KTH

Parprogrammering med legorobot

Lego Mindstorm NXT i BrixCC

Oscar Mattsson
2015-08-27
oscarmat@kth.se
Introdata II0310

Sammanfattning

Att på ett effektivt sätt kunna felsöka i och rätta till kod är en viktig egenskap hos en programmerare. I denna laboration behandlas installation av utvecklingsverktyg och drivrutiner samt felsökning och rättning av kod genom parprogrammering och testning.

Verktygen som användes för att uppnå detta var en robot av typen Lego Mindstorm NXT samt ett felaktigt program skrivet i NXC. Detta program redigerades i utvecklingsmiljön BrixCC. Tillsammans med en labbpartner programmerades denna robot till ett givet beteende. Vad vi lärde oss utav denna laboration var att felsöka, angripa ett problem i taget, analysera det, diskutera det och lösa det tillsammans.

Innehållsförteckning

1. Inledning

Ett felaktigt program skrivet i NXC för en Lego Mindstorm-robot används som utgångspunkt i denna laboration som ska justeras genom parprogrammering och testning. Uppgiften är att rätta till programmet så att roboten kan följa en bana av svart tejp till dess att den åker in i en vägg där den stannar och spelar upp ljud.

1.1 Bakgrund

Parprogrammering är ett sätt att upptäcka fel i och rätta till kod mer effektivt än en person kan göra själv. Med ett extra par ögon går både kodning och testning snabbare och blir mer noggrant genomfört. Att kunna felsöka och rätta till kod på ett ingenjörsmässigt sätt är viktigt för att kunna leverera ett väl fungerande program inom rimlig tid.

1.2 Syfte och målsättning

Syftet med laborationen är att introducera parprogrammering som koncept samt att få en grundläggande förståelse för hur felsökning i kod går till, hur vi genom testning hittar fel och hur vi rättar till dem.

2. Genomförande

Först laddades drivrutiner och utvecklingsmiljön BrixCC ned samt installerades på datorn som användes för laborationen. Sedan anslöts roboten via USB till datorn. Vid uppstart av BricxCC valdes USB som input.

Filen linefollower.nxc som innehöll det felaktiga programmet öppnades upp i BrixCC. Till att börja med satt jag framför datorn och min labbpartner satt bredvid och observerade. Vid kompilering av programmet dök det upp fel direkt på rad 34 (se resultat för logg över ändringar). Vi felsökte genom att gå till den angivna raden för felet där vi såg att en funktion fick input av fel typ. Efter att vi hade rättat till detta kunde programmet kompileras och överföras till roboten.

Roboten kopplades loss från datorn, placerades på golvet och programmet kördes från robotens styrenhet. Roboten körde en liten ”dans” där den åkte runt i en cirkel innan den började köra framåt. Under testningen observerades även att roboten drog åt höger istället för att köra rakt fram.

Efter detta bytte jag och min labbpartner plats så att han skrev kod och jag observerade. Vi letade rätt på koden som fick roboten att dansa som visade sig vara ett funktionsanrop i main-funktionen. Vi kommenterade ut denna kod och testade igen. Denna gång började roboten köra framåt utan att dansa. Vi fortsatte byta plats efter varje test vi körde under resten av laborationen.

Att roboten drog åt höger berodde på ett kod-stycke som justerade hastigheten på varje hjul för att svänga beroende av värdet på en ljussensor. Vi fick hjälp av labbassistenten för att förstå vad olika värden på ljussensorn betydde. Hur roboten svängde justerades sedan i koden i ett par iterationer av kodning och test på ett sådant sätt att roboten till slut kunde följa den svarta linjen.

Efter att vi var färdiga med redigeringen av koden och programmet var redovisat för labbassistenten raderades programmet från roboten genom knapparna på styrenheten.

3. Resultat

Komplett lista av ändringar i linefollower.nxc:

Rad	Gammal Kod	Ny Kod	Kommentar
34	int groupMembers[] …	string groupMembers[] …	Fel typ
35	1,	”Oscar”	
36	2	”Andreas”	
46	… (8*i – 16) …	… (8*i) …	Feljusterad text på skärm efter körning
76	… SensorRaw(IN_1);	… SensorRaw(IN_3);	Fel input ljussensor
92	OnFwd(OUT_A, SpeedSlow);	OnFwd(OUT_A, SpeedFast);	Justera svängriktning
100	OnFwd(OUT_B, SpeedFast);	OnFwd(OUT_B, SpeedSlow);	Justera svängriktning
115	dance();	//dance();	Omotiverad dans

Efter dessa ändringar i koden kunde roboten följa en kurva av svart tejp oavsett riktning fram till en vägg där den stannade, spelade upp en melodi och sedan visade våra namn på skärmen.

4. Analys

För det mesta gick det smidigt att hitta och rätta till fel. Vissa fel var lite mer diskreta än andra i början, såsom att koden som läste av ljussensorn läste på fel port och att namnen skrevs ut fel på skärmen. Men eftersom vi var noggranna med att undersöka portarna på roboten och se vad som hände när vi testade så lyckades vi upptäcka dessa fel utan större svårighet.

Där vi fick mest problem var när vi skulle förstå vad värdena på ljussensorn betydde. Där fick vi hjälp från en labbassistent som beskrev att högre värden på sensorn innebar tejp (mörkare) och lägre värden innebar golv (ljusare). Det fanns även ett intervall i koden med ”magiska nummer” som innebar att man var på tejpkanten där de individuella hjulens hastighet kunde justeras. Där kunde vi lista ut att vi ville ha samma värden på höger och vänster hjul inom det intervallet så att roboten skulle åka rakt fram på tejpkanten. Med motsatta värden utanför intervallet fungerade det så att roboten kunde svänga för att hålla sig på tejpkanten när vi testade programmet.

Trots att denna del var den som tog mest tid och var krångligast att förstå gick det rätt så fort att lösa efter att vi hade förstått hur ljussensorn fungerade. Resten av ändringarna i programmet gick väldigt enkelt så fort vi hade upptäckt vart det var fel.

5. Diskussion

Parprogrammeringen tyckte jag gick bra. Jag tyckte inte att det var så stor skillnad om man var den som satt vid tangentbordet eller inte, men att kunna diskutera koden och reflektera över möjliga lösningar tillsammans kändes användbart. Jag tror det gäller allt typ av skapande. Att kunna bredda

sitt eget perspektiv och sina kunskaper genom att diskutera med en annan person känns väldigt värdefullt och är något jag tror att jag kommer att ha användning för i både studier och i yrkeslivet.

Processen för att rätta till programmet gick väldigt smidigt den också. Att ofta variera mellan att koda och att testa samt att fokusera på en sak i taget känns som ett bra förhållningssätt när man kodar. I just den här laborationen kändes detta rätt naturligt då det inte var så stora ändringar som skulle göras. Men jag kan tänka mig att detta förhållningssätt är ännu viktigare i större projekt där det är mer som ska ändras, mer som kan gå fel och framförallt mycket mer som ska hållas koll på.

Vad gäller programvaran så kändes gränssnittet i BrixCC rätt så föråldrat jämfört med andra utvecklingsmiljöer. En sak så enkelt som att kunna scrolla ordentligt var rätt så besvärligt. Men förutom det så verkade det finnas en del kraftfulla verktyg för att läsa av sensorer som var kopplade till roboten eller för att exempelvis styra roboten direkt från datorn för att testa motorerna på olika inputs. Att ansluta sig via USB till roboten och föra över programmet gick väldigt smidigt. Såg ut som att roboten även hade stöd för Bluetooth, såg inget om det i BrixCC, men smidigt om det finns flera sätt att ansluta sig på. Språket i sig har jag inga klagomål på. Med hjälp av de verktyg som finns inbyggda i BrixCC samt dokumentationen för NXC online gick det rätt så enkelt att använda sig av NXC.

Referenser

Labb-PM: https://bilda.kth.se/courseId/12708/content.do?id=23767349
NXC Programmer's Guide: http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html

Bilagor

Dagboksinlägg på KTH Social: