UNIVERSITEIT GENT

Opgave Project
Functioneel Programmeren

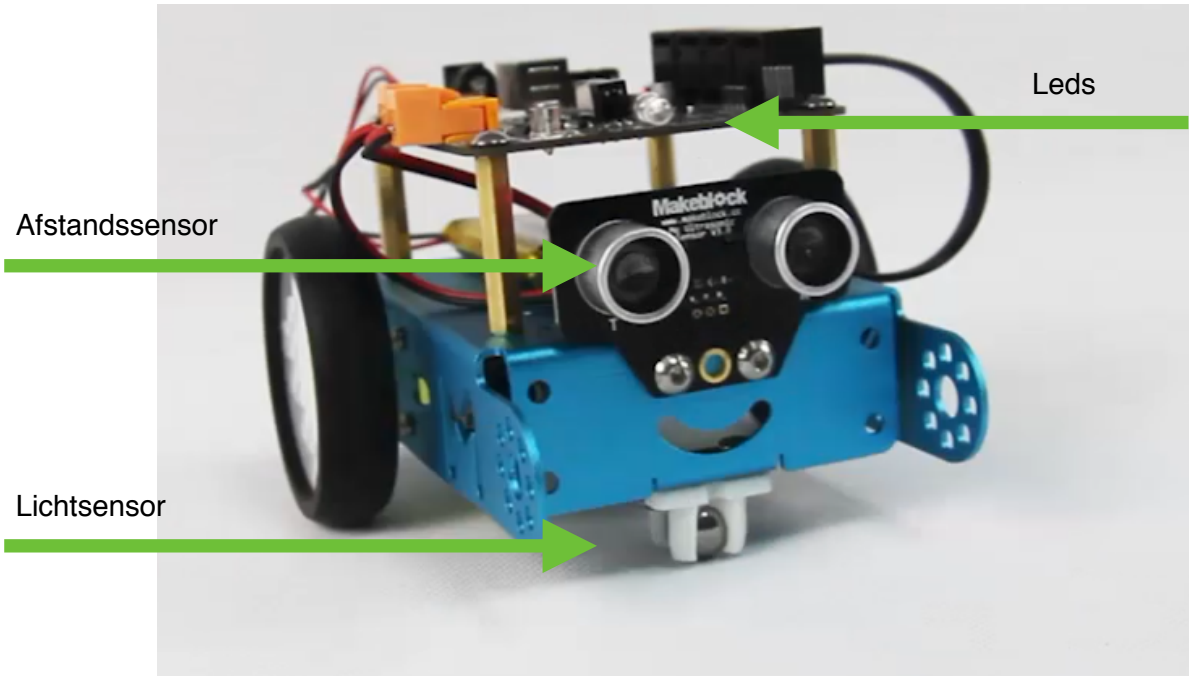
Inleiding

Het project functioneel programmeren bestaat uit **twee** onderdelen. Het eerste onderdeel bestaat eruit om een ***simulator*** te maken voor een robot die kan rondrijden in een fictieve wereld. Het tweede onderdeel bestaat uit het maken van een ***mini programmeertaal*** om een robot te programmeren. Deze mini taal zal je zo uitwerken dat je zowel de fictieve als een echte robot enkele leuke taken kan laten uitvoeren zoals het volgen van een lijn en het uit ontwijken van obstakels.

Om te valideren of je programmeertaal goed werkt zal je **drie** verschillende programma's maken in je eigen programmeertaal. Ten eerste een programma om van de robot een brandweerwagen of politiewagen te maken door *de leds van de robot te doen blinken*. Ten tweede een programma die de robot *een lijn laat volgen* en ten derde een programma die de robot *obstakels ontwijken*.

Robot

De robot die je zal gebruiken is een didactische robot die ook gebruikt wordt om kinderen de basisbeginsels van programmeren aan te leren. We gaan deze robot echter niet programmeren in Scratch maar in Haskell. Om dat te doen staat er op minerva een hardware bibliotheek ter beschikking. De robot heeft twee motoren die je toelaten om de wielen van de robot individueel te laten bewegen. Verder heeft de robot ook twee leds bovenaan op de printplaat. Deze leds kan je elke RBG waarde laten aannemen. De robot heeft ook verschillende sensoren waarvan we voor dit project twee zullen gebruiken. Een afstand sensor en een lichtsensor. De afstandssensor laat toe om na te gaan of er een voorwerp voor de robot staat. De lichtsensor is naar de grond gericht en geeft de gebruiker de mogelijkheid om na te gaan of de grond net onder de robot wit of zwart is.



Simulator

De simulator moet alle functionaliteit die de robot bibliotheek aanbiedt ook ondersteunen. Concreet wilt dit zeggen dat alle functies die de bibliotheek voor de mBot implementeert ook geïmplementeerd dienen te worden door de simulator: `openMBot`, `closeMBot`, `sendCommand`, `readUltraSonic`, `readLineFollower`, `setMotor`, `leftMotor`, `rightMotor`, `setRGB`, `Line(LEFTB, RIGHTB, BOTHB, BOTHW)`, `Command(..)`, `Device`.

Inplaats van een echte robot te laten rondrijden zal je implementatie een gesimuleerde robot laten rondrijden in een fictieve wereld. Concreet zal je virtuele robot rondrijden in een rechthoekige gridwereld. Deze wereld dient volgende twee elementen te ondersteunen:

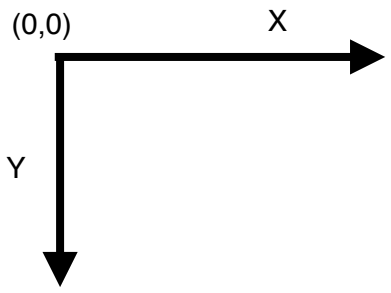
- **Muren:** Elk van de cellen in de gird wereld kunnen ofwel een muur of een lege ruimte zijn. Muren kan de (virtuele) robot detecteren met zijn sensoren. Uiteraard kan je robot niet door een muur rijden.

- **Lijnen:** De robot kan detecteren of de grond onder zijn sensors zwart of wit is. Het moet dus mogelijk zijn voor de gebruiker om lijnen te tekenen op de grond. Als lijnen over een muur lopen moet het tekenen van de muur voorrang krijgen.

Representatie gridwereld

Het inladen van de virtuele wereld dient te gebeuren via een tekst file. Een voorbeeld van zulke tekstfile kan je hieronder bekijken. De tekst file begint met een ascii representatie van de gridwereld gevolgd door een opsomming van lijnstukken. De coördinaten van deze lijnstukken bevinden zich in een assenstelsel met oorsprong bovenaan links.

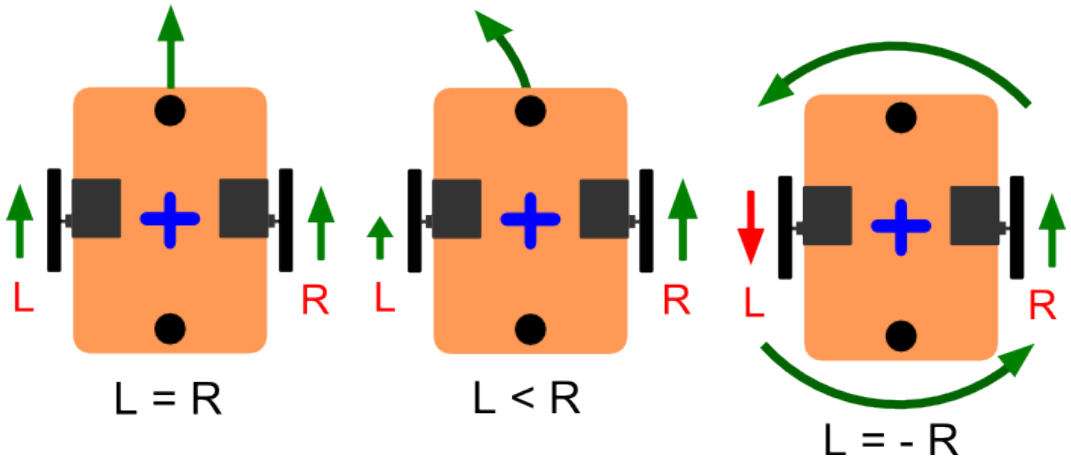
```
+------------------+
|                  |
|  x    >          |
|  x               |
|  xxxxxxxxxx      |
|            x     |
|            x     |
+------------------+
 (0,0)   (10,10)
(10,10)  (20,10)
```



Gridwereld formaat	
+	Hoek van de wereld
-	Horizontale grens van de wereld
I	Verticale grens van de wereld
< > ^ v	Plaats en richting van de robot
X	Muur
(x,y)	Coördinaat voor het voorstellen van een lijnstuk.

Simulatie robot

Een belangrijk onderdeel van je simulatie zal eruit bestaan om de robot op een realistische manier laten rond te rijden in de wereld. De robot heeft twee motors die we zowel vooruit als achteruit kunnen laten draaien met verschillende snelheden. Afhankelijk van de relatieve snelheid van deze motors kunnen we de robot vooruit, achteruit, links en rechts laten bewegen. Een voorbeeld van hoe dit in zijn werk gaat kan je zien op onderstaande figuur[1].



Om de beweging van de robot te simuleren kan je gebruik maken van onderstaande formules[2].

$$\Delta x = \Delta t \frac{W_r}{2}(v_L + v_R)\ cos\ \theta$$

$$\Delta y = \Delta t \frac{W_r}{2}(v_L + v_R)\ sin\ \theta$$

$$\Delta \theta = \Delta t \frac{W_s}{W_a}(v_R - v_L)$$

In deze formules is v_L de snelheid van het linkerwiel, v_R de snelheid van het rechterwiel, W_s de straal van het wiel, W_a de afstand tussen de wielen, en θ de richting van de robot. De eerste formule geeft dus de verandering van de x-positie (Δx) in functie van de verandering in tijd Δt . De twee andere formules geven soortgelijk de verandering in y-positie en richting in functie van de tijd.

Het is voldoende om deze formules te gebruiken als approximatie van hoe de echte robot werkt. In de praktijk zal je zien dat de bewegingen van de robot deze formules niet naleeft !

---

[1] http://42bots.com/tutorials/differential-steering-with-continuous-rotation-servos-and-arduino/

[2] http://planning.cs.uiuc.edu/node659.html

Programmeertaal

Het tweede onderdeel van het project bestaat eruit om een mini programmeertaal te maken om zo een robot te programmeren. Je bent vrij om zelf de syntax van je programmeertaal te bepalen en je mag deze maken in het Engels of in het Nederlands (maar wees wel consistent). Zoals reeds in de inleiding vermeld zal je verschillende instructies moeten maken zodat je op zijn minst de drie gevraagde programma's kan implementeren in jou programmeertaal.

We verwachten dus op zijn allerminst de volgende functionaliteit voor je programmeertaal:

1. *Variabelen*: In je taaltje zal het mogelijk zijn om de uitgelezen waardes van de sensoren bij te houden.
2. *Getallen*: Het uitlezen van de afstandssensor zal natuurlijk een getal als waarde hebben. Je zal dus in je taaltje getallen moeten ondersteunen.
3. *Booleans*: Om te beslissen of je naar links of rechts gaat moet je taaltje kunnen omspringen met booleans.
4. *Operatoren*: Je zal allerhande primitieve operatoren moeten voorzien, bijvoorbeeld (>,<,==, +, -, *).
5. *Loops* : De programmeertaal zal moeten toelaten om instructies in een loop te herhalen.
6. *Conditionals* : Je zal op zijn minst een if test moeten toevoegen zodat je kan beslissen dat de robot andere acties uitvoert afhankelijk van de staat van de sensoren.
7. *Sturen van de Motors*: Je zal de motors moeten kunnen controleren. Je mag zelf kiezen of je dat doet op een statische manier bijvoorbeeld: TURN_LEFT, TURN_RIGHT, FORWARD of meer generisch (Turn LEFT) (Turn RIGHT).
8. *Uitlezen sensoren:* De sensoren van de robot moeten kunnen uitgelezen worden. Voor de lichtsensor zal je een boolean of een "richting" moeten teruggeven. Voor de afstand zal je een getal moeten teruggeven die de afstand uitdrukt.
9. *Sturen van de leds*: De leds kan je programmeren in de verschillende kleuren. Je zal dus primitieven moeten voorzien om de leds een kleur te geven.
10. *Commentaar :* Zorg ervoor dat de programmeur code commentaar kan toevoegen in zijn code.

Rapportering

Naast je implementatie verwachten we ook een (verzorgd) verslag met de volgende structuur:

1. *Inleiding* : In je inleiding geef je een overzicht van je project en wat je verwezenlijk hebt. Geef ook aan op welke bestaande programmeertalen je eigen programmeertaal gebaseerd is.
2. *Syntax van de taal:* Geeft een overzicht van de constructies in je taal in (informele) BNF vorm.
3. *Semantiek van de taal*: Voor elk van je taalconstructies geef je een korte uitleg wat de taalconstructies doen en hoe je deze gebruikt.
4. *Voorbeelden programma's:* Geef volledige uitleg bij de programma's die je geïmplementeerd hebt in je eigen programmeertaal. Deze programma's moet je ook als aparte files indienen bij je project.
5. *Implementatie*: Geef een overzicht van de belangrijke punten van de implementatie. Refereer naar de lijnnummers in je code. Kleine stukjes code die heel belangrijk zijn kan je ook in je rapport plaatsen. Het is echter niet de bedoeling dat je verslag een kopie van je broncode is.
6. *Conclusie* : Geef een overzicht van wat je gerealiseerd hebt en hoe je de bestaande code eventueel nog zou kunnen verbeteren.
7. *Appendix Broncode*: Geef de volledige code van je project, zorg ervoor dat hierbij lijnnummers staan zodat je hier makkelijk naar kan refereren.

Niet Functionele vereisten

Naast de functionele vereisten zijn er ook een aantal niet functionele vereisten die dienen om na te gaan of je de concepten tijdens de hoorcolleges goed begrepen hebt. **Deze niet functionele vereisten zijn even belangrijk als de functionele vereisten.** Hoewel deze niet functionele vereisten normaal gezien zullen opduiken als je een goede implementatie van het project maakt, lijsten we deze hier expliciet op.

- *Gebruik van parser monad:* Om je taal te implementeren zal je beginnen van een tekst file, deze tekstfile zal je moeten parsen om zo de taal te kunnen uitvoeren. Voor het implementeren van deze parser verwachten we dat je gebruik zal maken van de parser monad. **Bestaande parser bibliotheken mogen enkel gebruikt worden als inspiratie voor je eigen bibliotheek.**

- *Gebruik van monad transformers:* Je zal tijdens de implementatie van je project zowel IO als State moeten gebruiken. Daarom verwachten we dan ook dat je gebruik zal maken van de monad transformer bibliotheek.

- *Code Kwaliteit:* Gebruik beschikbare tools om je code op te kuisen, gebruik "**hlint**" om de meest gebruikelijke bad smells uit je code te halen.

- *Commentaar:* Schrijf voldoende commentaar bij je code.

- *Code voorbeelden:* De drie gevraagde voorbeeld programma's moeten als een tekstfile meegegeven worden bij de rest van je code.

- Voor het tekenen van de robot moet je gebruik maken van de gloss bibliotheek (https://hackage.haskell.org/package/gloss). Uitzonderingen op deze vereiste zijn mogelijk maar dienen expliciet vooraf gecommuniceerd te worden met de assistent van het vak. Projecten die gebruik maken van andere bibliotheken zonder voorafgaande communicatie zullen dus niet aanvaard worden.

- Je simulator moet **multithreaded** zijn. Dit wilt concreet zeggen dat je simulator en het programma dat gebruik maakt van de simulator in een aparte thread draaien. Communicatie tussen deze twee programma's kan je implementeren door gebruik te maken van een `MVar`.