Marc Ilsø Poulsen, Peter Bartram Hansen og Bjarne Antonsen 8/11-2010
dLego: Lab Session 8 Rapport
Varighed: 2 ½ time

Formål 1:

Formålet var at anvende en færdig implementation af flere ”konkurrerende” opførselsmønstre på den samme NXT enhed, og have en form for prioritering imellem dem vha. ”subsumption” mekanismer som beskrevet af Rodney Brooks.

Formål 2:

Formålet var desuden at udvide den færdige implementation med et ekstra opførselsmønster, som skulle bevæge robotten i retning af lyskilder.

Plan:

Der kunne ikke planlægges så meget til den første del af øvelserne, da vi her blot skulle tage den udleverede kode (Sound Car programmet) og så gøre os nogle observationer undervejs. I forbindelse med den anden halvdel af øvelserne skulle vi overveje hvordan det nye opførselsmønster bedst blev tilføjet til den eksisterende implementation (hvor i hierarkiet den nye opførsel gav mest mening i forhold til prioritering). Vi skulle desuden finde en fornuftig threshold værdi for hvornår lystiltrækningen skulle træde i kraft.

Resultater:

Ved at lade robotten køre med den udleverede kode kunne vi observere dens opførsel. Vi kunne forholdsvist let identificere hvilken opførsel hos robotten som hørte til hvilket lag.

RandomDrive laget er det lavest prioriterede lag, som udføres hele tiden så længe ingen af de højere prioriterede lag overtager robotten. Det bevæger robotten i små, korte ryk i forskellige vilkårlige retninger. Af koden ses det at bevægelserne udføres med tilfældigt 1,5-2,5 sekunders mellemrum:

```
delay((int)(1500+1000*Math.random()));
```

AvoidFront laget er højere prioriteret end RandomDrive laget, da det skal sikre at robotten ikke støder ind i objekter i dens tilfældige bevægelser. Derfor skal dette lag overtage robotten så snart UltraSonicSensor'ens måling viser at et objekt er for tæt på robottens front

```
private final int tooCloseThreshold = 20; // cm
```

Vi kunne uden problemer se betydningen af dette lag ved at stikke en hånd ind foran UltraSonicSensor sensoren, hvorefter vi kunne observere at robotten bakker lidt efterfulgt af et 90 graders højresving for at slippe fri fra forhindringen.

PlaySounds er et simpelt lag, som overtager robotten hvert 10. sekund. Det eneste laget gør er at spille en vidunderlig række af tilfældige toner, hvorefter koden sættes til at vente i 10 sekunder før laget igen overtager robotten.

Under udførslen skrives der diverse informationer på NXT enhedens LCD display. Informationerne vises på denne måde:

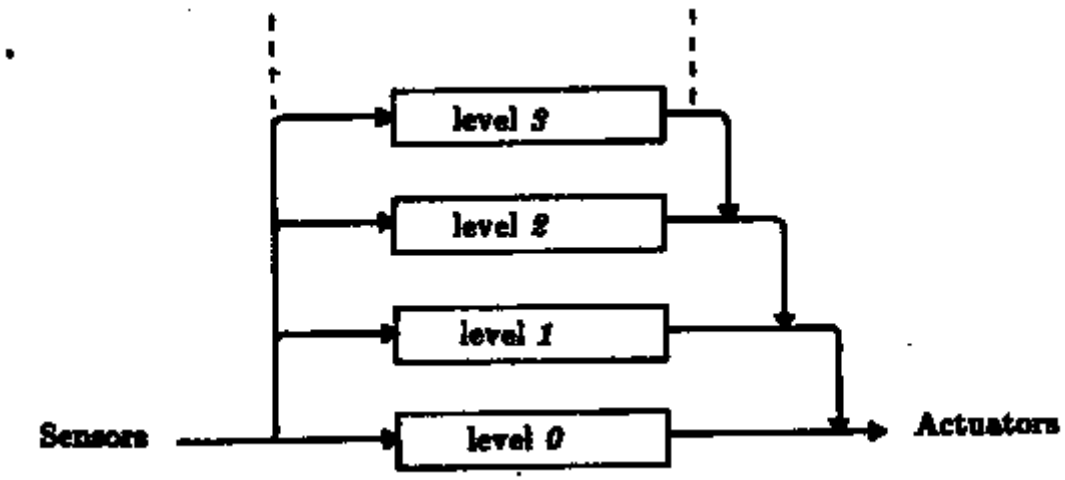
Marc Ilsø Poulsen, Peter Bartram Hansen og Bjarne Antonsen 8/11-2010
dLego: Lab Session 8 Rapport
Varighed: 2 ½ time

```
Sound Car
Drive 0/1       f/s
Avoid 0/1  255 f/b/s
Play  0/1       s
```

Første kolonne viser blot navnene på de tre funktionslag, Drive, Avoid og Play. Den anden kolonne viser om variablen suppressCount er 0 eller 1 i det Behavior objekt, som hører til det givne lag (Forklaring af denne mekanisme følger nedenfor). Tredje kolonne viser hele tiden sample værdien fra UltraSonicSensor'en fra AvoidFront laget, så vi kan følge med i målingerne under kørslen. Den fjerde og sidste kolonne viser hvilke kommandoer de forskellige lag giver til motorerne i øjeblikket (f=forward, b=backward og s=stopped).

Subsumption mekanismen:

Den hierarkiske opdeling af lagene fungerer vha. en ”subsumption” mekanisme (Se diagram nedenfor). Mekanismen fungerer ved at gøre højere prioritere lag i stand til at undertrykke (suppress) alle de underliggende lag. Suppression mekanismen er rekursiv, så i denne implementation har PlaySounds tilknyttet AvoidFront laget, som det lag det skal undertrykke. Når AvoidFront undertrykkes af PlaySounds så undertrykker AvoidFront selv videre på RandomDrive laget. Et lag anvender suppress() metoden, når det vil tjekke om det kan få lov at overtage robotten. Hvis et ovenliggende lag har undertrykt de underliggende lag så er denne metode blokerende. For at frigive robotten anvender lagene release() metoden. Da dette er implementeret uden nogen former for interrupt mekanisme, så betyder det at så snart et lag har fået lov til at slippe forbi suppress() kaldet, så vil alle kommandoerne i dette lag blive udført før et andet lag kan komme til.



Tilføjelse af lystiltrækning:

Vi fandt det mest indlysende at det nye lag skulle ligge imellem RandomDrive og AvoidFront, da det skal have højere prioritet at blive tiltrukket af lyskilder i forhold til blot at køre tilfældigt rundt, men samtidig skal det have højere prioritet ikke at støde ind i ting når robotten er på vej mod lyskilden. Vi tog udgangspunkt i vores implementation af Braitenberg robotten fra sidste øvelsesgang, som blev tiltrukket af lyskilder på samme måde som vi nu ønsker denne robot skal blive tiltrukket. Derfor var vores kode næsten allerede skrevet, men der skulle et par små tilretninger til. Vi syntes ikke det var nødvendigt at anvende proportionel hastighed på motorerne

ift. lysniveauet fra lyssensorerne i denne implementation. Så vi anvendte en simpel test med de to lyssensorer til at definere en fornuftig threshold værdi for hvornår lys-laget skal overtage robotten (lightThreshold i koden nedenfor).

Selve control-loopet i LightSpeed klassen følger implementationen af control-loopet i AvoidFront klassen. Vi har derfor en while-løkke øverst, som hele tiden sampler lyssensorerne og tjekker værdierne ift. lightThreshold. Vi bliver i denne while-løkke så længe lyset foran begge sensorerne ikke er kraftigt nok. Så snart lyset foran en af sensorerne bliver kraftigt nok, så kalder vi suppress() metoden, hvorefter vi sætter robotten til at køre enten til højre eller venstre ved at sætte motorkraften til enten 75/100 eller 100/75. Til sidst kalder vi release() metoden for at frigive robotten til de andre lag.

LightSpeed.java:

```
import lejos.nxt.LCD;
import lejos.nxt.SensorPort;
import lejos.nxt.addon.RCXLightSensor;


public class LightSpeed extends Behavior {

	RCXLightSensor ls1, ls2;
	int lightThreshold = 40;

	public LightSpeed(String name, int LCDrow, Behavior subsumedBehavior) {
		super(name, LCDrow, subsumedBehavior);
		ls1 = new RCXLightSensor(SensorPort.S1);
		ls2 = new RCXLightSensor(SensorPort.S4);
	}

	public void run()
	{
		while (true)
		{
			int value1 = ls1.readValue();
			int value2 = ls2.readValue();

			while (value1 < lightThreshold && value2 < lightThreshold)
			{
				value1 = ls1.readValue();
				value2 = ls2.readValue();
			}

			suppress();

			if (value1 > value2) {forward(75, 100); drawString("r");}
			else if (value2 > value1) {forward(100, 75); drawString("l");}
			else {forward(100, 100); drawString("f");}

			delay(1000);

			stopBehavior();
			drawString("s");

			delay(500);
```

```
                    drawString(" ");

                    release();
                }
        }

}
```

Konklusion:

Vi har opnået vores mål med denne øvelse. Vi har fået afprøvet hvordan flere vidt forskellige opførselsmønstre kan implementeres på den samme NXT enhed. Tilsammen skaber prioriteringen af disse lag en ønsket opførsel, som robotten skal udvise. I dette tilfælde var det i første omgang via tre lag af opførsel som blev til fire ved tilføjelsen af vores eget lystiltrækningslag.