Robottiohjelmoinnin harjoitustyö

Ristinollarobotti

Aaro Salosensaari
aaro.salosensaari@cs.helsinki.fi

Tietojenkäsittelytieteen laitos
Helsingin Yliopisto

Helsinki, 11. tammikuuta 2015

Sisältö

Tämä on joulun 2014 robottikurssin loppuraportti. Luettavin versio on pdf.

1 Ristinollarobotin kuvaus

Ristinollarobotti on ristinollaa web-kameran avulla pelaava Lego Mindstorms -robotti.

Robotti koostuu varsinaisesta piirtorobotista (`PenBot`) ja erillisestä tietokoneella (= kannettava tietokone) ajettavasta varsinaisesta peliohjelmasta (`BotGame`). Ohjelmointikieli on molemmissa Java (piirtobotissa LeJOS), ja kuvantunnistukseen käytetään OpenCV:n Java-API:ia.

Peliohjelma osaa tunnistaa ulkoisen web-kameran avulla paperille piirretyn pelilaudan ja pelaajien (robotti ja sen vastustaja) sille piirtämät merkit, ja tämän avulla pelata ristinollaa ihmisvastustajaa vastaan. Peliohjelma lähettää tekoälyn valitsemat siirrot Bluetoothin yli piirtorobotille, jolla on valmiit rutiinit ruksin piirtämiseksi kuhunkin ruutuun.

2 Robotin rakenne

2.1 Materiaalit ja tarvikkeet

Robotin toteuttamiseen käytettiin

- (hieman vajaa) Lego Mindstorms NXT -sarja
  - NXT Brick, kolme moottoria
  - sekalaisia Mindstorms -legoja
  - (NXT:n omia sensoreita ei tarvittu / käytetty)
- Web-kamera: Logitech C270
- Tussikynä: Stabilo Pen 68
- Kannettava tietokone jossa Bluetooth
- Kuminauhaa, paperia, teippiä, korotettu alusta kameralle

2.2 Piirtobotti

Piirtobotin perusrunko perustuu NXTPrograms.com 3-Motor Chassis -rakenteeseen, jota jouduttiin hieman muokkaamaan osien puutteesta johtuen (esim. samanlaista rullapyörää ei ollut käytettävissä, joten piti soveltaa) ja johon lisättiin ylös ja alas liikkuva kynä.

Rakenteeltaan robotti on kahden moottorin avulla liikkuva auto, joka pystyy kääntymään pyörittämällä vasenta ja oikeaa moottoria eri nopeuksilla. Pyörittämällä moottoreita samalla nopeudella eri suuntiin robotti pystyy kääntymään paikoillaan renkaiden välisen kuvaannollisen 'akselin' keskipisteen ympäri.

Koska ns. rullapyöräksi sopivia pieniä renkaita ei allekirjoittaneelle päätyneessä sarjassa ollut mukana ja käytettävissä olevien suurehkojen kumipyörien sijoittaminen perusrunkoon osoittautui varsin haastavaksi (ilman kumiosaa pyörien liike oli liian tökkivää), robotissa ei ole tukena tavanomaista rullapyörää (*castor wheel*), vaan yksinkertainen pyödän pintaa pitkin liukuva tuki.

Kynän liikuttelumekanismin toiminnan ymmärtänee parhaiten oheisista kuvista. Käytännössä kynä on kiinnitetty kumilenkillä telineeseen, jonka liike on rajoitettu ylös-alas -suuntaiseksi kiskojen avulla. Lisäksi kynän edessä ja takana on rajoittavat tuet jotka pitävät sen asennon vakaasti paikoillaan piirtämisen aikana.

Kumilenkkikiinnitys mahdollistaa teoriassa pienen hätävaran siltä varalta että käyttäjä menee ja poistaa turvarajat PenBotin ohjelmakoodista ja kalibroi moottorin väärin väärin: jos moottori yrittäisi painaa kynää alemmas kuin turvalliseen käyttöön on suunniteltu, kumilenkit teoriassa joustaisivat sen sijaan että moottoriin tai rakenteeseen kohdistuisi haitallista rasitusta. (Robotin toiminta voidaan myös välittömästi keskeyttää hätäseis-nappulaa painamalla.)

Lisäksi kynän vierässä on pieni työkalu, joka helpottaa robotin asettamista oikeaan suuntaan ruutupaperin päälle.

Varsinaisen erillisen rakennusohjeen sijasta lukijaa pyydetään seuraamaan NXT-Programs.com:n ohjeen vaiheita 1 – 4 ja 12 –, ja vertailemaan eroavaisuuksien kohdalla alla oleviin kuviin. Huom. erityisesti että rullapyörän korvaavan tuen kiinnitys on erilainen.

2.3 Kuvia

TODO kuvat tähän

2.4 Strategiset mitat

- Renkaiden välinen etäisyys ~120 mm
- Etäisyys keulasta perään ~210 mm
- Säkäkorkeus ~75 mm
- Kynän kärjen etäisyys akselista piirtoasennossa ~70 mm

3 Ohjelmakoodi

3.1 Penbot

3.2 BotGame

3.2.1 Kuvantunnistuksen toimintaidea

Kuvantunnistusmenetelmän pääinspiraationa oli AI Shackin Sudoku-lukija, jota tosin on sovellettu varsin paljon. Menetelmän idea on yleisellä tasolla seuraava:

1. Ensin etsitään taustakuva, johon mahdollisia muutoksia verrataan:
2. Muunnetaan kuva harmaasävykuvaksi.
3. Ruutupaperin ruutujen häivyttämiseksi sumennetaan kuvaa Gauss-sumennoksella, jonka jälkeen tehdään harmaasävykuvasta mustavalkoinen muuttamalla (adaptive threshold), jolloin kuvaan jää jäljelle vain pääasiassa merkitseviä viivoja ja merkkejä. Tämän 'binäärikuvan' värit käännetään jatkoa varten.
4. Aiemmassa vaiheessa jotkut tärkeätkin ruudukon viivat saattavat 'katketa', joten niitä yritetään palauttaa morphologisella sulkemisella (morphological closing).
5. Näin käsitellystä kuvasta etsitään Hough-muunnoksella kaikki viivat.
6. Koska OpenCV:n Hough-rutiini löytää sellaisilla parametreilla joilla varmasti saadaan kaikki *tärkeät* viivat myös *paljon* viivoja jokaista pelilaudan oikeaa viivaa kohti, lähellä toisiaan olevien viivojen parvet yhdistetään yhdeksi viivaksi per parvi (keskiarvo).
7. Yhdistetyistä viivoista etsitään äärimmäiset (tietyn marginaalin puitteissa) vaaka- ja pystyviivat, jotka vastaavat pelilaudan reunoja. Näiden leikkauspisteet (= peliruudukon nurkat) lasketaan.
8. Leikkauspisteiden avulla kuvan perspektiivi korjataan ja se jaetaan 3x3 -ruudukoksi. Kunkin ruudun reunoista 'leikataan pois' pieni kaistale (jotka sisältävät piirretyn ruudukon viivat) ja (alkutilanteessa tyhjä) sisäalue ja sen histogrammi talletetaan.
9. Jokaiselle verrattavalle kuvalle tehdään sama prosessi, ja kuvien vastaavia alueille verrataan toisiinsa. Mikäli jonkin solun histogrammeissa peruskuvan ja verrattavan välillä on suuri ero, todetaan että tähän ruutuun on verrattavassa kuvassa piirretty uusi merkki.
10. Mikäli havaittu merkki hyväksytään oikein luetuksi siirroksi, se päivitetään uudeksi peruskuvaksi seuraavan siirron lukemista varten.

TODO Kuvia laudan hahmottamisesta.

4 Testaus

Kaikkea mitä olisi voinut testata, ei tullut testattua. Erityisesti matematiikkametodeja ja kuvanhahmotustoimintaa varten olisi voinut kirjoittaa suoranaisia yksikkötestejä.

Käytännössä robotin kehittäminen oli iteratiivinen prosessi: "testataan toimiiko jokin toiminnallisuus näin" -> "korjataan kunnes toimii" -> "kun toimii, lisätään toiminnallisuus". Valitettavasti tälläisestä epä-TDD 'patternista' ei jäänyt hirveästi varsinaista testikoodia.

4.1 Ohjelmallisia testi'skriptejä'

Eri toiminnallisuuksien kokeilemista ja säätöä varten on ohjelmissa erityisen `test`-paketin luokissa muutama `main`-metodeja, joita ajamalla varsinaisen `main`:n sijaan voi testata robotin eri toiminnallisuuksien toimintaa.

4.1.1 PenBotin kynänliikuttelun testaus ja säätö

`PenBot`: n `test.PenConfigureTest` sisältää testin kynän kalibrointirutiinille, jonka avulla voi kokeilla yleisesti kynänliikuttelun toimivuutta ("liikkuuko kynä oikein? piirtääkö se jäljen paperille?").

Lisäksi `PenConfigureTest` mahdollisti piirtobotin kynänliikuttelun ohjelmallisten turvarajojen testaamisen. Tulos: turvarajat toimivat, kalibrointiskriptin (= normaali käyttö) avulla bottia ei saatu kääntämään kynämoottoria yli 20 astetta alaspäin, joka oli todettu vielä täysin turvalliseksi asennoksi.

4.1.2 Kuvananalysointitoiminnallisuuden testaus ja säätö

`BotGame`:n `test.BoardReaderCamTest`:ia voi käyttää web-kameran kuvankaappauksen toiminnan testaamiseen käynnistämättä varsinaista pelirutiinia. Esimerkiksi tarkistin ennen demotilaisuutta että kuvankäsittelymetodit toimivat myös yliopiston tilojen valaistuksessa.

Vastaavankaltaista koodinpätkää käytettiin merkintunnistustoiminnallisuutta koodatessa myös eri raja-arvojen ja kuvaruutujen vertailumenetelmien tutkimiseen. Lopulta päädyttiin koodin tämänhetkisessä versiossa oleviin vakioihin ja metodeihin.

4.1.3 Käskyjen välittäminen tietokoneelta PenBotille ja piirtäminen

Bluetooth-kommunikaation testaamista varten on erillinen komentoriviohjelma `BotCommander`, jonka avulla käyttäjä voi suoraan komentoriviltä käskyttämällä lähettää viestintäprotokollan mukaisia komentoja `PenBot`:ille.

`BotCommanderin` avulla tehtiin seuraavat testit:

1. Bluetooth-yhteyden muodostaminen ja `Input/OutputStream`:n avaaminen `PenBotin` ja `BotCommanderin` välillä onnistuu.
2. `PenBot` vastaanottaa ja lukee Bluetoothin kautta lähetettyjä käskybittejä onnistuneesti.
3. `PenBot` suorittaa käskyn mukaisen komennon oikein (piirtää ruksin oikeaan koordinaattiin).

4.2 Testibotit

Varsinaisen `PenBot`-ohjelman lisäksi jäljelle jäi pari pientä 'testibottia' jotka voitiin myös ladata Lego-robotin brickille LeJOS:n yms. eri ominaisuuksien testaamiseksi.

4.2.1 Hello Ironman!

Käytettiin testaamaan toimiiko yksinkertaisen "hello world" -ohjelman kääntäminnen ja lataaminen robottiin eri ympäristöissä ja yhteysmenetelmillä. Jouduttiin mm. toteamaan että Ubuntun ajureilla ei saanut toimivaa USB-yhteyttä Lego-robottiin.

4.2.2 Hello BT!

Testattiin viestibittien vastaanottamisen lisäksi myös lähettämistä robotilta tietokoneelle, mutta tätä ominaisuutta ei sitten varsinaisessa pelirobotin toteutuksessa hyödynnetty.

4.3 Muita testiskenaarioita

4.3.1 Hätäpysäytys

`PenBot`:iin asetettua vaadittua hatäpysäytystoiminnallisuutta testattiin painamalla kesken ohjelman suorituksen pysäytysnapiksi valittua `ESCAPE`-nappulaa. Hätäpysäytys toimi.

5 Rajoitukset ja tulevaisuus

5.1 Toteuttamatta jääneet ominaisuudet

Robotti jäi kahdelta osin hieman keskeneräiseksi:

Ensinnäkin, peli ei osaa pelata ristinollaa täysin itsenäisesti, sillä `BotGame`:n hahmontunnistuskoodi ei osaa hylätä sellaisia webkameran kuvia, joissa pelilaudan ja

kameran välissä on este (esimerkiksi ihmispelaajan käsi tai piirtorobotti itse). Ohjelmasta puuttuu myös botin piirtämien merkkien tunnistus kameran kuvasta AI:n omiksi siirroiksi. Tämän vuoksi peli pyytää käyttäjältä vahvistuksen jokaiselle pelilaudalla havaitulle muutokselle esittääkö se botin tai pelaajan tekemää siirtoa.

Toiseksi varsinainen tekoäly puuttuu. Robotti pelaa ristinollaa sääntöjen mukaan, muttei erityisen älykkäästi.

Molemmat ongelmat olisi ollut tarkoitus ratkaista: Ristinollatekoälyn toteutus jonkinlaisella minimax-algoritmilla tai alpha-beta -karsinnalla olisi melko triviaali tehtävä. Kameran eteen tulleen esteen kaltaiset huomattavat muutokset kuvassa puolestaan olisi (ainakin teoriassa) yksinkertaista havaita OpenCV:n avulla.

Esimerkiksi eräs vaihtoehto tähän olisi tarkastella värikuvan histogrammin poikkeamia (pelilautaa esittävään kuvaan nähden) kun laudan päällä on robotin tai käden kaltainen 'ylimääräinen' esine. Oletettavasti histogrammissa nähtäisiin suuri poikkeama verrattuna tilanteeseen, jossa ainoa muutos on peliruutuun ilmestynyt pieni merkki.

Botti-tekoälyn tekemät siirrot vuorostaan olisi luultavasti mahdollista tunnistaa (ja ohittaa kysymättä pelaajan vahvistusta) hieman ohjelmakoodia laajentamalla.

5.2 Muita puutteita ja rajoitteita:

Piirtorobotti ei osaa asemoida itseään pelilautaan nähden. Käyttäjän on sijoitettava robotti ennaltavalittuun asentoon peliruudukkoon nähden (pelilaudan lävistäjän kautta kulkevalle suoralle). Mikäli robotti on hieman vinossa, se myös ajaa hieman sivuun ja pahimmassa tapauksessa piirtää ristejä vääriin paikkoihin. Ongelmaa voisi hieman helpottaa laajentamalla nykyistä toiminnallisuutta pienellä kalibrointiskriptillä (robotti kulkisi edestakaisin ja piirtäisi pisteitä sinne missä se kuvittelee esim. peliruudukon nurkkien olevan; käyttäjä voisi korjata robotin asentoa).

Piirtorobotti nykyisessä muodossaan piirtää kulkemalla edestakaisin ja pyörimällä moottoriakselinsa keskipisteen suhteen; toisin sanoen vaakasuuntainen viiva on kaareva. Tämän vuoksi mahdolliset kuviot käytännössä ovat ruksien ja pisteiden kaltaisia yksinkertaisia kuvioita, joita tämä rajoitus ei haittaa. Ristinollan perinteisen 'nolla'-kuvion piirtäminen olisi nykyisellä rakenteella melko vaikeaa.

Kuvan analysointimenetelmät ovat teoriatasolla yleistettävissä, mutta koodissa käytetyt vakiot, raja-arvot, jne. on löydetty käsin kokeilemalla tietynlaisella kamerakokoonpanolla. Esimerkiksi huomasin että tutkittavien kuvien resoluution vaihtaminen voi rikkoa nykyisen toiminnallisuuden.

6 Käyttöohjeet

6.1 Ohjelmistojen kääntäminen

6.1.1 Penbot

6.1.2 BotGame

6.2 Pelaaminen