KU Leuven
Departement Computerwetenschappen

P&O: Computerwetenschappen

Tussentijdsverslag 3

Team:
Zilver

Sam Gielis
Sophie Marien
Toon Nolten
Nele Rober
Gerlinde Van Roey
Maxim Van Mechelen

Academiejaar 2012 – 2013

Samenvatting

Het P&O-project heeft als doel een robot autonoom een doolhof te laten verkennen. Dit verslag beschrijft de invulling die team Zilver aan het project gaf. De robot is voorzien van een lichtsensor en een ultrasone sensor. Deze staan vast gemonteerd en kunnen niet onafhankelijk van de robot bewegen. De aansturing van de robot gebeurt via bluetoothverbinding. Een Grafische User Interface (GUI) maakt deze aansturing op een gebruiksvriendelijke manier mogelijk. De GUI geeft de baan van de robot weer. Ook de sensorwaarden en een historiek ervan worden weergegeven.

De robot kan zich autonoom door een doolhof bewegen zonder op muren te botsen. Tijdens het rijden slaat de robot een map van de doolhof op. Op elke nieuwe tegel kijkt de robot rond zodat muren gedetecteerd kunnen worden door interpretatie van sensorwaarden. Wanneer de hele doolhof verkend is, gebruikt de robot het $A*$-algoritme om de kortste weg naar de finish (via het checkpoint) te bepalen. De finish en het checkpoint worden aangegeven met een bepaalde barcode. Andere barcodes laten de robot een opdracht uitvoeren.

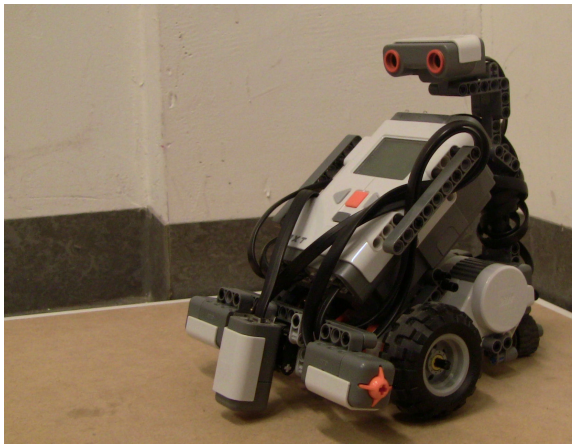
Om de afwijking op de aansturing te minimaliseren, oriënteert de robot zich regelmatig op een witte lijn. Zo blijft de robot steeds zo dicht mogelijk bij het midden van de tegels.

Een computerprogramma simuleert de werking van de robot. Het is voor de simulator mogelijk door een virtuele doolhof te rijden en deze op gelijkaardige wijze te verkennen. De sensorwaarden worden gesimuleerd met een afwijking om de werkelijke robot zo goed mogelijk te benaderen. Verder worden dezelfde algoritmes gebruikt als voor de robot. Dit laat toe de algoritmes te testen zonder de robot hiervoor te gebruiken.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren



Figuur 1: Robot

1 Inleiding

In het kader van het vak 'Probleemoplossen en Ontwerpen: computerwetenschappen' wordt gewerkt rond autonome intelligente robots. Verschillende teams bouwen en programmeren een robot met behulp van LEGO Mindstorms [1]. Deze robot moet uiteindelijk volledig autonoom een doolhof kunnen verkennen.

Op de derde demonstratie kan de robot alle taken van de vorige demonstratie nog steeds uitvoeren. De robot kan zich volledig autonoom voortbewegen. Wanneer de robot zich in een doolhof voortbeweegt, kan hij deze in kaart brengen. Bij het inlezen van barcodes voert de robot een bepaalde opdracht uit. Op het moment dat de volledige doolhof is ingelezen, bepaalt de robot de kortste weg naar de tegel met de barcode 'checkpoint'. Vandaar gaat de robot zo snel mogelijk naar de tegel met barcode 'finish'.

2 Bouw van de robot

LEGO Mindstorms [1] biedt een bouwpakket voor een robot aan. Een NXT-microcomputer laat toe de robot te programmeren. Met behulp van leJOS [2] kan dit in Java.

2.1 Fysieke bouw

Bij het bouwen van de robot (zie figuur 1) werd het ontwerpboekje gevolgd. Deze compacte samenstelling leek geen directe nadelen te hebben. Twee grote wielen worden elk met hun eigen motor aangestuurd. Een klein wiel achteraan zorgt ervoor dat de robot vlot kan draaien en wordt niet aangedreven. De sensoren (zie figuur 2) werden als volgt geïnstalleerd:

- *lichtsensor:* vooraan en dicht tegen de grond.
- *ultrasone sensor:* bovenaan, naar voren kijkend; de sensor staat vast gemonteerd.
- *druksensoren:* aan beide zijkanten, één aan de linkerkant en één aan de rechterkant.

Een alternatieve opstelling bestaat erin de ultrasone sensor op een derde motor te monteren zodat deze onafhankelijk van de robot kan ronddraaien. De meetwaarde van de ultrasone sensor zou dan afhangen van zijn positie ten opzichte van de robot, wat het interpreteren moeilijker maakt. Voor deze opstelling werd daarom niet gekozen.

(a) Ultrasone sensor (b) Licht- en Druksensor

Figuur 2: Sensoren

2.2 Calibratie van de motoren

De robot wordt aangedreven door twee motoren, elk verbonden met één van de twee grote wielen. De aansturing gebeurt door te bepalen hoeveel graden de wielen moeten draaien. Beide motoren kunnen onafhankelijk ingesteld worden. De robot kan vooruit bewegen en rond zijn as draaien. De resultaten van de calibratie worden weergegeven in tabel 1.

	linkerwiel	rechterwiel	aantal graden
1 cm vooruit	voor	voor	20,8°
1 cm achteruit	achter	achter	20,8°
180°draaien linksom	achter	linker	701°
180°draaien rechtsom	voor	achter	701°

Tabel 1: Resultaten calibratie motoren

2.2.1 Eén cm vooruit bewegen: bepalen van x

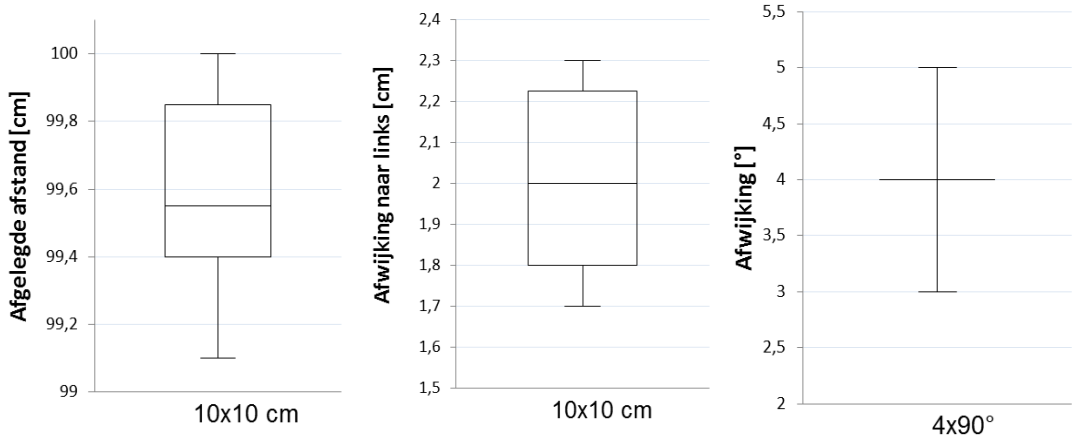
De parameter x bepaalt het aantal graden dat de wielen moeten draaien opdat de robot één cm vooruit beweegt.

Een schatting voor x via de *diameter* (in centimeter) gebeurt als volgt:

$$x_0 \approx \frac{360}{\pi \cdot diameter}$$

Een nauwkeurigere bepaling van x gebeurt via tests.
De robot wordt naast een lintmeter geplaatst en krijgt opdracht beide wielen $100 \cdot x$ graden te laten draaien. Bij een perfect gekozen waarde voor x, legt de robot precies één meter af. Indien niet, wordt x aangepast. De test wordt herhaald tot x een voldoende nauwkeurig resultaat geeft.

Een volgende test bestaat eruit de robot beide wielen tien maal $10 \cdot x$ graden te laten draaien. Een boxplot van de totaal afgelegde afstand wordt weergegeven in figuur 3. Wanneer geen afwijking op één meter wordt waargenomen, heeft het starten en stoppen van de motoren geen invloed op de totale afgelegde afstand. Dit blijkt echter wel het geval. In verdere algoritmes wordt hier rekening mee gehouden. De robot legt steeds één lange afstand af in plaats van vele korte.
Uit deze test blijkt bovendien dat de robot een afwijking naar links heeft wanneer hij rechtdoor rijdt. De boxplot van deze afwijking wordt eveneens weergegeven in figuur 3.



(a) Afgelegde afstand rechtdoor (b) Afwijking naar links (c) Afwijking op 360°

Figuur 3: Boxplots calibratie motoren

2.2.2 Volledig rond de as draaien: bepalen van y

De parameter y bepaalt het aantal graden dat de wielen moeten draaien opdat de robot 360°rond zijn as zou draaien. Beide wielen draaien hierbij in tegengestelde richting.
Een schatting voor y via de diameter van het wiel, met as_{robot} (afstand tussen beide wielen) en *diameter* in centimeter:

$$y_0 \approx \frac{(2 \cdot \pi) \cdot (as_{robot}/2)}{diameter_{wiel}/2}$$

De robot wordt naast een lijn geplaatst en krijgt de opdracht zijn wielen y graden in tegengestelde richtingen te laten draaien. y wordt aangepast tot de robot na het draaien opnieuw precies naast de lijn uitkomt.

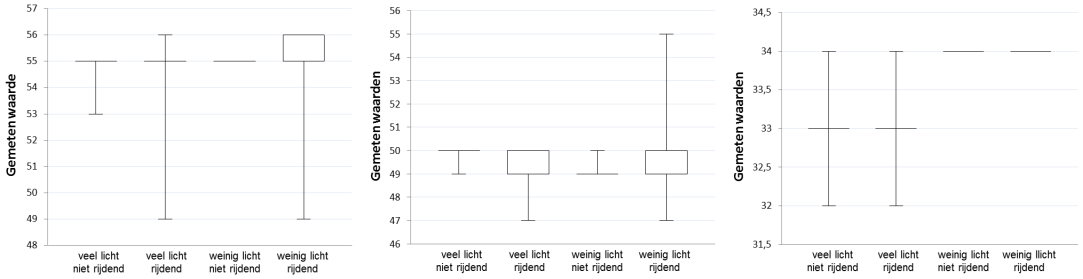
Door de robot in stappen rond zijn as te laten draaien ($4 \cdot 90°$) kan de invloed van het starten en stoppen van de motoren bepaald worden. Figuur 3 geeft een boxplot van de gemeten afwijkingen op 360°. De algoritmes houden rekening met deze afwijking. Wanneer de robot tijdens het verkennen van de doolhof naar één kant draait, draait hij, zo mogelijk, evenveel graden naar de andere kant terug. De totale som van de gedraaide graden wordt zo dicht mogelijk bij nul gehouden. Dit maakt de afwijking gedeeltelijk ongedaan. Met de variaties op de afwijking kan echter geen rekening gehouden worden. De algoritmes laten de robot zo weinig mogelijk draaien.

2.2.3 Besluit calibratie motoren

Uit de tests blijkt dat de robot niet perfect rijdt en draait. Het is daarom nodig regelmatig te corrigeren, zodat de robot toch steeds in het midden van de regels rijdt. Deze correcties worden in de sectie 3.1 en **??** beschreven.

2.3 Calibratie van de lichtsensor

De lichtsensor meet de lichtintensiteit van de omgeving. De sensor kan zelf ook een rood licht uitsturen. Hoe minder licht gereflecteerd wordt door de omgeving, hoe donkerder de omgeving. Op deze manier kunnen de meetwaarden geïnterpreteerd worden als witte of zwarte lijnen.
De lichtsensor wordt in verschillende omstandigheden getest: bij direct kunstlicht, in de schaduw, terwijl de robot rijdt en terwijl hij stilstaat. Dit voor alle soorten ondergrond die in de doolhof voorkomen: een paneel, een witte lijn en een zwarte lijn. Boxplots worden weergegeven in figuur 4. Deze toont dat het gemiddelde enkel bij een zwarte lijn echt afhangt van de omstandigheden. De afwijking wordt wel sterk bepaald door het al dan niet rijden van de robot. Bij het simuleren en het interpreteren van de meetwaarden wordt hier rekening mee gehouden.



(a) Witte lijn (b) Paneel (c) Zwarte lijn

Figuur 4: Boxplots calibratie lichtsensor

2.4 Calibratie van de ultrasone sensor

Een ultrasone sensor zendt ultrasone geluidsgolven uit. Indien een object in de buurt staat, weerkaatsen de golven hierop. De robot ontvangt dan zijn eigen golven. De tijd tussen het uitzenden en ontvangen laat toe de afstand tot het object te bepalen.

In een doolhof, kan een robot muren tegenkomen. Ook de paaltjes, die de muren omhoog houden, worden als object gedetecteerd. Om te vermijden dat de paaltjes als muur worden geinterpreteerd worden enkel meetwaarden kleiner dan 28 cm als muur beschouwd.

Deze 28 cm werd als volgt bepaald: De robot wordt in het midden van een tegel voor een muur geplaatst. De ultrasone sensor meet waarden tussen 12 cm en 24 cm. Vervolgens werd de muur verwijderd en werden er paatjes geplaatst. Nu werden er waarden tussen 32 cm en 50 cm gemeten. Deze metingen leidden tot een grens van 28 cm.

De afwijking van de ultrasone sensor wordt gemeten door de robot op een bepaalde afstand van een muur te zetten. Verschillende afstanden worden zo gemeten. Boxplots in figuur 5 geven de gemeten waarden ten opzichte van de werkelijke waarden weer. Hoewel de variantie op de gemeten waarde niet groot is, komt de gemiddelde gemeten waarde niet altijd overeen met de werkelijke afstand. Vooral voor afstanden kleiner dan 20 cm en groter dan 145 cm. Waarden buiten het interval [20,145] worden daarom niet als betrouwbaar verondersteld.

3 Algoritmes

3.1 Rechtzetten op een witte lijn

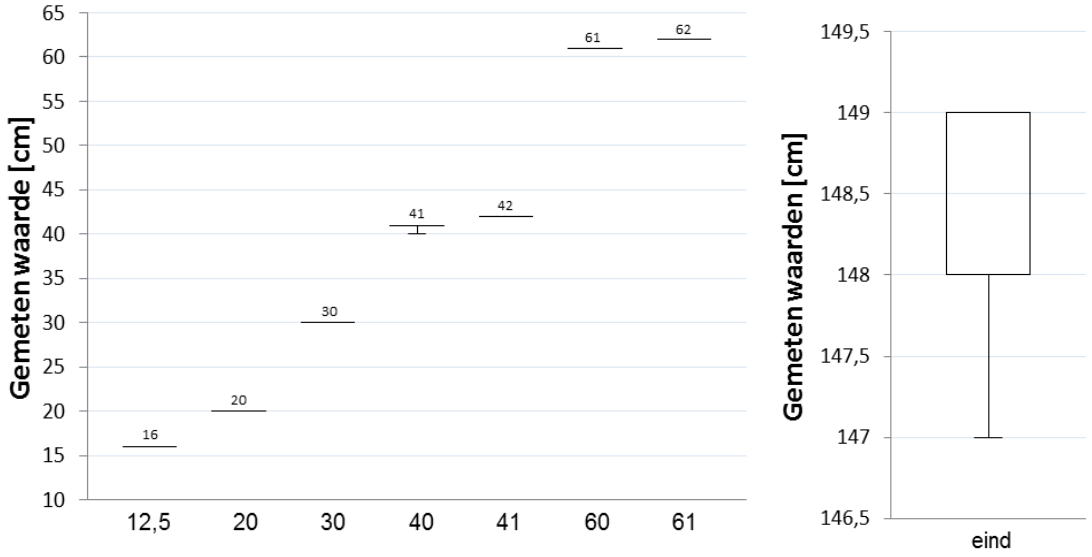
Indien regelmatig gecorrigeerd wordt op de calibratieafwijkingen kan de impact ervan geminimaliseerd worden. Deze correctie gebeurt onder andere door de witte lijnen als referentie te nemen en hier loodrecht op te oriënteren. List De robot rijdt dan weer rechtdoor. Listing 2 toont een algoritme dat gebruikt kan worden wanneer de robot net over een witte lijn staat (wanneer hij volledig rond zijn as draait zou hij de witte lijn moeten detecteren).

Listing 1: Witte Lijnalgoritme (pseudocode)

```
1 AngleTurned = 0
2
3 Draai(rechts,5) tot witte lijn.
4 Draai(links,1) tot witte lijn EN AngleTurned++.
5 Draai(rechts, AngleTurned/2).
```

De robot staat best net over de witte lijn. Wanneer de robot dan op een witte lijn tussen twee panelen staat, zorgt de donkere scheidingslijn tussen twee panelen niet voor verwarring.

Dit algoritme wordt tijdens het verkennen en tijdens het rijden naar de finish uitgevoerd. De robot zet zich elke vijf tegels recht.



(a) meetwaarden (mediaan van de meetwaarden wordt in de figuur weergegeven)

(b) laatste geldige meetwaarde

Figuur 5: Boxplots calibratie ultrasone sensor

3.2 Centreren aan de hand van twee muren

Een alternatieve manier om te corrigeren op calibratieafwijkingen is door gebruik te maken van muren. Met dit algoritme zet de robot zich in het midden van een tegel. De robot kijkt naar een muur en meet op hoeveel afstand hij zich van de muur bevindt. Hij rijdt eventueel wat naar voren en meet dan de afstand tot de tegenovergestelde muur. Eventueel rijdt hij weer wat naar voren. Het algoritme is enkel toepasbaar wanneer de robot zich op een tegel met muren aan twee kanten bevindt.

Listing 2: Muuralgoritme (pseudocode)

```
1  Draai(rechts,90)
2  ALS       ultraSensor() < 28
3  DAN       Rijd(ultraSensor()-23)
4            Draai(links,180)
5            ALS       ultraSensor() < 28 EN ultraSensor() NIET in [21,25]
6            DAN       Rijd(ultraSensor()-23)
7                      Draai(rechts,90)
8
9  ANDERS    Draai(links,180)
10           ALS       ultraSensor() < 28
11           DAN       Rijd(ultraSensor()-23)
12                     Draai(rechs,90)
13           ANDERS    Draai(rechts,90)
```

Dit algoritme wordt uitgevoerd op tegels met barcodes. Zo zijn er zeker muren aan weerskanten. Het algoritme wordt niet uitgevoerd tijdens het rijden naar de finish.

3.3 Lezen van een barcode

Een *Barcode* bestaat uit acht stroken van twee cm breed. De eerste en de laatste strook zijn altijd zwart. De tussenin liggende stroken zijn ofwel zwart (waarde nul) ofwel wit (waarde één). Deze combinatie van enen en nullen is een binaire voorstelling van een integer.

De klasse *BarDecoder* zet de integer om in de juiste opdracht. Ook indien de *Barcode* in de

andere richting gelezen werd, wordt de juiste opdracht toegekend.

Zowel de robot als de simulator hebben een *BarcodeThread* die checkt of er een zwarte ondergrond is. Dit gebeurt terwijl de robot rijdt. Bij detectie van een zwarte ondergrond, wordt het barcodealgoritme opgeroepen.

Aan de start van het barcodealgoritme staat de robot op de eerste zwarte strook. De robot rijdt twee cm vooruit, leest de waarde van zijn lichtsensor en interpreteert de waarde van de strook. Zo tot alle stroken gedetecteerd zijn. De waarde van de barcode wordt opgeslagen in de map van de doolhof. Nadien rijdt de robot nog vier cm vooruit zodat hij de barcode zeker voorbij is. Het muuralgoritme wordt opgeroepen en daarna wordt de opdracht bij de barcode uitgevoerd.

3.4 Verkennen van een doolhof

De robot kan een volledige doolhof autonoom verkennen. Elke tegel die de robot passeert, wordt gemarkeerd. Indien alle tegels gemarkeerd zijn, is de volledige doolhof doorzocht. Elke tegel houdt een boolean bij die deze markering voorstelt.

Bij elke tegel onthoudt de robot alle uitwegen (de naburige tegels). Vervolgens slaat hij de laatste nog-niet-bekeken uitweg in. Wanneer de robot op een doodlopend stuk komt, keert hij terug tot een kruispunt waar nog niet alle uitwegen van bekeken werden. De robot gebruikt het kortste pad algoritme om de weg naar dit kruispunt te bepalen. Nadat de robot het hele doolhof bekeken heeft, kijkt hij na of de doolhof een 'finish'-barcode en een 'checkpoint'-barcode bevat. Indien niet, dan stopt de robot en voert hij het kortste pad algoritme niet uit.

Enkele optimalisaties werden doorgevoerd in het algoritme. De genoemde aanpassing implementeert steeds ook de aanpassingen die eerder werden toegevoegd. Tabel 2 geeft de resultaten van de aanpassingen weer. Figuur 6 toont de gebruikte doolhoven en de uitvoering van het algoritme met aanpassing E. Er werd getracht het aantal draaiingen en afgelegde afstand te minimaliseren.

A basisalgoritme zonder optimalisatie: draai bij elke tegel vier keer (eindig in startoriëntatie) en neem de laatste tile in de queue als volgende tile.

B neem steeds de buur die met het minst aantal rotaties bereikt kan worden als volgende tile.

C draai bij elke tegel slechts drie keer (eindig niet meer in startoriëntatie).

D muren die vanuit een naburige tile reeds gedetecteerd werden, worden niet nog eens nagekeken.

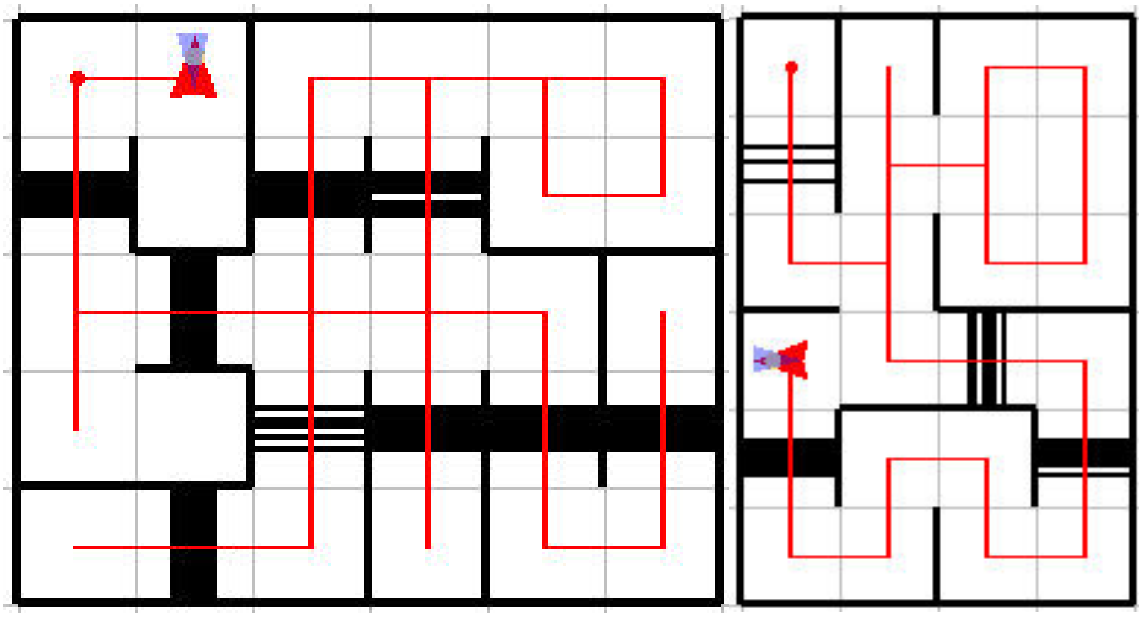
E tiles waarvan de vier zijden al gekend zijn en waaraan drie muren grenzen, worden niet meer bezocht ('dead-ends' kunnen onmogelijk barcodes bevatten, dus dit is geen probleem)

	map 1		map 2	
optimalisatie	°gedraaid	cm afgelegd	°gedraaid	cm afgelegd
A	14400	2160	10620	1080
B	13770	1920	10800	1160
C	11970	2000	9090	1080
D	9180	2000	6570	1080
E	8820	1880	6570	1080

Tabel 2: Testen algoritmes

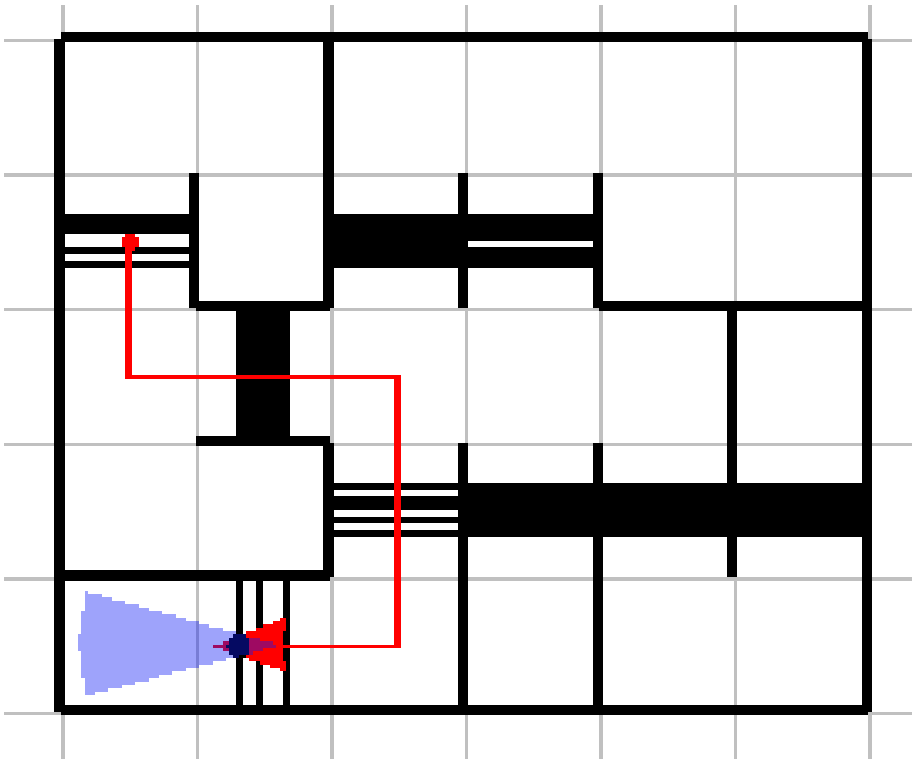
3.5 Vinden van het kortste pad

Het kortste padalgoritme implementeert $A*$ [3]. Het maakt gebruik van een graaf die opgesteld wordt tijdens het verkennen van het doolhof. Als heuristiek wordt de Manhattanafstand [4] geïmplementeerd (zonder rekening te houden met muren) en als kost de totale afgelegde afstand (wel rekening houdend met muren). Figuur 7 toont een uitvoering van het algoritme.

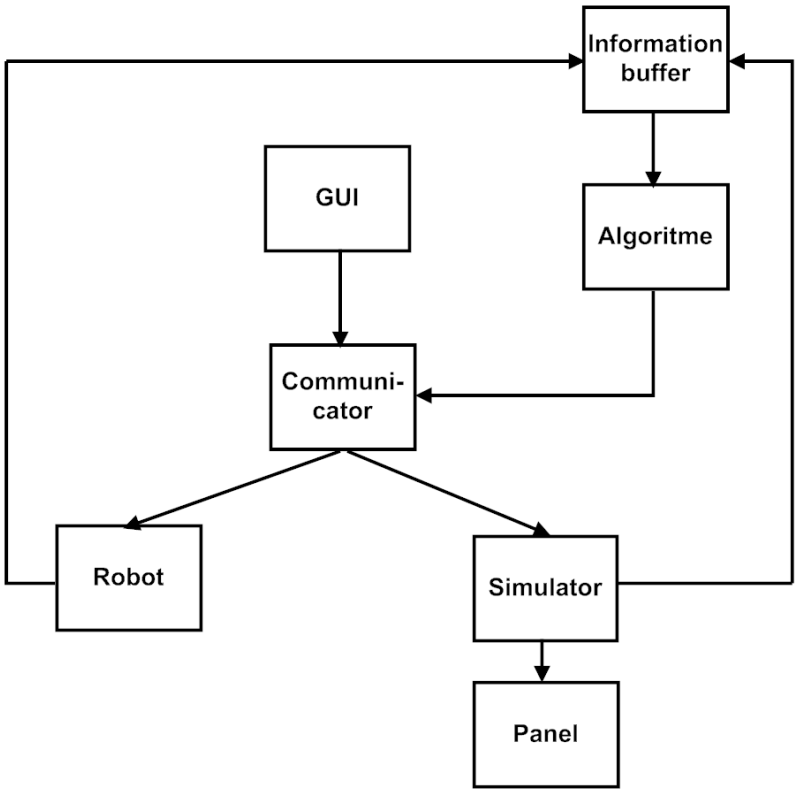


(a) map 1 (b) map 2

Figuur 6: Verkennen van een doolhof met implementatie van aanpassing E. Deze aanpassing maakt dat in map 1 niet alle tegels bezocht hoeven te worden. In map 2 doet deze situatie zich niet voor. Dit is te wijten aan de opbouw van de doolhof.



Figuur 7: Kortste padalgoritme



Figuur 8: Klassendiagram van het computerproject.

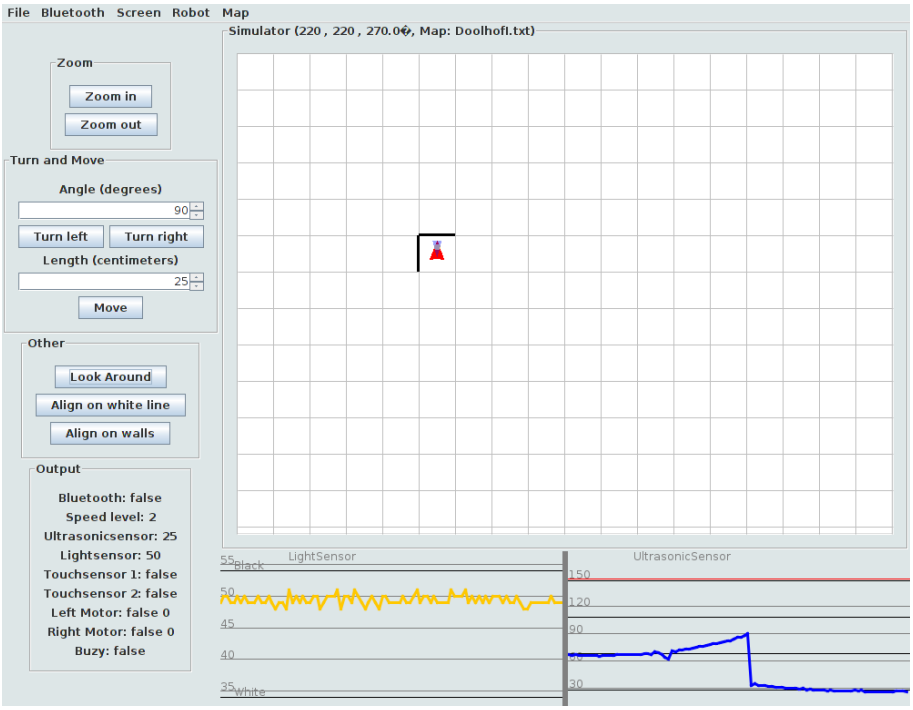
Dit algoritme markeert alle tegels die al onderzocht zijn (dit is een andere mark dan het verkenalgoritme). In de beginsituatie zijn alle tegels ongemarkeerd. De kost van de startpunt wordt op nul gezet. Daarna wordt de kost opbouwend aan elke tegel meegegeven.

4 Software

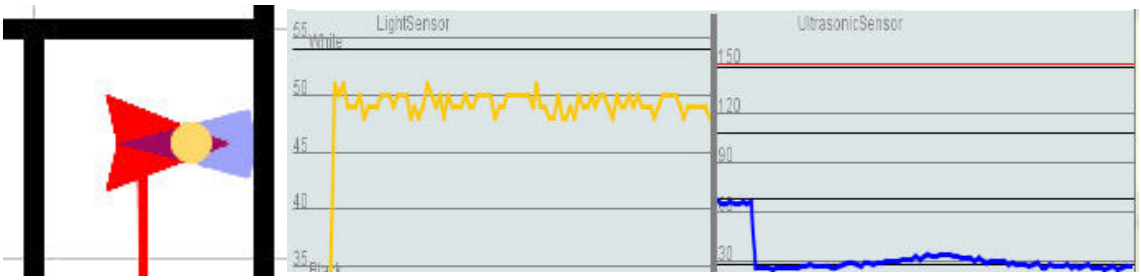
De software bestaat uit twee delen: een project dat op de NXT van de robot loopt (sectie 4.5) en een project dat op de computer loopt (sectie 4.1). Alles wordt aangestuurd via de *Graphical User Interface (GUI)* (sectie 4.2) . Deze toepassing laat toe de robot te besturen (via bluetooth: sectie 4.4) en de reacties van de robot weer te geven. Via de GUI kan ook een virtuele robot aangestuurd worden: de simulator (sectie 4.6). Een *Communication*-pakket stuurt de commando's van de GUI door naar de juiste unit (sectie 4.3). Wanneer robot of simulator een doolhof verkennen wordt een map opgebouwd met behulp van het *Mapping*-pakket (sectie 4.7).

4.1 Ontwerp van het computerproject

Zoals reeds vermeld bestaat de software uit twee projecten: één draait op een computer en één draait op de NXT-Brick. Figuur 8 toont een vereenvoudigd klassendiagram van de software op de computer. De gebruiker geeft zijn commando's door aan de *GUI*. Deze stuurt de commando's weer door naar de *Communicator* die verder doorstuurt naar ofwel de robot ofwel de simulator. De *Simulator* en de *Robot* geven hun sensorwaarden door aan de *Informationbuffer*. De *Informatiebuffer* stuurt ze weer door naar de *GUI* zodat deze de sensorwaarden kan weergeven.



Figuur 9: Grafische User Interface



(a) grafisch (b) grafieken

Figuur 10: Weergave van de sensorwaarden

4.2 Grafische User Interface

De GUI toont een grafische weergave van de doolhof en de baan die de robot erin aflegt: zie figuur 9. De sensorwaarden worden grafisch weergegeven: een bolletje op de robot geeft de kleur van de ondergrond aan en een blauwe straal toont het bereik van de ultrasone sensor. Grafieken tonen de historiek van de ultrasone sensor en de lichtsensor. Figuur 10 toont zowel de grafische weergave als de grafiek.

Via de GUI kan de gebruiker de robot of de simulator aansturen, naargelang de bluetooth geactiveerd is. Het is mogelijk virtuele doolhoven te laden en te verkennen. Gemeten sensorwaarden kunnen geëxporteerd worden naar een text-file. Bovendien is het mogelijk in de zoomen op de robot en kan met het scherm geschoven worden.

4.3 Het doorgeven van commando's

De GUI zet een actie van de gebruiker om in een commando. Dit commando wordt vertaalt naar een integer die naar de klasse *UnitCommunicator* wordt gestuurd. Twee van de commando's

hebben echter extra informatie nodig: *automatic move x cm forward* en *rotate x degrees*. Deze informatie wordt toegevoegd aan de integer door de integer uit te breiden met extra cijfers.

4.3.1 Bewerking op de integers

Integers stellen de commando's voor. In twee gevallen is echter meer informatie nodig: om de robot een bepaalde afstand te laten afleggen en om de robot een bepaald aantal graden te laten draaien. Deze afstand en dit aantal graden moet mee doorgegeven worden met de integer. De eenheden waarin de afstand en de hoek worden doorgestuurd zijn respectievelijk cm en graden.

De doorgegeven integer wordt als volgt opgebouwd:

- de waarde van de afstand (hoek) wordt vermenigvuldigd met 1000.
- de integer die het commando representeert, wordt hierbij opgeteld.

Om de bekomen resultaten terug op te splitsen in de twee oorspronkelijke gegevens worden volgende stappen gevolgd:

- een modulo-operatie van honderd geeft het laatste cijfer terug. Dit stelt het soort commando voor.
- dit getal wordt van de integer terug afgetrokken.
- de oorspronkelijke afstand (hoek) wordt bekomen door de integer door 1000 te delen.

Deze werkwijze brengt een beperking met zich mee: de waarde van de afstand (hoek) kan slechts tot twee cijfer(s) na de komma doorgegeven worden. De robot kan niet nauwkeuriger dan 0,1 cm aangestuurd worden. Hierdoor is het niet nodig de afstand nauwkeuriger door te geven. De nauwkeurigheid van de hoek is gevoeliger. De veelhoek stapelt immers veel afrondingsfouten op naarmate de lengte van de zijde en/of het aantal hoeken stijgt. Doordat de begin- en eindpunten niet samenvallen is te zien dat de som van de berekende hoeken samen geen 360 graden vormt.

4.4 Bluetooth

De communicatie tussen robot en computer gebeurt volledig via bluetooth. De GUI voorziet een knop om deze verbinding te maken en geeft de status van de verbinding weer. De GUI stuurt commando's door naar de robot via de *Communicator*-klassen.

De leJOS-API [2] voorziet een *NXTConnector* klasse. De methode *connectTo((String, String, int, int)* zet de bluetoothverbinding tussen computer en brick op. De belangrijkste argumenten hiervoor zijn de naam van de NXT-Brick en zijn DeviceUrl - in het geval van de gebruikte NXT: 'Silver' en '00:16:53:0A:04:5A'.

De bluetooth laat toe twee 'datastreams' tussen robot en computer op te zetten: een inputstream en een outputstream. Alle informatie wordt hierover doorgestuurd in de byte-voorstelling van een string. Op de Die bytes worden bij aankomst weer omgezet naar een string. De computer interpreteert die aan de hand van tags die vooraan de string staan. Naast de tags worden enkel integers doorgestuurd.

4.5 Robot

Het project op de NXT-brick bestaat uit een klasse *CommandUnit*, een klasse *Command* en enkele threadklassen. Deze laatsten maken het mogelijk meerdere dingen tegelijk te doen (bijvoorbeeld: sensorwaarden inlezen terwijl de robot rijdt). Bij het opstarten, initialiseert de robot zijn sensoren. Wanneer de computer verbinding maakt, initialiseert de robot ook zijn 'datastreams' en start hij een *SensorThread* op.

De computer stuurt commando's in de vorm van integers. De robot vertaalt deze met behulp van de klasse *Command* en voert de juiste actie uit. Sommige methodes retourneren een waarde

aan de computer. Dit gebeurt door de waarde op een 'datastream' te zetten, voorafgegaan door een tag (bijvoorbeeld: [US] voor ultrasonesensor data).

Enkele belangrijke methodes van *CommandUnit*:

- *updateCoordinates:* de robot houdt zijn positie bij. Bij elke beweging zullen de coördinaten worden geüpdated.
- *updateStatus:* stuur alle statusgegevens (lichtsensor, coördinaten, ...) naar de computer.
- *main:* wacht tot de computer een commando geeft en voer dit uit.
- *moveForward:* beweegt voorwaarts, maar stopt bij een barcode.

De *SensorThread* stuurt elke 50 milliseconden sensorinformatie door naar de computer. Zo blijft de data up to date zonder dat de 'datastream' te veel informatie moet slikken.

4.6 Simulator

De *Simulator* bootst de werking van de robot virtueel na. Hij kan dezelfde commando's uitvoeren als de werkelijke robot.

De belangrijkste klassen in het *Simulator*-pakket:

- *SimulationSensorData:* bevat numerische resultaten van sensortests met de echte rotot; deze data wordt gebruikt om de sensorwaarden te simuleren.
- *SimulationPilot:* houdt de positie en de richting van de 'robot' bij.
- *SimulationPanel:* tekent de baan van de 'robot' in het tekenpaneel en houdt de map van de verkende doolhof bij.

Het opzetten van het tekenpaneel gebeurt in de GUI. De roosters op de achtergrond van het tekenpaneel hebben dezelfde afmetingen als de secties van de panelen. Zo kan een muur enkel op een lijn van het grid staan. Wanneer de 'robot' een pad aflegt, tekent de simulator dit in het tekenpaneel als een rode lijn (herschaald: één cm = één pixel). De lijn bestaat uit verschillende cirkels die elkaar gedeeltelijk overlappen. Het *SimulationPanel* houdt alle bezochte coördinaten bij. De klasse bevat een methode die deze cirkels één na één tekent. Dit zorgt ervoor dat de lijn continu bijgewerkt wordt. De huidige positie en de huidige oriëntatie van de 'robot' wordt weergegeven door een driehoek die draait met de oriëntatie.

4.7 Mappen van een doolhof

Het *Mapping*-pakket heeft klassen zoals *Tile*, *Edge*, *Obstruction* en *Barcode* die elementen uit de wereld van een robot voorstellen. Een *Tile* stemt overeen met één tegel van de doolhof en heeft vier *Edges*: één voor elke zijde. Die *Edges* houden de twee aanliggende tegels bij en eventueel een *Obstruction*, bijvoorbeeld een muur. De klasse *MapGraph* brengt al deze elementen samen. Ze houdt een begin-tegel bij en en huidige-tegel. Ze biedt functionaliteiten aan om van de huidige tegel naar de tegel Noord, Oost, Zuid of West ervan te reizen en de map dynamisch uit te breiden. Zo wordt impliciet een hele graaf bijgehouden. De klasse *MapReader* kan uit een bepaalde text-file een MapGraph opstellen die overeenkomt met de doolhof die in het bestand gedefinieerd wordt.

De *Simulator* heeft een *MapGraph* die de virtuele doolhof voorstelt. Tijdens het verkennen wordt een nieuwe *MapGraph* opgesteld in de klasse *SimulationPanel*, die de muren ook tekent. Ook de robot maakt gebruik van het *SimulationPanel* om de verkende map op te slaan.

5 Besluit

De uiteindelijke fysieke bouw van de robot bestaat uit de NXT, de wielen met aandrijvingen, een ultrasone sensor, een lichtsensor en twee druksensoren. De calibratie toont aan dat het nodig is regelmatig te corrigeren. Bovendien moeten de sensorwaarden juist geïnterpreteerd worden om correctie conclusies te trekken.

De GUI biedt de gebruiker een mogelijkheid om verschillende commando's aan de robot te geven. De uitleeswaarden van de sensoren vertellen de gebruiker wat de robot 'ziet'. Op deze manier kan de doolhof geïnterpreteerd worden. De robot wordt tijdens het verkennen van de doolhof regelmatig gecorrigeerd met het witte lijnalgoritme en het muuralgoritme. Telkens wanneer een barcode gesignaleerd wordt, voert de robot de bijhorende opdracht uit. Één bepaalde barcode stelt een 'checkpoint' voor en één bepaalde barcode stelt een 'finish' voor. Wanneer de hele doolhof verkend is, rijdt de robot zo snel mogelijk via de 'checkpoint' naar de 'finish'.

De simulator is ook verbonden aan de GUI. Deze voert dezelfde opdracht uit als de robot. Het is mogelijk een virtueel doolhof te laden en de simulator hierin te laten 'rijden'.

A Demo 2

De sensoren worden voor demo 2 wel geïnstalleerd. Na calibratie kunnen ze informatie doorzenden naar de robot. Threads zorgen ervoor dat de robot tegelijkertijd sensorwaarden kan lezen en doorsturen.

De meetwaarden worden in de GUI weergegeven zodat een gebruiker de robot kan besturen zonder deze te zien. Bovendien is de simulator gekoppeld aan de robot. Wat de robot doet, doet de simulator ook en wordt getekend in de GUI. De simulator kan ook onafhankelijk van de robot opereren. Het is mogelijk een virtuele doolhof te laden en te simuleren dat de 'robot' zich hierdoor beweegt. Zowel robot als simulator kunnen zich rechtzetten op een (virtuele) witte lijn. Het is bovendien mogelijk de robot zich in het midden van de tegel te laten zetten.

A.1 Resultaten

Het verslag voor de tweede demonstratie bevatte enkele paragrafen die tegen het einde nog snel geschreven waren. Deze paragrafen waren van mindere kwaliteit. Ook sommige afbeeldingen werden verkeerd ingevoegd.

Op de demonstratie reed de robot de eerste tegels zoals het hoorde. Ook het witte lijnalgoritme werd goed uitgevoerd. De sensorwaarden werden weergegeven in de GUI, zodat de bestuurder, die de robot en de doolhof niet zag, zich een beeld kon vormen van de doolhof. Op een gegeven moment gaf de bestuurder de robot opdracht zich midden op een tegel te zetten. Dit algoritme maakt gebruik van de muren rond de tegel. De robot bevond zich op dat ogenblik echter op een tegel die door geen enkele muur omsloten werd. De bestuurder had dit niet eerst nagekeken. Dit had als gevolg dat de robot helemaal scheef stond, zonder dat de bestuurder dit wist.

De sensorwaarden werden niet gesimuleerd in de simulator. De simulator maakte in zijn algoritmes rechtstreeks gebruik van de virtuele doolhof. Robot en simulator maakten met andere woorden gebruik van andere algoritmes, wat niet het doel is van een simulator.

De simulator kon wel door een virtuele doolhof manoeuvreren. Meestal 'botste' hij tegen de muren, maar soms reed hij toch door een muur.

A.2 Conclusies

Aan het verslag zou vroeger begonnen moeten worden zodat fouten vermeden kunnen worden.

De align-algoritmes (op een witte lijn en in het midden van een tegel) werken enkel in specifieke situaties. De algoritmes dienen enkel in deze omstandigheden uitgevoerd te worden.

De simulator dient de sensorwaarden te simuleren en zou dezelfde algoritmes moeten gebruiken als de robot. Op deze manier kunnen de algoritmes getest worden zonder steeds op de robot te wachten.

A.3 Oplijsting aanpassingen verslag

Volgende secties werden aangepast ten opzichte van de tweede demonstratie:

- *2.2 Calibratie van de motoren:* boxplots en besluit toegevoegd.
- *2.3 Calibratie van de lichtsensor:* boxplots en conclusie m.b.t. interpretatie toegevoegd.
- *2.4 Calibratie van de ultrasone sensor:* boxplots en conclusie m.b.t. interpretatie toegevoegd.
- *3.1 Rechtzetten op een witte lijn:* robot moet over de witte lijn staan.
- *3.2 Centreren aan de hand van twee muren:* nieuwe sectie.
- *3.3 Lezen van barcodes* nieuwe sectie.

- *3.4 Verkennen van een doolhof:* nieuwe sectie.
- *3.5 Vinden van het kortste pad:* nieuwe sectie.
- *4.1 Ontwerp van het computerproject:* nieuw klassediagram en nieuwe klassen.
- *4.2 Grafische User Interface:* historiek sensorwaarden in grafieken.
- *4.4 Bluetooth:* 'datastreams'.
- *4.6 Simulator:* simuleren via data.
- *4.7 Mappen van een doolhof:* nieuwe sectie.

Referenties

[1] *Lego Mindstorms*: Een uitbreiding op de LEGO bouwstenen waarmee kleine, aanpasbare en programmeerbare robots gebouwd kunnen worden. Een centrale besturingsmodule ('the brick') kan geprogrammeerd worden met verschillende programmeertalen. In eerdere versies werd een RCX gebruikt voor de brick, nu wordt met NXT gewerkt. De brick kan enkele motoren aandrijven. Bovendien kunnen er verschillende sensoren, o.a. een ultrasone sensor en een lichtsensor, aangesloten worden. [www.lego.com] [http://en.wikipedia.org/wiki/Lego–Mindstorms]

[2] *leJOS*: Een kleine Java Virtuele Machine die toelaat de NXT-brick te programmeren. leJOS voorziet verschillende klassen die o.a. de motoren aansturen en een bluetoothverbinding opzetten. [http://lejos.sourceforge.net/]

[3] *A**: Een optimaal zoekalgoritme. *A** kiest zijn volgende punt op basis van een kostenfunctie: de werkelijke kost om tot de beschouwde buur te geraken, plus de geschatte kost om van dat punt naar het doel te geraken. Met andere woorden, het algoritme combineert een heuristisch algoritme met een kostenalgoritme. Zie ook de cursus *Artificiële Intelligentie (H06U1A)* door Daniel De Schreye.

[4] *Manhattanafstand:* Deze heuristiek tekent vanuit het ene punt een horizontale lijn en vanuit het andere punt een verticale lijn. De gemeten afstand is die van het ene punt tot het snijpunt, plus die van het andere punt tot het snijpunt.
In de doolhof wordt geen rekening gehouden met muren.
De Manhattanafstand verwijst naar een gelijknamige stadsdeel in New York City VSA). New York werd in het jaar 1625 door de Nederlanders gesticht. De straten werden in dambordpatroon gelegd: 'Avenues' van noord naar zuid en 'Streets' van oost naar west.
$[http://nl.wikipedia.org/wiki/Manhattan\_(New\_York)]$
$[http://nl.wikipedia.org/wiki/New\_York\_City]$

[5] *Bag*: Een manier om objecten op te slaan. Het is mogelijk te itereren over de verzameling, maar het is niet mogelijk objecten te verwijderen uit de verzameling. Meer documentatie: Section 1.3 of *Algorithms, 4th Edition* door Robert Sedgewick en Kevin Wayne.
[http://algs4.cs.princeton.edu/13stacks] [http://algs4.cs.princeton.edu/13stacks/Bag.java.html]