Felsökning och test av ett NXC-program för en LEGO Mindstorms-robot

Linnea Kylén, <lkylen@kth.se>

3 september 2013

II1310, Introduktionskurs i datateknik
ICT, KTH

Sammanfattning

En ny årskull av nyantagna studenter har anlänt på ICT-skolan, KTH. Med anledning av detta hölls en introduktionskurs i datateknik för att introducera programmering och ingenjörsarbete på ett kul sätt. Kursens huvudmoment var en laboration där studenterna felsökte och testade ett program i språket NXC. Målet med laborationen var att få programmet att fungera, vilket innebar att man skulle få en robot att utföra vissa uppgifter.

Programmet Brixc Command Center användes för att redigera och kompilera koden och sedan ladda ner den till roboten för körning av programmet. Ett protokoll fördes över de förändringar som gjordes i koden. Studenterna jobbade i grupper om två personer och använde parprogrammering som arbetssätt. Resultatet var att roboten efter förändring av ett antal kodrader utförde sina uppgifter på ett korrekt sett. Analysen tyder på att ännu effektivare felsökning hade kunnat göras om gruppen hade lagt mer fokus på programmets flöde. Den funktion som orsakade mest problem (`dance()`-funktionen) störde nämligen det övriga felsökandet genom att få roboten att svänga i början av programmet. I diskussionen menar författaren att uppgiften var lärorik men att parprogrammering som arbetssätt inte kom till sin fulla nytta i denna uppgift då det var mer "observerande" än skrivande av kod.

Innehåll

1 Inledning **3**
1.1 Bakgrund 3
1.2 Syfte och målsättning 3

2 Genomförande **3**

3 Resultat **4**

4 Analys **5**

5 Diskussion **6**

6 Referenser **7**

7 Bilagor **8**

1 Inledning

Denna laboration gick ut på att att felsöka och testa ett program i språket NXC. Programmet kontrollerade en LEGO Mindstorms-robot, och målet var att denna skulle utföra vissa uppgifter på ett korrekt sätt. Syftet med detta var att träna de nyanlända studenterna på ICT-skolan, KTH, i ingenjörsarbete samt att ge dem en intressant introduktion till programmering.

1.1 Bakgrund

Hos de nyanlända studenterna på ICT-skolan fanns varierande kunskaper i programmering. För att ge en intressant introduktion till programmering genomfördes en laboration där studenterna fick hjälpas åt att två och två lösa en uppgift. Speciellt låg fokuset i uppgiften på att felsöka ett program. Detta för att skapa förståelse för hur de olika bitarna i koden samverkar samt att visa hur enkla fel kan ge oväntade beteenden vid körning av ett program.[2] Laborationen var även en övning i problemlösning och rapportskrivning som är viktiga förmågor för att lyckas som ingenjör.

1.2 Syfte och målsättning

Syftet med laborationen var att på ett spännande och enkelt sätt introducera programmering. Studenterna fick vid laborationen träna på felsökning och testning som är viktigt för effektivt programmeringsarbete. Uppgiftens syfte var också att introducera studenterna till ingenjörsarbete och rapportskrivning samt att skapa förståelse för den datormiljö och de informationssystem som används på ICT-skolan.[2] Syftet med rapportskrivning är verifiering, trovärdighet, tidseffektivitet och generell dokumentation. Detta för att man utifrån rapporten ska kunna upprepa försöket på samma sätt, för att man ska kunna sätta sin tillit till resultatet och för att förhindra att andra i onödan ska behöva göra om samma försök eller misstag (redundant arbete).[3]

Personligen var mitt syfte med laborationen att ha roligt och att absorbera så mycket kunskap som möjligt som kan vara till nytta i kommande kurser och ingenjörsarbete. Ett personligt mål med rapporten var att prova att göra ett dokument i LaTeX.

Studenterna tilldelades ett färdigt program i språket NXC. Målet var att få detta program att fungera, vilket innebar att en Lego Mindstorms-Robot skulle utföra vissa uppgifter.[2] Dessa bestod i att roboten skulle följa en svart halvcirkelformad linje på golvet. När roboten slog i väggen så skulle studenternas namn skrivas ut efter varandra på skärmen. Uppgiften skulle lösas genom parprogrammering i grupper om två personer.

2 Genomförande

För att genomföra laborationen krävdes ett antal förberedelser. Det första steget var att läsa igenom Labb-PM som fanns på kurshemsidan i Bilda. Där framgick syftet, målet, uppgiften, vilket material som behövdes samt examinationskrav.[2]

Som kursmaterial fanns programmet Bricx Command Center (som ZIP-arkiv) som laddades ner och installerades (på en bärbar Windows-dator) för att kunna redigera och kompilera koden och sedan ladda ner den till roboten för körning av programmet. Det fanns även en drivrutin till LEGO NXT som krävdes för att datorn skulle kunna hitta roboten. Även denna laddades ner och installerades. Utöver detta fanns en tutorial kring hur man programmerar Lego NXT robotar med hjälp av NXC [1], en länk till API:et för NXC [6] samt en monteringsspecifikation (PNG-bild) som visade hur roboten skulle vara monterad.[2] Inför laborationstillfället läste vi igenom tutorialen, bekantade oss med API:et samt studerade monteringsspecifikationen.

Under "Kursmaterial" på kurshemsidan fanns även en rapportmall (i tre filformat) samt det program i NXC som skulle användas vid laborationen (`linefollower.nxc`). Även dessa laddades ner (rapportmallen endast i .doc-format [5]).

Vid laborationstillfället fick vi en LEGO Mindstorms-robot samt en USB-kabel som användes till att överföra programmet från datorn till roboten. Vi kontrollerade att LEGO Mindstorms-roboten var korrekt monterad enligt moneringsspecifikationen samt kopplade in den till datorn med USB-kabeln. Bricx Command Center användes sedan till att granska och redigera koden i `linefollower.nxc`. Till detta använde vi arbetssättet parprogrammerering i intervaller om 20 minuter, dvs en person kodade ("driver") och en person observerade och utvärderade kontinuerligt det som den andra skrev ("observer") [4]. På detta vis gick vi metodiskt igenom koden i NXC-programmet och sökte efter fel samt försökte korrigera dessa så att roboten skulle utföra uppgifterna på ett korrekt sätt. Till vår hjälp använde vi tutorialen och API:et för att bättre förstå vad som skedde i koden.

Vi testkörde även koden efter varje ändring för att se hur roboten betedde sig. Då kompilerade vi programmet och laddade ner det till roboten med USB-kabeln. Sedan kopplade vi loss roboten och placerade den på den halvcirkelformade svarta (tejp)linjen som började och slutade vid väggen. Vi kunde därefter starta programmet från menyn på roboten och utvärdera dess beteende. Ett protokoll fördes över alla förändringar som vi gjorde i koden (Figur 2 på sidan 9).

När uppgiften var löst skrev vi varsitt dagboksinlägg på KTH Social (se Figur 1 på sidan 8) samt varsin labbrapport. Jag skrev min labbrapport i LaTeX och tog hjälp av rapportmallen[5] för att strukturera skrivandet.

3 Resultat

Laborationen resulterade i bifogade kodförändringar (se Figur 2 på sidan 9). Då vi hade gjort dessa förändringar betedde sig roboten som den skulle enligt uppgiften, dvs den följde linjen och skrev ut våra namn efter varandra när den slog i väggen.

Först ändrade vi typen på listan `groupMembers[]` till `string` (sträng) och skrev vi in våra namn istället för siffrorna `1` och `2` (på raderna 34–36). Sedan ändrade

vi `names` i `ArrayLen(names)` på rad 44 till `groupMembers`. På rad 46 ändrade vi `TextOut(0, (LCD_LINE2 - (8*i-16)), names[i])` genom att ta bort `-16` så att texten skrevs ut på rätt rader. Vid testkörning efter dessa förändringar kunde roboten inte följa linjen men skriva ut namnen när den slog i väggen.

Vidare så ändrade vi rad 76 så att den tog sin input från rätt sensor (ljussensorn som var monterad mot `IN_3`). Detta gav ingen för oss märkbar förändring vid testkörning. Vi noterade dock att roboten hoppade till mystiskt efter ett antal sekunder.

Sedan korrigerade vi if-satserna på rad 90–101 i `followLine()`-aktiviteten genom att sätta `SpeedSlow` på rad 92 till `SpeedFast` och `SpeedFast` på rad 100 till `SpeedSlow`. Inte heller detta hade någon större inverkan vid testkörning.

Slutligen kommenterade vi ut `dance()`-funktionen på rad 115. Nu följde roboten linjen vid testkörning samt skrev ut namnen vid kollison.

4 Analys

Den första deluppgiften, att få namnen att skrivas ut vid kollision med väggen, uppnådde vi snabbt eftersom vi förstod att typen på `groupMembers[]` skulle ändras till sträng och att funktionen `printNamesToScreen()` behövde skriva på rätt rader. Vi hade därefter lite svårare att förstå robotens beteende, då den verkade bete sig ungefär likadant oavsett vad vi ändrade. Vi förstod först inte varför den svängde redan vid start.

När vi upptäckte att roboten använde fel port för att läsa av ljussensorn (som var kopplad till `IN_3` ej `IN_1`) trodde vi att vi skulle få ett genombrott. Den forsatte dock att svänga vid start. Efter ett tag märkte vi att roboten hoppade till efter att ha kört runt några sekunder. Vi började leta mer strukturerat efter vad som hände efter start och som fick roboten att svänga. Då såg vi att `dance()`-funktionen i `main()`-aktiviteten var orsaken till svängandet. När vi kommenterade bort den så hade vi redan gjort tillräckligt med förändringar (t ex korrigeringen av if-satserna i `followLine()`-aktiviteten) för att allt skulle fungera som det skulle.

Generellt gick laborationen bra eftersom det gick ut på att felsöka programmet. Vi kunde dock ha sparat lite tid om vi tidigare hade tittat på i vilken ordning saker skedde, framför allt genom att studera `main()`-aktiviteten där `dance()`-funktionen kördes innan `readTouchSensors()`- och `followLine()`-aktiviteterna hade startat.

`readTouchSensors()`-aktiviteten ansvarade för att skriva ut namnen och spela en trudelutt när roboten kolliderade med väggen. `followLine()`-aktiviteten ansvarade för läsa av ljussensorn (värdet lagrades i variabeln `lightIntensity`) och svänga (med hjälp av if-satser och ett tröskelvärde) så att linjen följdes. Roboten svängde genom att öka hastigheten på den ena motorn tills `lightIntensity` var tillbaka på tröskelvärdet. `followLine()`-aktiviteten stängde också av motorerna och bröt (med `break`) sin while-loop när `readTouchSensors()`-aktiviteten

meddelade att den var klar (i.e. roboten hade kolliderat).

En utav raderna som vi förändrade hade inte behövt förändras (rad 44). Eftersom funktionen `printNamesToScreen()` tog in en lista som argument behövde rad 44 inte referera direkt till listan `groupMembers[]` i `ArrayLen(names)`. Snarare så blev koden där sämre av vår förändring eftersom `printNamesToScreen()`-funktionen nu inte skulle kunna användas korrekt på en annan lista med ett annat antal element.

5 Diskussion

Resultatet av laborationen var att roboten följde linjen och skrev ut våra namn när den slog i väggen. Målet att få programmet att fungera var därmed uppfyllt. Syftena att ge en rolig introduktion till programmering och att låta studenterna öva på felsökning och programmering uppnåddes också då det var kul och givande att programmera i par och att se resultatet i robotens beteende. Det var också tillfredställande när roboten till sist följde linjen efter en stunds intensiv felsökning och testning. Jag uppnådde även mina personliga syften och mål att ha kul, absorbera ny kunskap samt prova att skriva en rapport i LaTeX.

Det som orsakade mest problem för oss var `dance()`-funktionen. Eftersom den fick roboten att alltid svänga i några sekunder i början av körningen gjorde den det svårt att felsöka resten av koden då vi inte förstod varför våra förändringar inte fick roboten att bete sig annorlunda. Det var ett smart litet hyss från lärarnas sida. Från det har jag lärt mig att vara mer noggrann med att förstå programmets flöde när jag felsöker ett program som någon annan skrivit. Det är en bra lärdom för framtida programmeringsprojekt, såväl i skolan som i en framtida yrkesroll.

Programmet Brixc Command Center fungerade smidigt och var lätt att förstå. När det gäller NXC som språk så var strukturen lätt att förstå men jag tycker att uttryck som `OnFwd()` skulle kunna vara av mer beskrivande karaktär, t ex `TurnOnForward()`. Det skulle göra koden mer lättläst vilket sparar tid och minskar risken för fel.

Den lösningsmetod som vi använde var framförallt att diskutera fram vilka förändringar som borde göras och sedan implementera och testa om dessa gav önskat resultat. Vi gjorde små förändringar mellan varje testkörning för att lättare kunna se hur det påverkade (eller inte påverkade) robotens beteende. Detta var ett mycket tacksamt sätt att arbeta på då vi hade något så pass visuellt tydligt som en svängande robot.

Parprogrammeringen kom inte till sin fulla nytta då uppgiften främst innebar felsökning och testning. Detta innebar att det inte var så mycket skrivande utan mest läsande av koden (i princip två "observers"). Det vore intressant att prova detta arbetssätt när man programmerar mer från början eller lägger till funktionalitet. Dock så var det meningsfullt och lärorikt att vara två personer eftersom man kunde diskutera vad olika delar av koden innebar och få nytta av den sammanlagda programmeringskunskapen.

6 Referenser

[1] Daniele Benedettelli (with revisions by John Hansen), *Programming LEGO NXT Robots using NXC (beta 30 or higher)*, Version 2.2, June 7, 2007.

[2] Labb-PM, *II1310 Introduktionskurs i datateknik*, HT2013.

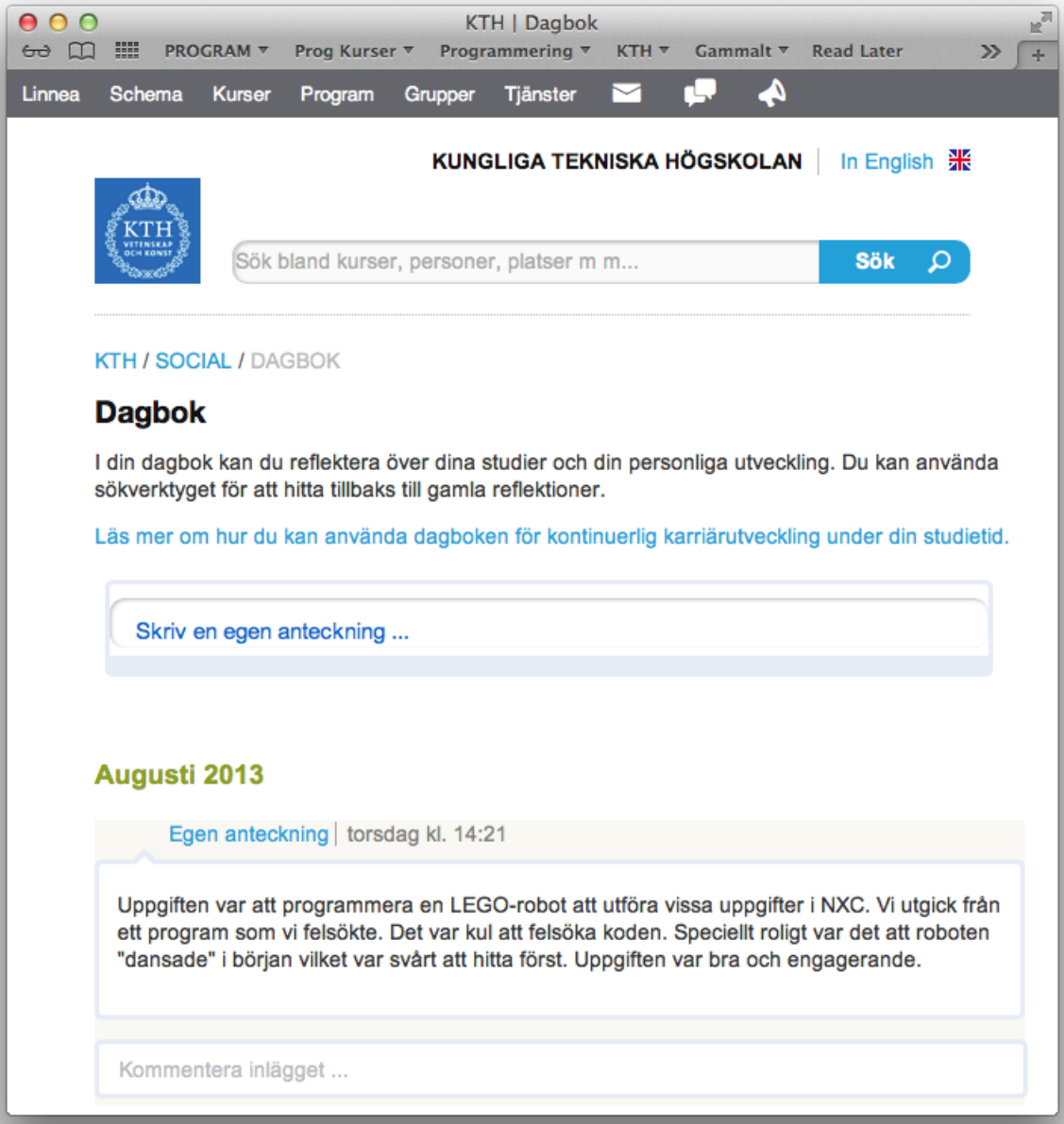
[3] Föreläsning 2.pdf, *II1310 Introduktionskurs i datateknik*, HT2013.

[4] Föreläsning 3.pdf, *II1310 Introduktionskurs i datateknik*, HT2013.

[5] Rapportmall.doc, *II1310 Introduktionskurs i datateknik*, HT2013.

[6] NXC Programmer's guide (API), `http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html`, besökt 29 augusti 2013.

7 Bilagor



Figur 1: Dagboksinlägg på KTH Social (skrevs vid laborationstillfället).

Kommentarsblad för laborationsuppgift i II1310 Introduktionskurs i datateknik

Radnummer	Ny kod	Kommentar
34-36	string " Henrik" " Linnea"	int → string
44	groupMembers	names → groupMembers ↑ loopar listan med namn
46	tar bort -16 i TextOut	skriver på rätt rader
76	IN_3	IN_1 → IN_3 (SensorRaw) ljussensor
92	Speed Fast	out A svänga
100	speed Slow	out B ej svänga
115	//	kommentera ut dance-funktionen

Figur 2: Protokoll över kodförändringar.