Tomi Ilonen, Tommi Nuotiomaa, Matias Pitkänen

Kokoava projekti

Lego Mindstorms

Metropolia Ammattikorkeakoulu
Koneautomaatio
Kokoava projekti
9.4.2012

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Tomi Ilonen, Matias Pitkänen, Tommi Nuotiomaa Lego Mindstorms 16 sivua + 1 liite 11.5.2012
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Kone- ja tuotantotekniikka
Suuntautumisvaihtoehto	Koneautomaatio
Ohjaaja(t)	Vastuuopettaja: Jari Savolainen

Tämä projekti tehtiin yhteistyössä Metropolia ammattikorkeakoulun kanssa. Työn tavoitteena oli tilata koulun Lego Mindstorms robotteihin uusia antureita, testata niiden toiminta sekä tehdä Metropolian Wikiin yksinkertaiset esimerkit ja tehtävät kustakin anturista, joita tullaan mahdollisesti hyödyntämään tulevien opiskelijoiden opetuksessa. Toissijaisena tavoitteena oli pitää Haminassa aiheesta demoesitys.

Työn teoriaosuudessa kerrotaan lyhyesti Lego Mindstorms roboteista sekä RobotC -ohjelmointikielestä. Teoriaosuudessa kerrotaan myös tilatuista HiTechnicin antureista.

Työn käytännön osuus koostui antureiden tilaamisesta, asennuksesta sekä niiden toiminnan testauksesta. Käytännön osuus painottui antureiden toiminnan testaukseen eli RobotC -ohjelmointikielellä koodien kirjoittamiseen.

Suurin hyöty projektista koululle ovat valmistuneet esimerkit ja tehtävät tilatuista antureista, koska koulu voi hyödyntää niitä tulevien insinööriopiskelijoiden opetuksessa. Lisäksi tätä projektia voidaan tulevaisuudessa jatkaa toisen ryhmän toimesta.

Avainsanat	

Metropolia

Abstract

Author(s) Title Number of Pages Date	Tomi Ilonen, Matias Pitkänen, Tommi Nuotiomaa Lego Mindstorms 16 pages + 1 appendix 11 May 2012
Degree	Bachelor of Engineering
Degree Programme	Mechanical Engineering
Specialisation option	Machine Automation
Instructor(s)	Teacher in Charge: Jari Savolainen

This project was done in cooperation with Metropolia University of Applied Sciences. The goal of the project was to order new sensors for the school's Lego Mindstorms robots and test them out and make simple examples and tasks of each of the sensors to Metropolia Wiki which could be utilized in teaching with future students. Secondary goal was to keep a demo presentation in Hamina on the subject.

Lego Mindstorms robots and RobotC programming language are explained shortly in the theory part of the thesis. Also the sensors ordered from HiTechnic are explained in the theory part.

The practical part consists of ordering the sensors, installation and testing out the functionality of them. In the practical part the focus was on testing in other words writing code in RobotC.

The greatest benefits for the school are the examples and tasks made for the sensors because school can make use of them with the future engineering students. In addition, this project can be continued in future by another group.

Keywords	

Sisällys

1 Johdanto

Lego Mindstorms on Lego Groupin valmistama tuotesarja, josta voi rakentaa erilaisia robotteja. Sarjan paketit koostuvat keskuspalikoista, antureista ja moottoreista sekä tavallisista Lego Technic -palikoista. NXT versio julkaistiin vuonna 2006 ja se sisältää neljä anturia: valo-, ultraääni-, kosketus-, ja äänianturi.

Projektin tarkoituksena on kartoittaa eri antureiden valmistajia, tilata uusia antureita Lego Mindstorms robotteihin, testata niiden toimintaa sekä tehdä niistä yksinkertaiset esimerkit ja tehtävät Metropolian Wikiin tulevien insinööriopiskelijoiden opetusmateriaaliksi. Projektissa keskitytään lähinnä antureiden toiminnan testaukseen sekä esimerkkien ja tehtävien tekemiseen. Projektissa käytetään oppilaitoksen tietokoneita ja niistä valmiiksi löytyvää RobotC -ohjelmaa.

Projektin kirjallisessa osuudessa käsitellään projektin eri osa-alueita. Kirjallisessa osuudessa on esimerkki koodit, jotka kirjoitimme sekä yleiskatsaus Lego Mindstorms robotteihin sekä RobotC -ohjelmointikieleen. Projektin käytännön osuus koostui antureiden tilaamisesta sekä niiden toiminnan testauksesta.

2 Lego Mindstorms

Lego Mindstorms (virallinen nimi Lego Mindstorms Robotic Invention System) on Lego Groupin valmistama tuotesarja, josta voi rakentaa erilaisia robotteja. Tuotesarjaan kuuluu monia erilaisia pakkauksia, joista voi rakentaa erilaisia robotteja. Pakkaukset koostuvat mikropiirillä varustetuista keskuspalikoista, niihin liittyvistä antureista ja moottoreista sekä tavanomaisista Lego Technic -palikoista. Ensimmäinen Lego Mindstorms versio, RCX, julkaistiin vuonna 1998 ja toinen versio, NXT, vuonna 2006. Projektissamme käytimme NXT versiota.

Sekä RCX- että NXT -keskuspalikat ovat ohjelmoitavissa. Helpoin tapa ohjelmoida keskuspalikoita on käyttää pakkauksen mukana tulevaa graafista ohjelmointityökalua. Ohjelmat rakennetaan erilaisista palikoista, joiden avulla voidaan määritellä esimerkiksi tiettyjen moottorien pyöriminen anturien luvut jne. Näin voidaan esimerkiksi rakentaa ohjelma, joka vierittää keskuspalikkaa moottorien avulla pyörittämällä renkaita X kierrosta ja sitten lukea kosketusanturin arvon.

Ohjelmaa robotille voi tehdä myös ”oikeilla” ohjelmointikielillä, kuten C:llä tai C++:lla. Käytimme projektissamme RobotC kehitysympäristöä, joka on ehkä helpoin tapa ohjelmoida melkein oikealla C:llä. RobotC:llä voi tehdä helpoiten kaiken mitä Mindstorms tarjoaa.

2.1 NXT

NXT versio sisältää kolme sähkömoottoria ja neljä anturia: valo-, ultraääni-, kosketus- ja äänianturin. Pakettiin kuuluvien antureiden lisäksi on olemassa paljon muitakin antureita, joita voi tilata esim. Legon internet sivuilta. Markkinoilta löytyy myös muiden valmistajien tekemiä antureita, jotka toimivat Lego Mindstorms roboteissa. Käytimme projektissamme HiTechnicin valmistamia antureita. Lisäksi sarja sisältää keskuspalikan ja erinäisen määrän Lego Technic osia. NXT ohjelmoidaan tietokoneella. Projektissamme käytimme RobotC ohjelmointikieltä. NXT:ssä on kolme ulostuloporttia, neljä sisääntuloporttia ja yksi USB -portti, jonka kautta tietokoneella ohjelmoitu ohjelma siirretään NXT:iin. NXT on varustettu bluetoothilla.

2.2 RCX

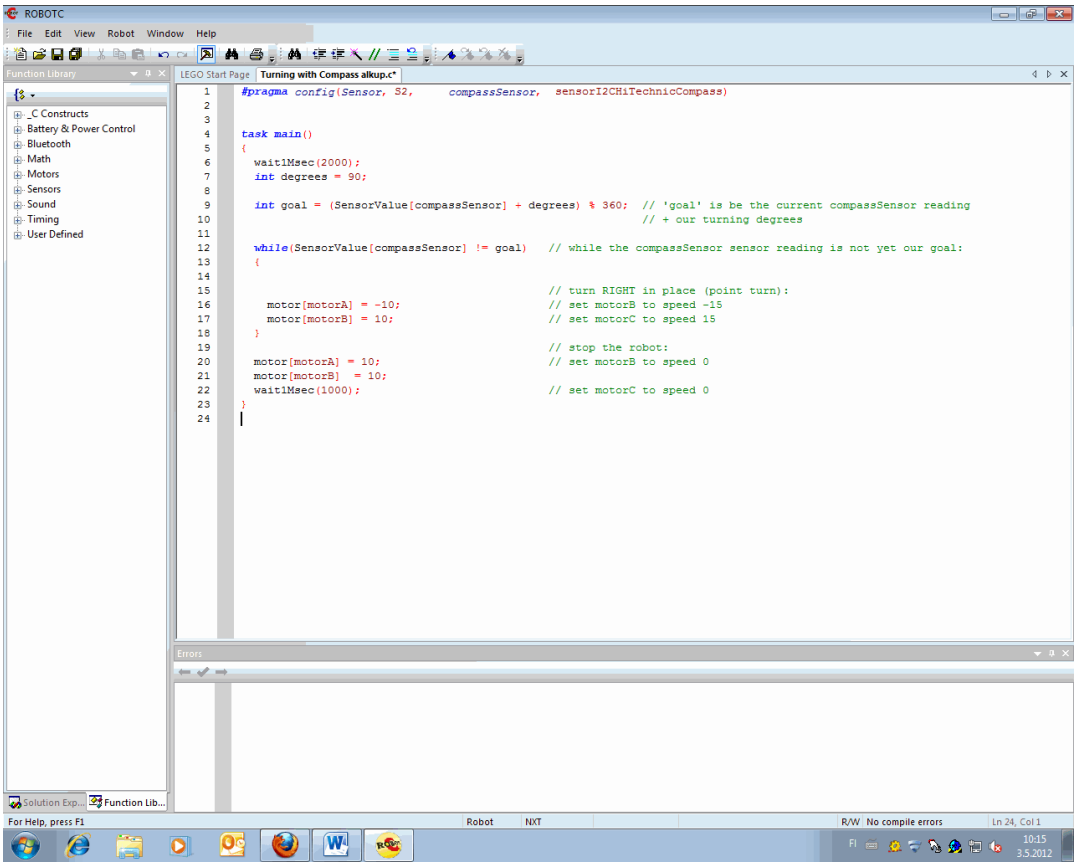
RCX versio on NXT version edeltäjä. RCX versiossa on kolme anturiporttia ja kolme sähkömoottoriporttia. Se ohjelmoidaan tietokoneella tätä varten kehitetyllä ohjelmalla. Ohjelmia voidaan ajaa joko suoraan tietokoneen kautta tai tallentamalla ne RCX -keskuspalikkaan josta niitä voidaan ajaa nappia painamalla. RCX versioita saa nykyään vain huutokaupoista tai kirpputoreilta, koska RCX on korvattu uudemmalla NXT versiolla.

2.3 Cybermaster

Cybermaster ohjelmoitava keskuspalikka julkaistiin samana vuonna alkuperäisen RCX:n kanssa. Cybermaster -tuotesarja on suunnattu nuoremmille kuin RCX ja se oli integroitu tehokkaasti setin mukana tulleeseen tietokonepeliin. Cybermasterissa eroaa RCX:stä esimerkiksi siten, että siinä on vain yksi ulostuloportti sekä yksi sisääntuloportti.

3 RobotC

RobotC -kieli pohjautuu C-kieleen. Siitä on karsittu turhia ominaisuuksia, joita ei tarvita Mindstormsin ohjelmoinnissa, joten se on ehkä helpoin tapa ohjelmoida Mindstorms robotteja melkein oikealla C -kielellä. Kehitysympäristö on maksullinen, mutta kokeiluversio on ladattavissa ilmaiseksi. Jokaisessa RobotC -ohjelmassa (kuvio 1) on samanlainen ”task main” runko. RobotC -kieli tunnistaa isot ja pienet kirjaimet. Kirjoitettu koodi käännetään tietokoneen/robotin ymmärtämään muotoon, konekielisiksi käskyiksi. Tämä tapahtuu, kun ohjelma käännetään. Jos käännös menee läpi, eli kääntäjä on tulkinnut ohjelmoijan kirjoittamat käskyt hyväksytysti, tuottaa se binääritiedoston (tiedoston, joka voidaan suorittaa robotissa), joka voidaan ladata robotille ja suorittaa.



Kuvio 1. RobotC ohjelma ja yksinkertainen ohjelmanosa

RobotC–kieli koostuu tekstipohjaisista komennoista ja ohjelma sisältää useita oppaita, joiden tarkoituksena on auttaa käyttäjää oppimaan ohjelmointikieli. Ohjelmassa on myös virheenkorjaustoiminto, joka helpottaa virheiden löytämistä sekä niiden korjaamista. Osa ohjelmasta on mahdollista merkitä kommentiksi, joka tarkoittaa sitä että kommenttia ei oteta huomioon ohjelmaa käännettäessä. Tämä mahdollistaa sen, että koodia voi kommentoida ja sanallisesti selittää mitä siinä tapahtuu, joka helpottaa muistamista. Toinen mahdollinen käyttötarkoitus kommenttitoiminnolle on se, että sen avulla voi tilapäisesti poistaa käytöstä jonkin osan koodista, kuitenkin samalla säilyttäen sen.

4 HiTechnic tuotteet

HiTechnicin tuotteet ovat Legon kanssa yhteen sopivia, ja niiden käyttö sekä ohjelmointi ovat mahdollista myös RobotC ohjelmointiympäristössä. Lisäksi markkinoiden edullisin hinta ja laajin valikoima tekivät tavarantoimittajan valinnasta helppoa. HiTechnic antureiden helppo tilaaminen Suomeen oli kuitenkin tärkein kriteeri, jonka perusteella se valittiin.

4.1 HiTechnic anturit

HiTechniciltä löytyy useita erityyppisiä antureita ja niiden sovelluspaketteja. Tuotevalikoimat ja käyttötarkoitukset laajenevat kokoajan, joten kaikkia antureita ja laitteita emme valinneet projektiimme. Valitsimme niistä vain mielestämme opetustarkoituksessa käytännöllisimmät ja helppokäyttöisimmät.

4.1.1 Kulma-anturi

HiTechnic NXT Angle Sensor (kuvio 2) mittaa kolmea eri pyörimisominaisuutta yhdeltä akselilta. Anturia voidaan parhaiten hyödyntää laitteissa, joissa akselin pyörimisliike on avainasemassa esimerkiksi tuulennopeusmittarin tai mittauspyörän yhteydessä. Se mahdollistaa pyörimisnopeuden, pyörittyjen asteiden määrän sekä akselin aseman mittaamisen samalla kertaa. Pyörimisnopeuden anturi ilmoittaa yhden yksikön tarkkuudella kierroksina minuutissa (RPM, revolutions per minute), akselin aseman ja pyörittyjen asteiden määrän anturi ilmoittaa yhden asteen tarkkuudella.

Kuvio 2. HiTechnic NXT Angle Sensor

4.1.2 Kiihtyvyysanturi

HiTechnic NXT Acceleration / Tilt Sensor (kuvio 3) mittaa kiihtyvyyttä kolmen eri akselin suuntaisesti. Anturia voidaan käyttää +2g ja -2g välisellä kiihtyvyys alueella, jolloin kiihtyvyyden aiheuttamia voimia voidaan laskea. Yksi *g* vastaa noin 200 mittausta akselia kohti. Anturi mittaa myös kallistusta jokaisen akselin suhteen, joten sitä voidaan hyödyntää myös tasapainottelevissa roboteissa. Anturin jokaisen akselin mittaustulos päivittyy noin sata kertaa sekunnissa.

Kuvio 3. HiTechnic NXT Acceleration/Tilt Sensor ja mittausakseleiden asettelu.

4.1.3 Ilmanpaineanturi

HiTechnic NXT Barometric Sensor (kuvio 4) mittaa ilmanpainetta ja lämpötilaa. Sen mittaustuloksina saadaan ilmanlämpötila Celsius- tai Farenheit -asteina kymmenesosan tarkkuudella sekä ilmanpaine millibaareina (hPa=mbar) tai tuumina elohopeaa (inHg) tuhannesosa tuuman tarkkuudella. Kun anturin on kalibroinut oikein sen voi myös saada laskemaan korkeutta jalkoina tai metreinä.

Kuvio 4. HiTechnic NXT Barometric Sensor

4.1.4 Värianturi

HiTechnic NXT Color Sensor (V2) (kuvio 5) tunnistaa yli 15 väriä. Anturi on asetettu tunnistamaan Legon perusvärejä. Anturin avulla voidaan rakentaa esimerkiksi viivaa seuraava tai värejä lajitteleva robotti. Anturi käyttää valkoista LED valoa ja vastaanotinta värien tunnistuksessa. Kun anturin eteen tuodaan väri, se tunnistaa sen ja ilmoittaa sitä vastaavan, ennalta määrätyn numeroarvon. Värien arvoa laskeva osa anturista päivittyy noin sata kertaa sekunnissa. Tämä uudistettu versio (V2) on paranneltu väriskaalan laajuuden ja tarkkuuden osalta.

Kuvio 5. HiTechnic Color Sensor (V2) ja sen käyttöesimerkki.

4.1.5 Gyroskooppianturi

HiTechnic NXT Gyro Sensor (kuvio 6) mittaa kallistuskulman muutosnopeutta. Anturia voidaan hyödyntää roboteissa, joissa halutaan havaita kallistusliikkeet tarkasti, kuten heilureissa tai joissain tasapainoa vaativissa projekteissa. Anturin mittaustuloksena saadaan pyörittyjen asteiden määrä sekunteina sekä pyörimisen suunta, jota havainnoidaan plus- ja miinusmerkein. Anturi tekee havaintoja noin 300 kertaa sekunnissa.

Kuvio 6. HiTechnic Gyro Sensor ja sen mittaus alue.

4.1.6 Kompassianturi

HiTechnic NXT Compass Sensor (kuvio 7) toimii digitaalisena kompassina. Sen avulla voidaan rakentaa navigoivia robotteja. Kompassi anturi lukee maan magneettikenttää ja asettaa numeraalisen arvon sen hetkiselle suunnalle. Anturi päivittyy sata kertaa sekunnissa ja se antaa suunnan asteen tarkkuudella välillä 0°-359°. Anturilla toimii kahdessa eri tilassa, suunnan luku ja kalibrointi. Luku tila on päällä kokoajan kun anturi mittaa suuntaa, mutta kalibrointi on tehtävä erikseen. Kalibroinnin tarkoituksena on kompensoida haittatekijöiden vaikutuksia.

Kuvio 7. HiTechnic Compass Sensor

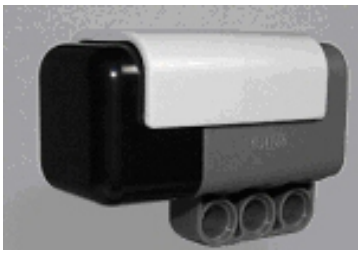
4.1.7 EOPD -anturi

HiTechnic NXT EOPD Sensor (kuvio 8) tarkkailee kohteen etäisyyttä ja sen muutoksia. Anturi käyttää havainnoidessaan hyödyksi omaa valonlähdettään poistaakseen kirkkaiden ja himmeiden valaistusympäristöjen vaikutukset. Anturi voi havaita kohteen jo 20cm päästä riippuen sen koosta, muodosta ja heijastavuudesta. Anturi päivittää noin 300 tulosta sekunnissa ja se toimii kahdella tarkkuus tasolla, x1 ja x4. Niin sanottu raa'at luvut syntyvät x4 tarkkuudella ja todelliset luvut x1 tarkkuudella. Kohteen lähestyessä raa'at ja todelliset luvut lähenevät toisiaan.

Kuvio 8. HiTechnic EOPD Sensor

4.1.8 Infrapunasäteilyn vastaanottoanturi

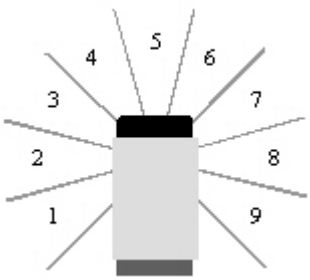
HiTechnic NXT IRReceiver Sensor (kuvio 9) vastaan ottaa siis infrapunasäteilyä. Sen käyttö mahdollistaa robotin kauko-ohjauksen. Anturi vastaanottaa Legon omien kauko-ohjainten lähettämiä signaaleja, joilla voidaan suoraan ohjata jokaista käytössä olevaa moottoria erikseen tai yhdessä.



Kuvio 9. HiThecnic IRReceiver Sensor

4.1.9 Infrapunasäteilyn hakuanturi

HiTechnic NXT IRSeeker V2 (kuvio 10) on myös infrapunasäteilyä vastaanottava anturi. Se mahdollistaa kauko-ohjauksen myös suurimalla osalla kaikista normaaleista TV -kaukosäätimistä. Sen kurvikas linssin muotoilu ja viisi sisäistä tunnistinta mahdollistaa 240 asteen havaintoalueen. Havaintoalue on jaettu yhdeksään osaan, jolloin anturi osaa kertoa suunnan, mistä saapuva säteily on voimakkain. Näiden tietojen avulla anturia hyödyntävä robotti pärjää erinomaisesti esimerkiksi jalkapallo pelissä.



Kuvio 10. HiTechnic IRSeeker V2 anturi ja havaintoalueet.

4.1.10 Infrapunasäteilyn linkkianturi

HiTechnic NXT IRLink Sensor (kuvio 11) kommunikoi vanhempien Lego tuoteperheiden kanssa infrapunasäteilyä hyödyntäen. Anturi toimii linkkinä uudempien NXT ohjelmien

ja vanhempien RCX antureiden sekä moottoreiden välillä. Se mahdollistaa RCX antureiden käytön NXT ohjelmissa.

Kuvio 11. HiTechnic IRLink Sensor

4.2 HiTechnic laitteet

HiTechnicin valikoimista löytyy myös useita elektronisia laitteita ja robottien lisäosia, joiden avulla robottien toiminnat ja tehtävät saadaan vielä uudelle tasolle. Valikoimista löytyy muun muassa erilaisia tuotesarjoja, jotka mahdollistavat esimerkiksi aineiden ominaisuuksien tutkimista. Projektiimme valittiin vain muutama lisälaite luomaan uusia ulottuvuuksia koulumme roboteille.

4.2.1 Infrapunapallo

HiTechnic Infrared Electronic Ball (kuvio 12) on pallo, joka lähettää infrapunasäteilyä neljällä eri tavalla. 20 infrapunalamppua voi lähettää, joko pulssimaista tai tasaista säteilyä. Tasaisen säteilyn taajuuden voi valita 600 Hz tai 1200 Hz väliltä, jolloin havaintoalue on vähintään viisi metriä. Pulssimaisen säteilyn avulla havaintoaluetta saadaan kasvatettua ja vastaanottavien antureiden on helpompi havaita kohde taustastaan. Palloa käytettään usein robottien välisissä ”jalkapallo” peleissä.

Kuvio 12. HiTechnicin Infrared Electronic Ball.

4.2.2 Kauko-ohjain

HiTechnic IR Remote Control (kuvio 13) mahdollistaa yhdessä IRReceiverin (kuvio 9) kanssa robottien kauko-ohjauksen. Kauko-ohjaimella voidaan kytkeä päälle tai pois päältä jokin moottoreista. IR Speed Remote Control puolestaan mahdollistaa on/off -kytkinten sijaan moottoreiden nopeuden säätelyn.

Kuvio 13. HiTechnic IR Speed Remote Control (vasen) ja IR Remote Control (oikea).

5 Mindsensors Vision Subsystem v4 for NXT

5.1 Mindsensors

Mindsensors on toinen legon Mindstorms roboteille antureita ja muita lisälaitteita valmistava yritys. Toisin kuin HiTechnicin ulkomuodoiltaan ja asennettavuudeltaan yhteneväiset sensorit, Mindsensorsin sensorit ovat ulkoisesti erimallisia. Mindsensors:n suppeamman anturivalikoiman vuoksi käytimme sitä vain Vision Subsystem v4 for NXT (NXTCam-v4)– kameran tilaamiseen.

5.2 Vision Subsystem v4 for NXT

Mindsensors Vision Subsystem v4 for NXT (NXTCam-v4)– kameralla (kuvio 14) voidaan seurata reaaliajassa viivaa tai maksimissaan kahdeksaa eri värikohdetta eli se on ominaisuuksiltaan samankaltainen kuin värisensori, mutta pidemmällä ja laajemmalla kantamalla. Väriskaalat määritellään kameraan erillisellä ohjelmalla tietokoneelta käsin. Kameran linssin tarkkuus on säädeltävissä ja vakiolinssi pitää sisällään myös infrapunavalolta suojaavan suotimen, joka poistaa valosta aiheutuvia häiriötekijöitä.

Kamerasta on mahdollista saada reaaliaikaista tietoa muun muassa kohteiden määrästä, väristä, kohteiden raja-arvojen koordinaatteja tai seurattavan viivan koordinaatteja, mutta reaaliaikaista liikkuvaa kuvaa siitä ei voida siirtää tietokoneen näytölle. Kamera pystyy seuraamaan liikkuvaa kuvaa 88x144 pikselin resoluutiolla, 30 kuvaa sekunnissa (30fps) sekä siirtämään täyden resoluution (176x144 pikseliä) kuvavedoksia koneeseen.

Kuvio 14. Vision Subsystem v4

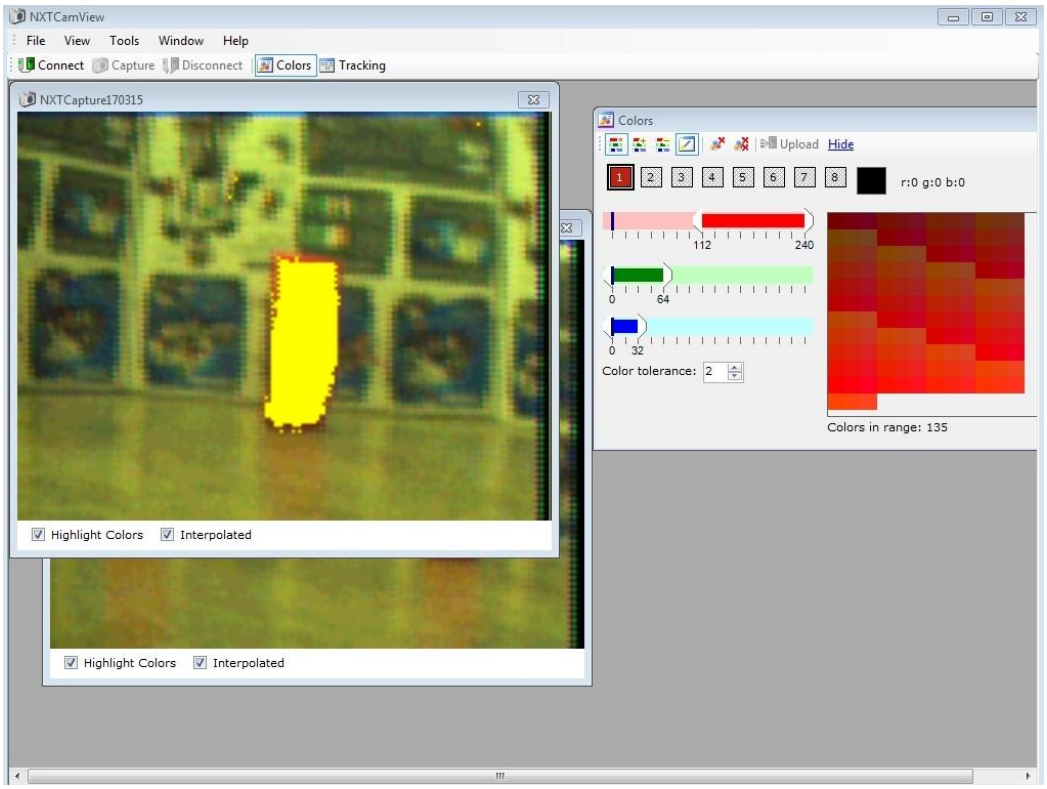
Kamera kytketään Lego Mindstorms robottiin standardin sensoriportin kautta ja tarvittaessa tietokoneeseen mini-USB-johdon kautta. Ohjelmointi onnistuu useassa eri ympäristössä, mukaan lukien RobotC:ssä. Kameran käyttöönotto vaatii käyttäjältä enemmän valmisteluja kuin muiden sensorien kanssa ja kameraa hyödyntävän ohjelman tekeminen vaatii edistynyttä ohjelmointitaitoa sekä analyyttisiä taitoja.

5.2.1 NXTCamView

NXTCam-v4 tunnistaa kiinnostuksen kohteet vertaamalla siihen taltioituja väriarvoja, sen ottamaan kuvaan. Tämä tapahtuu siten, että kiinnostuksen kohteen/kohteiden värit tallennetaan kameran muistiin. Näitä arvoja kutsutaan väriskaaloiksi (tai värikartoiksi).

Väriskaalat pystytään määrittelemään kameraan helposti erillisellä NXTCamView- ohjelmalla, joka on tehty varta vasten kameraa varten. Kamera kytketään kiinni tietokoneeseen ja NXTCamView:ssä (kuvio 15) sillä otetaan kuva (mielellään siinä ympäristössä, jossa robottia aiotaan käyttää). Kuvasta määritellään tämän jälkeen kohteen väri-

skaalat punaisena, vihreänä ja sinisenä, arvoasteikolla 0-240, kuten kuvasta (Kuvio 15) voi havaita. Useita värejä kameraan määriteltäessä, väriskaaloissa ei saa olla päällekkäisyyksiä ja NXTCamView huomauttaa asiasta, mikäli niitä syntyy. Kun väriskaalat on määritelty, ne voidaan lähettää kameraan.



Kuvio 15. NXTCamView. Keltaisella näkyvät alueet ovat määritellyn väriskaalan sisällä.

5.2.2 Ympäristö

Kameraa käytettäessä tulee ottaa huomioon ympäristö, jossa sitä käytetään. Ympäristön tulisi ihanneoloissa olla mahdollisimman yksivärinen, kuten valkoinen, ja valaistuksen olla valkoista loisteputkivaloa, johon kamera on suunniteltu ja sen tulisi tulla suoraan ylhäältäpäin varjojen muodostumisen välttämiseksi.

Käytännössä puolestaan, ympäristössä on usein muitakin värejä, mutta tulisi huolehtia, ettei ympäristössä ole ainakaan liian lähellä kohteen väriä olevia värejä. Tämän vuoksi on kannattavaa ottaa NXTCamView:llä kuvia myös ympäristöstä, jossa robottia käytetään, kun kohteen väri on määritelty ja tarkastaa onko ympäristössä väriskaalaan sisältyviä värejä, joita robotti voisi ohjelmaa ajettaessa tunnistaa kohteiksi.

6 Laitteiden ja antureiden käyttö projektissa

Kaikkien antureiden osalta tehtiin yksinkertainen testikoodi, jonka avulla voidaan selvittää anturin toimivuuden lisäksi sen antamat tiedot. Testikoodien tarkoitus on toimia opetusmateriaaleina ja esimerkkeinä, joiden pohjalta saadaan luotua erilaisia toimintoja roboteilla. Joidenkin antureiden kohdalla lähdettiin myös luomaan hieman haasteellisimpia mallitöitä. Testikoodien tekemiseen käytimme koulun tietokoneita, joissa on RobotC -ohjelmat. Käytimme myös koulun Mindstorms robotteja, jotka oli valmiiksi kasattu samanlaisiksi. Tämä mahdollistaa testikoodien käytön niillä kaikilla. Osaa roboteista muokkasimme, jotta pystyimme kokeilemaan haasteellisempien koodien toimintaa. Teimme esimerkiksi Vision Subsystem v4 kameran avulla robotin, joka ampuu punaiset kohteet.

7 Tehtävien ja esimerkkien luominen

Teimme antureista yksinkertaiset tehtävät, jotka lisäsimme Metropolian Wikiin. Teimme antureista myös helpot esimerkit (kuvio 16), joiden avulla tehtävien tekeminen pitäisi onnistua. Nämä esimerkit laitoimme myös Wikiin.

```
#pragma config(Sensor, S2,     ANGLE,              sensorI2CCustom)

#include "drivers/HTANG-driver.h"

task main () {

  nxtDisplayCenteredTextLine(3, "WAIT A MOMENT!");


  wait1Msec(3000);

  eraseDisplay();
  nxtDisplayCenteredTextLine(0, "HiTechnic Angle");                        // Just graphics for the display
  nxtDisplayTextLine(1, "-------------------");
  nxtDisplayTextLine(5, "-------------------");
  while (true) {

    if (nNxtButtonPressed == kLeftButton) {                                //left button resets angle value
      HTANGresetAngle(ANGLE);
      while (nNxtButtonPressed != kNoButton) EndTimeSlice();

    } else if (nNxtButtonPressed == kRightButton) {                        //right button resets all
      HTANGresetAccumulatedAngle(ANGLE);
      while (nNxtButtonPressed != kNoButton) EndTimeSlice();
    }


    nxtDisplayTextLine(2, "Ang: %7d deg", HTANGreadAngle(ANGLE));          // Read the current angle,
    nxtDisplayTextLine(3, "Tot: %7d deg", HTANGreadAccumulatedAngle(ANGLE)); // total angle and
    nxtDisplayTextLine(4, "RPM: %7d", HTANGreadRPM(ANGLE));                // RPM and display them
    nxtDisplayTextLine(6, " <-Reset Angle");
    nxtDisplayTextLine(7, "   Reset Total->");
    wait1Msec(50);
  }
}
```

Kuvio 16. Kulma-anturin esimerkkikoodi.

Suurin osa esimerkeistä on tehty sellaisiksi, että niiden avulla nähdään minkälaista tietoa anturi antaa, mitä arvoja se muuttaa milloinkin sekä ylipäätänsä toimiiko anturi. Joihinkin on lisätty joitakin robotin yksinkertaisia liikkeitä havainnollistamaan arvojen muutoksia.

Vision Subsystem v4 kameran käytöstä tehty esimerkki on valmiiksi toimiva koodi, jonka tarkoituksena on helpottaa kyseisen kameran käytön opettelua. Koodin sisällä voi muuttaa tiettyjä arvoja ja havainnoida mihin se johtaa. Kameran käyttöönotolle ja käytölle on tehty kattavat ohjeet Metropolian Wikiin.

8 Tulokset

Projektin ensisijaisena tavoitteena oli luoda esimerkkejä ja tehtäviä tilatuista antureista ja niiden käytöstä. Kaikista tilatuista antureista saatiin luotua selkeä ja yksinkertainen esimerkki sekä helposti ymmärrettävät käyttöönotto ohjeet. Tältä osin projektimme tavoitteet täyttyivät siis hyvin. Yksinkertaisien antureiden lisäksi teimme esimerkkitehtävän ja käyttöönotto-ohjeet myös Vision Subsystem v4 kameralle, mikä oli jo astetta haastavampaa.

Toissijainen tavoitteemme oli pitää demoesitys Haminassa helmikuun 2012 lopussa, mutta todellista pyrkimystä tämän tavoitteen täyttämiseen ei missään vaiheessa ollut. Demoesitys olisi vaatinut korkeampaa osaamista kuin meiltä löytyi, joten sitä ei lähdetty toteuttamaan.

Luotujen esimerkkien pohjalta voidaan projektin tuloksena pitää myös uusien projektien valmista pohjaa. Tämä projekti oli ensikosketus uusiin antureihin. Nyt kun niihin on tutustuttu, on niiden käyttömahdollisuudet lähes rajattomat. Tulevissa projekteissa antureiden toimintaan voisi esimerkiksi tutustua tarkemmin ottamalla käyttöön robotteja joissa pyörimisliikkeet ja erilaiset kiihtyvyydet ovat pääosassa, sillä nykyisissä roboteissa näitä mittaavat anturit jäävät aika vähälle käytölle. Myös tarkempi tutustuminen Vision Subsystem v4 kameraan sekä kauko-ohjauksella toimiva robotti ovat hyviä projektin aiheita.

Lähteet

Lego Mindstorms. 2012. Verkkodokumentti.

< http://mindstorms.lego.com/en-us/Default.aspx>. Luettu 19.4.2012

HiTechnic anturit. 2012. Verkkodokumentti.

<http://www.hitechnic.com/products>. Luettu 19.4.2012.

Mindsensors. 2012. Verkkodokumentti.

< http://www.mindsensors.com/>. Luettu 19.4.2012.

Metropolian Wiki. 2012. Verkkodokumentti.

< https://wiki.metropolia.fi/display/koneautomaatio/RobotC/ > Luettu 19.4.2012.

Esimerkkiohjelmat

Ir Seeker

```
#pragma config(Sensor, S1,     HTIRS2,              sensorI2CCustom)

#include "drivers/HTIRS2-driver.h" // download and include drivers


// main task
task main ()
{
  int _dirDC = 0;
  int _dirAC = 0;
    int dcS1, dcS2, dcS3, dcS4, dcS5 = 0;
    int acS1, acS2, acS3, acS4, acS5 = 0;


    for (int i = 0; i < 8; ++i)

    while (true)
    {


      // Read the current non modulated signal direction
      _dirDC = HTIRS2readDCDir(HTIRS2);
      if (_dirDC < 0)
        break; // I2C read error occurred

      // read the current modulated signal direction
      _dirAC = HTIRS2readACDir(HTIRS2);
      if (_dirAC < 0)
        break; // I2C read error occurred

      // Read the individual signal strengths of the internal sensors
      // Do this for both unmodulated (DC) and modulated signals (AC)
      if (!HTIRS2readAllDCStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5))
        break; // I2C read error occurred
      if (!HTIRS2readAllACStrength(HTIRS2, acS1, acS2, acS3, acS4, acS5 ))
        break; // I2C read error occurred

    }
  }
```

Color Sensor

```
#pragma config(Sensor, S1,     COLOR,          sensorI2CCustom)


#include "drivers/HTCS2-driver.h" //download and include drivers for robotC websites

task main () {
  int _color = 0;
  int selection=0;

  nxtDisplayCenteredTextLine(3, "WAIT A MOMENT!"); //"wait a moment" will be wroten in display for two seconds
  wait1Msec(2000);

  eraseDisplay();
  while (true)
    {
    _color = HTCS2readColor(COLOR); //_color gets values for the sensor

    if (_color < 0) //if color sensor value goes under 0, all tasks will stop.
      {           //varisensorin arvon ei pitaisi menna alle 0, jos menee tulee error
      nxtDisplayTextLine(4, "ERROR!!");
      wait1Msec(2000);
      StopAllTasks();
      }

   if(_color<5 && _color>3) // GREEN, Value from sensor = 4
    {
    selection=1;
    }

   if(_color<10 && _color>7) //RED, Value from sensor = 8
    {
    selection=2;
    }

   if(_color<7 && _color>5) //YELLOW, Value from sensor = 6
    {
    selection=3;
    }

   if(_color<3 && _color>1) //BLUE ,Value from sensor= 2
    {
    selection=4;
    }


    if(selection==1)
    {
      nxtDisplayCenteredTextLine(4, "GREEN"); //Display shows text "GREEN", 4 is a textline number
    }

    if(selection==2)
    {
      nxtDisplayCenteredTextLine(4, "RED");
    }

    if(selection==3)
    {
      nxtDisplayCenteredTextLine(4, "YELLOW");
    }

    if(selection==4)
    {
      nxtDisplayCenteredTextLine(4, "BLUE");
    }
}
}
```

EOPD Sensor

```
#pragma config(Sensor, S1, EOPD,              sensorAnalogActive)

#include "drivers/HTEOPD-driver.h" // download and include drivers

task main() {
  int raw = 0;

  int distance = 0;


    while(nNxtButtonPressed != kEnterButton) {
      // Read the raw sensor value
      raw = HTEOPDreadRaw(EOPD);

      if (raw> 70) // if raw > 70 then distance = 1
      {
        distance = 1; // distance = distance between sensor and obstacle
      }
      if (raw< 70) // if raw < 70 then distance = 2
      {
        distance = 2;
      }
      if (distance == 1)
      {
        motor[motorA]=0;
        motor[motorB]=0;

      }
      if (distance == 2)
      {
        motor[motorA]=20;
        motor[motorB]=20;

      }
    }
  }
```

Magnetic Compass

```
#pragma config(Sensor, S2,     COMPASS,  sensorI2CHiTechnicCompass)

task main()
{
  wait1Msec(2000);
  int degrees = 90;                                  // 90 degrees
  int goal = (SensorValue[COMPASS] + degrees) % 360;  // 'goal' is current compass sensor reading + degrees
                                                     // use modulus(%) 360 to keep the goal within the bounds
                                                     // of the compassSensor (1 to 360 degrees, 0 is 360).

  while(SensorValue[COMPASS] != goal)                // while the compassSensor sensor reading has not reach the goal:
  {                                                  // turn RIGHT in place
    motor[motorA] = -10;                             // set motorB to speed -10
    motor[motorB] = 10;                              // set motorB to speed 10
  }
                                                      // stop the robot:
  motor[motorA] = 0;                                  // set motorA and motorB to speed 0
  motor[motorB]  = 0;
  wait1Msec(1000);
}
```

Barometric Sensor

```
#pragma config(Sensor, S2,     HTBM,                sensorI2CCustom)

#include "drivers/HTBM-driver.h" //download and include the drivers, more in metropolia wiki

task main () {

  int pressmInHg = 0;
  float presshPa = 0.0;
  float pressPsi = 0.0;
  float tempC = 0.0;
  float tempF = 0.0;

  nxtDisplayCenteredBigTextLine(3, "WAIT");
  wait1Msec(2000);

  while (true) {
    eraseDisplay();
                                    // Read the sensor's data
        pressmInHg = HTBMreadMInHg(HTBM);   // Pressure in 1/1000th of an inch Hg
        presshPa = HTBMreadhPa(HTBM);       // Pressure in hecto Pascal
        pressPsi = HTBMreadPsi(HTBM);       // Pressure in Pounds per square inch
        tempC = HTBMreadTemp(HTBM);         // Temp in Celcius
        tempF = HTBMreadTempF(HTBM);        // Temp in Fahrenheit

    nxtDisplayTextLine(0, "%5d mInHg", pressmInHg);
    nxtDisplayTextLine(2, "%5.1f hPa", presshPa);
    nxtDisplayTextLine(4, "%5.1f PSI", pressPsi);
    nxtDisplayTextLine(6, "%3.1f C", tempC);
    nxtDisplayTextLine(7, "%3.1f F", tempF);
    wait1Msec(100);
  }
}
```

Accelometer Sensor

```
#pragma config(Sensor, S1,     ACCELOMETER,           sensorI2CCustom)


#include "drivers/HTAC-driver.h" // download and include drivers

task main () {
  int x_axis = 0;
  int y_axis = 0;
  int z_axis = 0;

  while (true) {
    eraseDisplay();

    // Read all of the axes at once
     HTACreadAllAxes(ACCELOMETER, x_axis, y_axis, z_axis);

    wait1Msec(100);
  }
}
```

Angle Sensor

```
#pragma config(Sensor, S2,     ANGLE,               sensorI2CCustom)


#include "drivers/HTANG-driver.h"

task main () {

  nxtDisplayCenteredTextLine(3, "WAIT A MOMENT!");


  wait1Msec(3000);

  eraseDisplay();
  nxtDisplayCenteredTextLine(0, "HiTechnic Angle");                       // Just graphics for the display
  nxtDisplayTextLine(1, "--------------------");
  nxtDisplayTextLine(5, "--------------------");
  while (true) {

    if (nNxtButtonPressed == kLeftButton) {                               //left button resets angle value
      HTANGresetAngle(ANGLE);
      while (nNxtButtonPressed != kNoButton) EndTimeSlice();

    } else if (nNxtButtonPressed == kRightButton) {                       //right button resets all
      HTANGresetAccumulatedAngle(ANGLE);
      while (nNxtButtonPressed != kNoButton) EndTimeSlice();
    }


    nxtDisplayTextLine(2, "Ang: %7d deg", HTANGreadAngle(ANGLE));         // Read the current angle,
    nxtDisplayTextLine(3, "Tot: %7d deg", HTANGreadAccumulatedAngle(ANGLE)); // total angle and
    nxtDisplayTextLine(4, "RPM: %7d", HTANGreadRPM(ANGLE));               // RPM and display them
    nxtDisplayTextLine(6, " <-Reset Angle");
    nxtDisplayTextLine(7, "   Reset Total->");
    wait1Msec(50);
  }
}
```

Gyroscopic Sensor

```
#pragma config(Sensor, S2,     GYRO,           sensorI2CHiTechnicGyro)

#include "drivers/HTGYRO-driver.h"

task main ()
  {
  int rotation=0;

  nxtDisplayTextLine(0, "HT Gyro");
  nxtDisplayTextLine(1, "Test 1");

  wait1Msec(2000);
  eraseDisplay();

  time1[T1] = 0;
  while(true) {
    if (time1[T1] > 1000)
      {
      time1[T1] = 0;
      }

      if(HTGYROreadRot(GYRO)<-15 && HTGYROreadRot(GYRO)>-40) //between sensor values -15 and -40
      {                                                      //will get rotation value 1
        rotation=1;
      }
      else
      {
        rotation=2;                                          //else it will be 2
      }

      if (rotation==1)                                       //when rotation value is 1 it will move
      {
        motor[motorA]=30;
        motor[motorB]=30;
      }

      if (rotation==2)                                       //when rotation value is 2, robot will stop
      {
        motor[motorA]=0;
        motor[motorB]=0;
      }
      eraseDisplay();

      nxtDisplayTextLine(1, "Reading");
                                                             // Read the current calibration offset and display it
      nxtDisplayTextLine(2, "Scale: %4d", HTGYROreadCal(GYRO));

      nxtDisplayClearTextLine(4);
                                                             // Read the current rotational speed and display it
      nxtDisplayTextLine(4, "Gyro:   %4d", HTGYROreadRot(GYRO));
      wait1Msec(10);

  }
}
```

Vision Subsystem v4 for NXT

```
#pragma config(Sensor, S4,            Kamera,           sensorI2CCustomFastSkipStates)
#pragma config(Motor,  motorA,        Oikea,        tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  motorB,        Vasen,        tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  motorC,        Ampuja,       tmotorNormal, PIDControl, encoder)

#define KohteidenMaara    3 // Kohteiden lukumaara
#define NOPEUS            20 // Etenemisnopeus
#define NOPEUS2           100 // Tykin nopeus
#define ResX              88  // NXTcamin resoluutio (Loydettavissa kameran dokumentaatiosta)
#define ResY              144
#define KpX               0.3
#define KpH               0.8
#define KohteenKorkeus    130 // Kohteen haluttu korkeus
#define Odotusaika         75

#include "rdpartyrobotcdr-v2.6/drivers/common.h"          // Sisallyttaa ajuripaketista ajurit koodiin
#include "rdpartyrobotcdr-v2.6/drivers/NXTCAM-driver.h"

void Pysayta();  // Pysayta kaikki moottorit
void Etsi();     // Etsi kohde
void Ammu();     // Ammu pallo

task main()
{
  blob_array blobs;
  memset(blobs, 0, sizeof(blob_array));
  int nblobs = 0, errX = 0, errH = 0, BlobN = 0, TargetNum = 0;
  eraseDisplay();     //Tyhjentaa robotin ruudun
  Pysayta();

  if(NXTCAMinit(Kamera))
  {
    PlaySound(soundUpwardTones);    //Mikali kamera kaynnistyy oikein, tulee soundUpwardTones aanimerkki
    while(bSoundActive);
  }
  else
  {
    PlaySound(soundDownwardTones);  //Mikali kamera ei kaynnisty oikein, tulee soundDownwardTones aanimerkki
    while(bSoundActive);
    StopAllTasks();                 //Pysayttaa kaikki toiminnot
  }
  wait1Msec(100);

  while(true)
  {
    eraseDisplay();     //Tyhjentaa robotin ruudun

      nblobs = NXTCAMgetBlobs(Kamera, blobs, true);
      nxtDisplayCenteredTextLine(1, "blobs: %d", nblobs);
      nxtDisplayCenteredTextLine(3, "Targets Down: %d", TargetNum);


if(nblobs > 0)
{
  BlobN = -1;
  for(int i = 0; i < nblobs; i++)   // Seuraa vain kohteita, joiden Korkeus > Leveys (ei "jahtaa" kaatuneita kohteita)
  {
    if((blobs[i].x2 - blobs[i].x1) < (blobs[i].y2 - blobs[i].y1))
    {
      BlobN = i;
      break;
    }
  }
  if(BlobN != -1)
  {
    errX = (blobs[BlobN].x1 + blobs[BlobN].x2) / 2 - ResX/2;  //X-suunnassa blobsin vas. reunan arvo + oik. reunan
                                                              //arvo jaettuna 2 - X:n resoluutio jaettuna 2
    errH = KohteenKorkeus - (blobs[BlobN].y2 - blobs[BlobN].y1);   //Kohteen korkeus - (y-suunnassa blobsin maksimi - minimi)

    if(abs(errX) > 4)       //Pyorimisnopeus etsiessa
    {
      motor[Vasen] = errX * KpX;
      motor[Oikea] = -errX * KpX;
    }
    else if(abs(errH) > 6)
    {
      motor[Vasen] = errH * KpH;
      motor[Oikea] = errH * KpH;
    }
    else
    {
      Pysayta();
      Ammu();
      Pysayta();
      TargetNum++;
```

```
            if(TargetNum == KohteidenMaara)  //Jos kohteiden maara on yhta kuin asetettu kohteiden tavoitemaara -> Pysayta kaikki
            {
              PlaySound(soundUpwardTones);
              while(bSoundActive);
              StopAllTasks();
            }
            else
              Etsi();
          }
        }
        else
          {
          Etsi();
          }
      }
      else
      {
        Etsi();
      }
      wait1Msec(Odotusaika);
  }
}
void Pysayta()  //Pysayttaa kaikki liikkeet
{
  motor[Vasen] = 0;
  motor[Oikea] = 0;
  motor[Ampuja] = 0;
}
void Etsi()   //Etsintatila
{
  blob_array blobs;
  memset(blobs, 0, sizeof(blob_array));
  int nblobs = 0, BlobN = -1;
  for(int i = 0; i < 1750 * 4.5 / Odotusaika; i++)
  {
    BlobN = -1;
    motor[Vasen] = NOPEUS;
    motor[Oikea] = -NOPEUS;
    nblobs = NXTCAMgetBlobs(Kamera, blobs, true);
    nxtDisplayCenteredTextLine(1, "blobs: %d", nblobs);
      if(nblobs > 0)
      {
        for(int j = 0; j < nblobs; j++)
        {
          if((blobs[j].x2 - blobs[j].x1) < (blobs[j].y2 - blobs[j].y1))
          {
            BlobN = j;
            break;
          }
        }
        if(BlobN != -1)
          break;
      }
    wait1Msec(Odotusaika);
  }
  Pysayta();

  if(BlobN == -1)
  {
    motor[Vasen] = NOPEUS;
    motor[Oikea] = NOPEUS;
    wait1Msec(2000);
  }
}
```

```
void Ammu()
{
  Pysayta();

  motor[Vasen] = -20;    //"Tahtays" eli kaantyy vasemmalle 200ms ajan
  motor[Oikea] =  20;
  wait1Msec(200);
  Pysayta();

  for(int i = 0; i < 3; i++)
  {
    PlayImmediateTone(440, 10);
      while(bSoundActive);
      wait1Msec(100);
  }
  Pysayta();
  nMotorEncoder[Ampuja] = 0;
  nMotorEncoderTarget[Ampuja] = 360;
  motor[Ampuja] = NOPEUS2;

  while(nMotorRunState[Ampuja] != runStateIdle);
  Pysayta();

  while(bSoundActive)
  {}
}
```