Felsökning i NXC-kod

Anton Kallin
05 – september – 2012
antonkal@kth.se
Introduktionskurs i Datateknik, II1310

Sammanfattning

Laborationen omfattar en NXC-fil gjord för att få en NXT-robot att följa en svart linje in i en vägg, som behöver en reform för att kunna utföra sitt uppdrag. Problemet löses genom att noggrant läsa igenom koden och bestämma sig för vilka delar som innehåller fel, anteckna de ändringarna man gör i koden och testa att köra koden efter att ha kompilerat den och fört över till roboten med hjälp av BricxCC och lite annan mjukvara. Felkällorna innehåller allt ifrån värdelösa rutiner och knasiga värden till orimlig konstruktion och anrop. Kan parprogrammering ge både gamla och nya kodare något nytt att tänka på i framtiden? Oundvikligt måste jag svara ja på den frågan. Beträffande programmerings-språket och mjukvaran så jobbar jag hellre med något annat än NXC.

Innehållsförteckning:

1. Inledning :

I laborationen har vi fått ett stycke kod gjord för att få LEGO NXT Robotar att följa en svart linje på golvet som leder mot en vägg (med hjälp av en ljus-sensor), sedan stanna när den åker in i väggen och till sist skriva ut gruppmedlemmarnas namn på skärmen; men koden innehåller en del programmeringsfel som istället gör att roboten åker runt i en cirkel på golvet. Uppgiften går ut på att hitta vad som egentligen orsakar roboten att inte följa linjen på golvet genom att verkligen gå in i hur koden fungerar.
1.1 beskriver vad detta egentligen har att göra med en ingenjörs karriär, och i 1.2 beskrivs vad man egentligen bör lära sig ifrån den.

1.1 Bakgrund:

Felsökning i programmerings-kod är lika vanligt som att kor producerar mjölk. Uppgiften lär studenter att även små enkla fel i koden kan ge upphov till alldeles absurda resultat, och den skall även bidra till att de lär sig ett slags noggrannhetstänkande när de går igenom en annans kod för att verkligen förstå hur den fungerar.

1.2 Syfte och målsättning:

Laborationen lägger mycket fokus på att introducera dem som inte har programmerat tidigare hur man specifikt instruerar en dator att ta hand om ett problem, förfriska gamla programmerares sinnen och introducera båda parterna till parprogrammering. Uppgiften skall även göra en bekant med arbetsgången vid ingenjörsarbete och även arbetsvanan för utbildning på ICT-skolan med IT-system.

2. Genomförande:

Innan laborationen kan utföras behövs det en Windows-dator med installerad mjukvara som kommer användas när robotarna programmeras. I denna uppgift kommer vi jobba med ”Bricx Command Center”, vilket är programmet vi kommer använda för att redigera källkoden samt kompilera koden och flasha roboten, och drivrutinen ”Fantom Drive” så BricxCC kan hitta Mindstorms-robotarna. Båda får vi tag på från nätet, i det här fallet kursmaterialsidan på Bilda[1] och LEGO Mindstorms egna hemsida[2], och båda är relativt enkla att installera (BricxCC:s material packas upp på valfritt ställe och Fantom Drive har en egen installations-fil). När dessa är installerade så kan vi börja med laborationsuppgiften.

Eftersom laborationen också går ut på att parprogrammera har vi en programmerare och en observatör. Programmeraren har makt att ändra i koden när de i gruppen har kommit fram till vad som måste göras eller när ena parten har kommit på vad som kan vara felkällan, och observatörens jobb är att se till att programmeraren inte kommer för långt före genom att kontrollera det programmeraren gör och säga till när han/hon inte hänger med i resonomanget. Programmerare och observatör byter plats var 20:e minut så båda kan testa på NXC och, igen, se till så båda förstår vad som händer.

Först måste koden granskas och båda parter måste komma överens om vilka rutiner som tar prio över andra, det är även enklast att jobba med en rutin i taget så kommentera gärna bort de metoder som kommer i vägen för testerna. När paret är överens om vilken rutin som verkar vara hörnstenen

av problemet så gäller det att verkligen hålla reda på vad som händer i den. Genom att försöka beskriva den för sig själva och lista ut hur koden egentligen är byggd för att lösa problemet; definiera koden med egna ord, d.v.s. se problemen på ett annat sätt. T.ex. istället för att säga if(x < 0 || y>f) så beskriver vi den som: ”Om x är mindre än 0, eller y är större än f…”. Utmaningen här är att lista ut vad han/hon som skrev koden tänkte när denne delade upp problemet.

När man kommit underfund med vad som egentligen borde hända i programmet och vilka ändringar som bör göras gäller det även att hålla koll på hur koden kommer att ändras härnäst via anteckningar. Skriv ner vilket radnummer du ändrar på, vad den nya raden innehåller och kommentera detta helst så man verkligen förstår vad som gjorts (ta helst en backup på den gamla koden, det skadar aldrig). Om man ska ta bort hela rader så bör man kommentera bort dem istället för att ta bort dem helt; då har man nämligen kvar samma radnummer som tidigare och det är enkelt att införa raden igen om man tänkte fel.

När man tycker koden ser bra ut och vill testköra den med hjälp av roboten måste programmet kompileras och laddas ner till den via BricxCC. Därefter kan man placera roboten vid änden av linjen och starta programmet. Om det misslyckas så går man tillbaka till koden och ser efter om man verkligen gjort rätt eller gjort problemet ännu värre. Om allt går som man vill och roboten utför det den ska så är det rekommenderat att göra några tester till så man försäkrar sig om att den verkligen kan utföra sitt uppdrag eller om man bara hade tur första gången.

Om koden fortfarande fungerar som den ska så gäller det att testa igen, denna gång genom att ta fram hur koden var från början och därefter göra de ändringar som man gjorde, och förhoppningsvis antecknat att man gjorde, i den gamla. Detta är en sorts ”failsafe” för att se till att man verkligen höll reda på allt man gjorde under kodningen och så andra kan få fram samma resultat via ens anteckningar. Om detta går utan problem, och en expert eller lärare kan konfirmera att man gjort rätt, så har du klarat av labben.

Du går igenom 4 steg konstant när du försöker hitta ett fel i programmering:

1. Skumläs koden och kom överens i vilken rutin felet befinner sig
2. Kolla noggrant efter luckor, försök översätta koden med egna ord om du inte förstår
3. Ta anteckningar på de ändringar du gör i koden (igen, ta gärna backup också)
4. Kompilera och testa koden

Om resultatet inte är det man letar efter går man tillbaka till steg 1 och försöker igen.

3. Resultat:

Radnummer	Ny kod	Kommentar
1	#define SpeedSlow=50;	Ändrar hastigheter så de blir mer anpassade till kurvan
2	#define SpeedFast = 70;	Samma som rad 1
35	"Anton","Emmy"	Gruppmedlemmarnas namn
45	(8*i)	Namnen ska inte förskjutas 16 enheter åt sidan

68	lightIntensity=SensorRaw(IN_3);	Se till så ljussensorn har rätt input
84	OnFwd(OUT_A, SpeedFast);	Måste svänga vänster, lägger mer kraft på högra hjulet
92	OnFwd(OUT_B, SpeedSlow);	Befinner sig på linjen, fortsätt långsamt farmåt
97	/* void dance()	Kommentera bort dance eftersom den inte används
102	}*/	Samma som rad 97
104	// dance();	Kommentera bort anropet till dance

4. Analys:

Det är rätt så uppenbart att rutinen followLine() har hand om robotens rörelse, men det gäller som sagt att klura ut vad koden egentligen försöker säga. I det här fallet så har vi fem variabler som ligger bakom followLine():

1. Top- och BotThreshold
2. SpeedSlow och SpeedFast
3. lightIntensity

Ljus-sensorn ger uppehov till lightIntensity och denna jämförs med Threshold i raderna 82 → 93. Om ljuset är för starkt så är har roboten kommit utanför linjen och måste svänga genom att köra fortare med ena hjulet medan det andra kör långsamt. Så länge lightIntensity håller sig mellan Top- och BotThreshold så rör sig roboten framåt.

Förutom followLine() så innehåller printNamesToScreen() ett fel där den förskjuter texten 16 enheter åt sidan (ett tecken = en enhet), om detta tas bort så kommer båda namnen att synas på skärmen. För bekvämlighetens skull så tog vi även bort dance() helt tidigt när vi förstod att den inte hade något med programmet att göra.

Generellt är followLine() helt beroende av ljus-sensorn, men är också lika beroende av Top- och BotThreshold. Om laborationen skulle göras om,och vi hade en grön linje istället för en svart så skulle problemet inte vara lika simpelt som tidigare (eller om hela golvet var en annan mörkare färg, som lila eller grå). Linjens bredd är också en hinder i followLine(). Om t.ex. den var ännu smalare skulle roboten ha ännu svårare att följa efter om man inte tar nya hastighetsvärden eller helt enkelt gör roboten mer flexibel och kapabel att göra skarpare kurvor.

5. Diskussion:

Denna laboration är ett väldigt bra exempel på vad som är mycket frustrerande för många: att problemet är mycket enklare än vad man tror. Direkt när jag kikade på koden innan labben så ville jag göra om followLine() helt bara så jag kunde tyda den bättre och eller få fram en egen lösning till problemet; detta må ha låtit bra då, men är en enorm omväg med tanke på hur enkel lösningen egentligen är.

Något som jag verkligen tyckte om var att parprogrammera med en som inte hade programmerat alls tidigare. Det tvingade mig att förklara för henne varför jag tyckte några ändringar var nödvändiga och

det hindrade även mig själv från att göra något väldigt dum dritsigt val. Jag har aldrig riktigt programmerat sida vid sida med någon förut och jag skulle ärligt talat inte ha något emot att göra det igen.

Generellt när det kommer till robotarna så är jag väldigt besviken. Det känns udda att LEGO har annonserat Mindstorms och sina andra robot-produkter som ett sätt vem som helst kan komma in i programmering med ska vara så begränsat till maskiner som innehåller dess drivrutiner. Installationsprocesserna av mjukvaran bör enkelt kunna fås tag på när man kopplar in robotarna till datorn istället för att ladda ner mycket mjukvara. BricxCC kan jag förstå eftersom det var en hel editor som aldrig krävde en exe-fil, men den andra mjukvaran borde det inte behöva vara så mycket problem med att installera, oavsett operativ-system eller prestanda. Ingen känner för att släpa på ett par installationsfiler överallt, menar jag.

Referenser:

1. https://bilda.kth.se/courseId/8498/content.do?id=19121762 (Kursmaterialsida på Bilda, kräver KTH login)
2. http://mindstorms.lego.com/en-us/Support/Files/Driver.aspx (”FantomDriver”-drivrutin till LEGO NXT)
3. https://bilda.kth.se/courseId/8498/content.do?id=19150198 (Labb-PM, kräver KTH login)

Bilagor:

Skärmdump på mitt dagboksinlägget på KTH Social (resencion av laborationen och kursen):

Känns som att kursen behövde mer övningar än genomgångar så folk verkligen kunde testa på programmering innan labben. Att bara prata om programmering är enkelt, men det tar ett tag att verkligen vänja sig vid att verkligen göra det (psuedo-kod VS. programmerings språk, leaning by doing, o.s.v).

Quizen i Bilda verkade vara lite mer trial-&-error istället för något som testar din kunskap om KTH:s miljö och känns inte som ett riktigt bra sätt att få redo på KTH:s uppbyggnad och miljö när man bara kunde söka efter många av svaren på internet. Jag tror det är en dålig idé att kräva 100% rätt från det när du kan köra om så många gånger du vill; du kan rent av testa dig fram till svaren genom att testa alla möjligheter utan någon sorts bestraffning eller varning. Givetvis fanns nästan alla av svaren i PDF-filerna som användes på genomgångarna, fast det blir bara mer ord-sökning för de som bara vill bli av med quizet.

Dock så var nästan inget av momenten direkt onödiga. Genomgångarna var bra upplagda (om lite uttråkande ibland) och informerar det mesta en behöver och det är en väldigt "fail-safe" kurs som så gott som ingen kan misslyckas med.

Även om kursen kan ha introducerat folk till kursen ännu bättre med mera övningar och tid, så har det aldrig känts som ett slöseri med tid (speciellt inte labben); även de som redan känner till programmering och datateknik kan åtminstone få någonting från denna kurs, och att para ihop dem med de som knappt vet något om datateknik tvingar dem att förklara det dom gör till sina gruppmedlemmar och klasskamrater (vilket även upplyser dem själva om de gör fel).

linefollower.nxc – Koden som vi arbetade med under labben

```
1 //Definierar hastigheterna som motorerna går i
2 #define SpeedSlow 80
3 #define SpeedFast 100
4
5 //Gränsvärden för ljussensorn (ändra ej dessa)
6 #define TopThreshold 630
7 #define BotThreshold 600
8
9 //Definierar raderna på robotens skärm
10 #define LCD_LINE1 56
11 #define LCD_LINE2 48
12 #define LCD_LINE3 40
13 #define LCD_LINE4 32
14 #define LCD_LINE5 24
15 #define LCD_LINE6 16
16 #define LCD_LINE7 8
17 #define LCD_LINE8 0
18
19 int lightIntensity;
20 bool finished = false;
21
22 //En liten trudelutt
23 Tone done[] = {
```

```
24 TONE_C4, MS_50,
25 TONE_E4, MS_50,
26 TONE_G4, MS_50,
27 TONE_C5, MS_50,
28 TONE_E5, MS_50,
29 TONE_G5, MS_50,
30 TONE_C6, MS_200
31 };
32
33 //Fyll i namnen på gruppmedlemmarna i listan
34 string groupMembers[] = {
35 "person1"
36 };
37
38 //Skriv ut namnen på skärmen
39 void printNamesToScreen(string names[])
40 {
41 TextOut(0, LCD_LINE1, "Gruppmedlemmar:");
42 int i;
43 for(i = 0; i < ArrayLen(names); i++) /* Loopar igenom listan med
namn */
44 {
45 TextOut(0, (LCD_LINE2 - (8*i-16)), names[i]);
46 }
47 }
48
49 //Läs av värden från tryckknapparna
50 task readTouchSensors()
51 {
52 while(true)
53 {
54 if(Sensor(IN_1) || Sensor(IN_4))
55 {
56 finished = true;
57 PlayTones(done); /* Spela den lilla trudelutten */
58 printNamesToScreen(groupMembers);
59 Wait(SEC_20);
60 abort();
61 }
62 }
63 }
64
65 //Läs av värdet från ljussensorn
66 void readLightSensor()
67 {
68 lightIntensity = SensorRaw(IN_1);
69 }
70
71 //Följ linjen!
72 task followLine()
73 {
74 while(true)
75 {
```

```
76 if(finished == true)
77 {
78 Off(OUT_AB);
79 break;
80 }
81 readLightSensor();
82 if(lightIntensity < TopThreshold)
83 {
84 OnFwd(OUT_A, SpeedSlow);
85 } else {
86 OnFwd(OUT_A, SpeedSlow);
87 }
88 if(lightIntensity > BotThreshold)
89 {
90 OnFwd(OUT_B, SpeedFast);
91 } else {
92 OnFwd(OUT_B, SpeedFast);
93 }
94 }
95 }
96
97 void dance()
98 {
99 OnFwd(OUT_A, 87);
100 OnFwd(OUT_B, 20);
101 Wait(SEC_3);
102 }
103
104 //Main-aktiviteten (körs alltid först!)
105 task main()
106 {
107 Precedes(readTouchSensors, followLine);
108 SetSensorType(IN_1, IN_TYPE_SWITCH);
109 SetSensorMode(IN_1, IN_MODE_BOOLEAN);
110 SetSensorType(IN_3, IN_TYPE_LIGHT_ACTIVE);
111 SetSensorMode(IN_3, IN_MODE_RAW);
112 SetSensorType(IN_4, IN_TYPE_SWITCH);
113 SetSensorMode(IN_4, IN_MODE_BOOLEAN);
114 dance();
115 OnFwd(OUT_AB, SpeedSlow);
116 }
```