Marc Ilsø Poulsen, Peter Bartram Hansen og Bjarne Antonsen 7/1-2011
dLego: Projekt Session 2 Rapport
Varighed: 5 timer

Formål:

Formålet for i dag var først at optimere fjernstyringen af vores robot, da vi ikke var helt tilfredse med præcisionen i vores første implementation, hvormed robotten kunne styres via et GUI.

Dernæst ønskede vi at komme i gang med at tilføje subsumption arkitekturen til vores robot, hvor fjernstyringen af robotten skulle indgå som en selvstændig opførsel sammen med i første omgang en undvige-opførsel til automatisk at undvige objekter foran robotten samt en touch-opførsel.

Plan:

Vores umiddelbare plan for optimeringen af fjernstyringen var i første omgang blot at kigge på forskellige muligheder for, hvordan vi får størst mulig kontrol over at dreje robotten. Det var især denne del, som vi ikke var tilfredse med.

Med hensyn til tilføjelsen af subsumption arkitekturen, så vidste vi at det ville kræve en omstrukturering af vores hidtidige kode, da vi indtil nu havde skrevet al robot-koden i én klasse, RCRobot.java.

Vi skulle desuden have tilføjet en UltraSonicSensor og en TouchSensor til vores robot, førend vi ville kunne begynde at lave disse nye opførsels-lag i vores subsumption hierarki.

Resultater:

Optimering af fjernstyringen:

Det helt store problem med vores første implementation af at dreje robotten, var manglen på fleksibilitet. Et tryk på højre eller venstre knappen i vores GUI resulterede altid i en 90-graders drejning af robotten med eller imod urets retning. Graden af drejningen af robotten ønskede vi at have mere kontrol over via vores GUI.

I den første implementation sendte vi en dreje-kommando til robotten ved click events på højre og venstre knapperne. Hvis vi skulle opnå mere kontrol over robotten ved fortsat at anvende click events som tidspunktet for at sende dreje-kommandoer til robotten, så ville det betyde at robotten skulle dreje et minimalt antal grader, for eksempel 5 grader, ved hver dreje-kommando. Dette ville kun kræve en meget lille ændring i vores kode. Ulempen ved denne løsning ville dog være, at man skulle klikke adskillige gange på højre eller venstre knapperne, hvis man ville have robotten til at dreje meget.

Det stod klart for os, at vi skulle anvende mousePressed sammen med mouseReleased events til at opnå større kontrol med drejning af robotten. Vi endte med en ganske simpel løsning, hvor vi i mousePressed for venstre/højre knapperne blot sender en enkelt kommando til robotten:

(LegoBlueToothGUI.java)

```
public void mousePressed(MouseEvent arg0) {
	try
		{
			if(dir.equals("left")) {
				dos.writeInt(3);
				dos.flush();
			}
			if(dir.equals("right")) {
				dos.writeInt(4);
				dos.flush();
			}
		}
		catch(IOException ex){}
	}
}
```

Ved modtagelsen af disse kommandoer starter robotten en vilkårlig stor rotation om sig selv til enten venstre (3) eller højre (4):

(RemoteControl.java)

```
while(true) {

...

switch (command) {
case LEFT:
	pilot.rotate(99999, true);
	break;
case RIGHT:
	pilot.rotate(-99999, true);
	break;
}
```

Når enten højre eller venstre knappen i vores GUI slippes igen, så sender vi en stop-kommando afsted til robotten, som hos robotten bruges til at stoppe alt bevægelse (via TachoPilot.stop() metoden):

(LegoBluetoothGUI.java)

```
public void mouseReleased(MouseEvent arg0) {
	try {
		if (dir.equals("left") || dir.equals("right")) {
			dos.writeInt(0);
			dos.flush();
		}
	} catch (IOException e) {}
}
```

For at undgå inkonsistens imellem fjernstyringen via højre/venstre og fremad/bagud knapperne, så lavede vi fremad/bagud knapperne om på samme måde som højre/venstre knapperne, så disse også anvendte mousePressed og mouseReleased events i stedet for mouseClicked events. Vi følte nu, at vi havde ganske stor kontrol over robottens bevægelser via fjernstyringen.

Tilføjelse af subsumption arkitekturen:

Det var forholdsvist problemfrit at tilføje sumbsumption arkitekturen for lag-delt styring af robotten, da vi var bekendte med denne funktionalitet fra kursets lab session 8. Herfra kunne vi genbruge Behavior.java klassen som super-klasse for alle vores ønskede opførselslag. Den eneste nødvendige ændring til Behavior klassen var at ændre til anvendelse af TachoPilot klassen i stedet for den Car klasse, som Behavior klassen oprindeligt anvendte til styringen af robottens motorer.

Vi ville nu afkoble selve fjernstyringsdelen fra RCRobot, så det i stedet kom til at indgå som et opførselslag. Koden for dette placerede vi i en ny klasse, RemoteControl.java, som nedarvede fra Behavior klassen. Det eneste der er tilføjet i forhold til da koden var placeret i RCRobot klassen er kaldene til suppress() og release() metoderne til brug i subsumption mekanismen imellem de forskellige opførselslag. Klassen ser således ud:

(RemoteControl.java)

```
public class RemoteControl extends Behavior {

	... //Various constants

	public RemoteControl(String name, int LCDrow, Behavior subsumedBehavior) {
		super(name, LCDrow, subsumedBehavior);
	}

	public void run() {
		pilot.setMoveSpeed(100);
		pilot.setTurnSpeed(200);

		LCD.drawString("Waiting",0,0);
		NXTConnection conn = Bluetooth.waitForConnection();
		LCD.drawString("Connected",0,0);
		DataInputStream is = conn.openDataInputStream();

		LCD.drawString("Data: ", 0, 2);

		while(true) {

			try {
				suppress();

				...

			} catch (IOException e) {}
			finally {
				release();
			}
		}
	}
}
```

Vi lavede herefter to simple opførselslag, AvoidFront.java og TouchBehavior.java, som henholdsvis skulle anvende en UltraSonicSensor og en TouchSensor til at foretage målinger for hvornår laget skulle træde i kraft og udføre dets formål.

Idéen med AvoidFront var, at det ved hjælp af målinger fra UltraSonicSensoren skulle sørge for, at robotten undviger, hvis robottens front kommer for tæt på et objekt. Dette gør vi ved at prædefinere en konstant tærskelværdi for den afstand, som vi ønsker robotten altid som minimum skal holde til objektet foran sig. I control-loopet samples UltraSonicSensoren hele tiden med ganske små mellemrum, og hvis den målte afstand er lavere end den fastsatte værdi træder laget i kraft og udfører sin undvigelsesmanøvre. I vores tilfælde består denne manøvre blot af, at vi lader robotten bakke med en vis hastighed i 1 sekund, hvorefter kontrollen igen frigives til RemoteControl laget.

Hele control-loopet for udførelsen af det ovenstående ser således ud:

(AvoidFront.java)

```
while (true)
{
	int distance = us.getDistance();
	while ( distance > tooCloseThreshold )
	{
		distance = us.getDistance();
		drawInt(distance);
	}

	suppress();

	backward(70,70);
	drawString("b");
	delay(1000);

	stopBehavior();
	drawString("s");

	drawString(" ");

	release();
}
```

TouchBehavior laget anvender en TouchSensor, som vi har monteret bagpå vores robot. Da vi ønsker, at det skal være muligt for en robot i denne lille ”leg” at køre hen og trykke på touch-sensoren hos en anden robot, så måtte vi have gang i vores kreative Lego-evner for at få touchsensoren monteret tilpas langt nok bagude på robottens bagside, da den helst ikke skal kollidere med AvoidFront opførslen. (Da vi indtil nu kun har arbejdet med én robot, så har vi endnu ikke afprøvet om dette rent faktisk er tilfældet...)

Vi er endnu ikke sikre på hvad TouchBehavior laget i sidste ende skal gøre, når der trykkes på touchsensoren. Derfor er det eneste laget indtil videre gør blot at stoppe alt bevægelse hos robotten i 10 sekunder, hvorefter robotten afgiver et beep for at tilkendegive for omverdenen at kontrollen af robotten (i form af RemoteControl og AvoidFront lagene) igen er blevet frigivet.

Hele control-loopet for udførelsen af det ovenstående ser således ud:

(TouchBehavior.java)

```
while (true) {
	if (touch.isPressed()) {
		suppress();

		try {
			pilot.stop();
			sleep(10000);
			Sound.beep();
		} catch (InterruptedException e) {}

		finally {
			release();
		}
	}
}
```

Hovedklassen som binder alle opførsels-lagene i sumbsumption arkitekturen er den nu reducerede RCRobot.java klasse. Her instantieres de forskellige Behavior subklasser og relationerne imellem dem defineres af den tredje parameter i konstruktøren. Hvert opførsels-lag kender kun til det underliggende lag, som det har ansvaret for at undertrykke hver gang det selv skal udføre noget. Hele klassen ser således ud:

(RCRobot.java)

```
public class RCRobot {
	public static void main(String[] args) {

		AvoidFront avoid;
		RemoteControl rc;
		TouchBehavior touch;

		rc = new RemoteControl("Remote", 4, null);
		avoid = new AvoidFront("Avoid", 5, rc);
		touch = new TouchBehavior("Touch", 6, avoid);

		rc.start();
		avoid.start();
		touch.start();

		while(!Button.ESCAPE.isPressed()) {
			rc.reportState();
			avoid.reportState();
			touch.reportState();
		}
	}
}
```

Ved afprøvningen af, om RemoteControl laget blev undertrykt af AvoidFront laget og om både AvoidFront og RemoteControl lagene blev undertrykt af TouchBehavior laget så viste det sig, at vi efter tryk på touchsensoren stadigvæk var i stand til at fjernstyre robotten i de 10 sekunder, hvor touch laget burde undertrykke al bevægelse af robotten. Det mærkelige var, at AvoidFront laget rigtig nok blev undertrykt af TouchBehavior laget, men RemoteControl laget blev af en eller anden grund ikke undertrykt. Vi var dog løbet tør for tid denne dag, så vi havde ikke mulighed for at finde problemet i denne omgang.

Konklusion:

Det lykkedes os både at optimere fjernstyringen af robotten og at tilføje subsumption arkitekturen uden at støde ind i store problemer. Næste gang vi mødes skal vi have fundet årsagen til, hvorfor RemoteControl laget ikke bliver undertrykt af TouchBehavior via AvoidFront laget, hvilket det ellers burde blive.