi 10

Testaa Kuten testi 9, mutta maalipisteeksi asetetaan esteeseen kuuluva piste. Ohjelman pitää päättyä testin 9 mukaisesti informatiiviseen virheilmoitukseen.

Tulos Onnistui.

Tapaus Testi 11

Testaa Kuten testi 9, mutta maalipisteeksi merkitään kartan ulkopuolinen piste. Järjestelmän pitää ilmoittaa, että piste on kartan ulkopuolella.

Tulos Testi onnistui: järjestelmä ilmoitti, että piste kuuluu esteeseen.

Tapaus Testi 12

Testaa Ajotiedoston parametrin `WRITE_YUV` arvoksi merkitään ensin 'n', sitten merkkijono "njii", "yhhh" ja lopuksi ei mitään. Ohjelman pitää päättyä virheilmoitukseen, jossa ilmoitetaan, että parametrin `WRITE_YUV` arvo on virheellinen. Tai vaihtoehtoisesti parametrille asetetaan oletusarvo. Ohjelma ei saa kaatua epämääräisesti.

Tulos Testi onnistui. Kun syöte oli virheellinen, annettiin muuttujalle oletusarvoksi n, eli kuvia ei tallennettu.

Tapaus Testi 13

Testaa ovatko tulosteet järkeviä, kun kaikkien osajärjestelmien tulostustasoksi merkitään hyvin tekninen: `MAP_LOG = 0` ja niin edelleen.

- Suoritetaan lyhyt ajo robotilla.
- Testi on onnistunut, jos kaikki lokin tulosteet ovat oikeinkirjoitettua englantia. Lokissa pitää olla kamerakuvista havaitut hahmot.

Tulos Onnistui.

Tapaus Testi 14

Testaa Lokin tulostustasoiksi merkitään 1. Tulosteissa ei saa olla ylimääräisiä debuggaustietoja.

Tulos Onnistui.

Tapaus Testi 15

Testaa Lokin tulostustasoiksi merkitään 2. Lokissa saa olla vain korkeamman tason tulosteita, jotka ymmärrettävästi kuvaavat järjestelmässä tapahtuvaa laskentaa.

Tulos Onnistui.

Tapaus Testi 16

Testaa Lokin tulostustasoiksi merkitään 3. Lokissa saa olla vain välttämättömät tulosteet, jotka kuvaavat järjestelmän laskentaa pääpiirteittäin.

Tulos Onnistui.

Tapaus Testi 17

Testaa

- Muutetaan tiedostoa `SelfTestingObject.h`: Otetaan kommentoinnit pois kohdasta `#define NDEBUG`, että kaikki assertit saadaan pois ohjelmasta. Lisätään muutettu tiedosto CVS:ään.
- Otetaan tuore koodi CVS:stä ja käännetään koko järjestelmä martissa. Ohjelman pitää kääntyä ilman muutoksia koodiin.

Tulos Onnistui.

Kirjallisuutta