

AI

Kunstmatige Intelligentie (AI)

Robotica

Hoofdstuk 25 van Russell/Norvig = [RN]

voorjaar 2011 — College 4, 1 maart 2011

www.liacs.nl/home/kosters/AI/

Een **robot** is een “actieve, kunstmatige agent wiens omgeving de fysieke wereld” is. Het woord stamt uit 1921 (of eerder), en is gemaakt door de Tsjechische broers Capek. En **softbots**: Robocode, internet programma’s.

Van de science fiction schrijver Isaac Asimov (auteur van “I, Robot”) zijn de drie (later vier) wetten van de **robotica**:

1. Een robot mag een mens geen kwaad doen.
2. Een robot moet menselijke orders gehoorzamen (tenzij dat tegen 1. ingaat).
3. Een robot moet zichzelf beschermen (tenzij dat tegen 1. of 2. ingaat).



Een leuke robotsimulatie, van Michael Genesereth en Nils Nilsson, is de volgende.

Bedenk een taak, bijvoorbeeld een toren maken van een paar gekleurde blokken. Stel je nu een “robot” voor die uit *vier mensen* bestaat:

Brein krijgt input van Ogen, maar kan zelf niet zien; geeft opdrachten aan Handen; maakt plannen

Ogen weet niet wat het doel is

Handen, Links en Rechts voeren simpele opdrachten uit; ze zijn geblinddoekt



NASA Sojourner

Sony Aibo



Hiroshi Ishiguro's robot (en zichzelf)

Honda

Enkele begrippen uit de robot-wereld zijn:

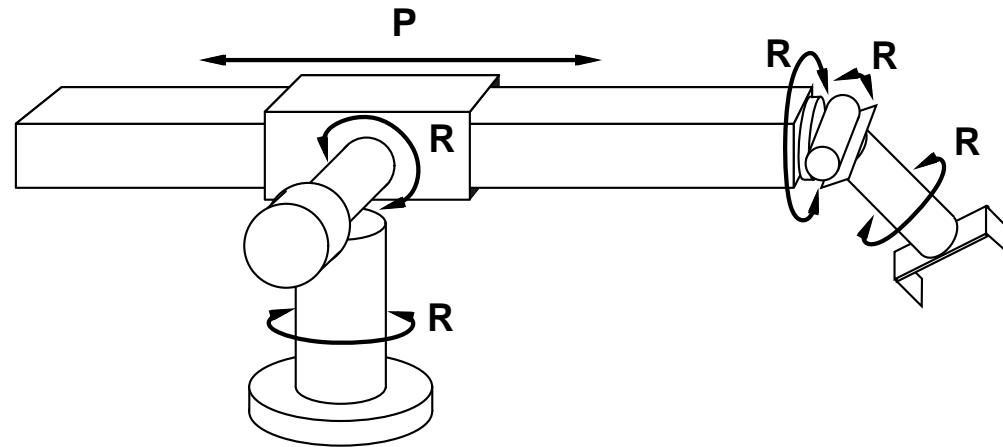
statisch stabiel stabiel zonder te bewegen

dynamisch stabiel robot verongelukt bij gedwongen rust

holonome robot totaal aantal vrijheidsgraden (onafhankelijke richtingen waarin robot of zijn “effectors” kunnen bewegen) = aantal controleerbare vrijheidsgraden
Voorbeeld niet-holonoom: auto (effectief 3: x, y, θ resp. 2: vooruit/achteruit, draaien); erger: met aanhanger

configuratie-ruimte bestaande uit bijvoorbeeld 6 hoeken voor een robot-arm

De **Stanford manipulator** ziet er als volgt uit:



P: prismatic joint; R: revolute joint

De configuratie van de robot wordt gespecificeerd door 6 getallen: dus 6 vrijheidsgraden (“degrees of freedom” = **DOFs**).

De arm van de mens heeft > 6 vrijheidsgraden (plaats hand op tafel en beweeg elleboog).

Een robot moet —in een 2-dimensionale wereld— van begin naar doel, weet precies waar hij is (GPS), en heeft een tastsensor. Een **online-algoritme** voor zo'n robot is:

- 1.** Bereken de rechte lijn ℓ van begin naar doel, en begin langs ℓ richting doel te bewegen.
- 2.** Als je botst: onthoud positie Q . Wandel met de klok mee om het obstakel heen, terug naar Q . Detecteer onderweg waar je ℓ kruist, en onthoud welk punt P het dichtst bij het doel is.
- 3.** Loop van Q naar P (met de klok mee of er tegenin, wat het kortste is), en ga via 2. zo verder.

Dit algoritme heeft een “competitive ratio” van ten hoogste $1 + 1.5B/|\ell|$ ($|\ell|$: afstand van begin naar doel via ℓ ; B : som van lengtes van randen van obstakels), de slechtst mogelijke verhouding tussen gevonden en kortste route.

Bewijs: de verhouding gevonden staat tot kortste is hooguit

$$\frac{|\ell| + 1.5B}{|\ell|} = 1 + 1.5B/|\ell|,$$

en je kunt die grens bereiken: leg een rafelig smal object met omtrek B op (en in de richting van) ℓ ; de kortste weg heeft dan lengte $\approx |\ell|$.

De **subsumption architectuur** is een idee van Rodney Brooks van MIT uit 1986, zie zijn artikel (via de AI-website).

Een robot bestaat hier uit verschillende *finite state machines* met klokken. Deze worden netjes gesynchroniseerd en gecombineerd. Zo vermijd je problemen met gigantische configuratieruimtes.

Het werkt goed voor één kleine taak, maar wat er gebeurt is soms lastig te snappen. Zie je hiervan iets terug bij Robocode?

Een simpel programma voor een Lego-robot is:

```
task main ( ) {  
    while ( true ) {  
        OnFwd (OUT_A + OUT_C);  
        Wait (Random (100) + 40);  
        OnRev (OUT_A);  
        Wait (Random (85) + 30);  
    }  
}
```



Dit laat de robot schijnbaar willekeurig rondlopen. En dat ziet er al heel intelligent uit!

Je kunt Lego-robots programmeren in Dave Baum's **NQC** (Not Quite C) met behulp van Mark Overmars' RCX Command Center, zie handleidingen op de websites.

```
#define MOVE_TIME 100
#define TURN_TIME 85
task main ( ) {
    while ( true ) {
        OnFwd (OUT_A + OUT_C); // motoren op A en B aan
        Wait (Random (MOVE_TIME) + 40);
        OnRev (OUT_A); // en die op A achteruit
        Wait (Random (TURN_TIME) + 30);
    }//while
}//main
```

Nadelen: maximaal 32 (integer) variabelen, maximaal 10 “parallele” tasks, alleen eenvoudige functies: subroutines (sub), inline functies (void) en macro's (#define).

A screenshot of the RCX Command Center software interface. The main window displays a code editor with the following text:

```
// Simple line follower
// =====
//
// Written by Mark Overmars
//
// This extra
// the black
//
#define SPEED          #define __NOTETIME 10
#define __WAITTIME 12

#define forward
#define turn

task checkLine
{
  while (true
    {
      PlayNote(262,4*__NOTETIME); Sleep(4*__WAITTIME);
      PlayNote(294,4*__NOTETIME); Sleep(4*__WAITTIME);
      PlayNote(330,4*__NOTETIME); Sleep(4*__WAITTIME);
      PlayNote(262,4*__NOTETIME); Sleep(4*__WAITTIME);
      PlayNote(294,4*__NOTETIME); Sleep(4*__WAITTIME);
    }
  }
```

An inset window titled "music.mxp" shows a message: "// Music file created by RCX Command Center." The bottom status bar of the application shows "1441: 1", "COM2", and "MindStorms".

Er zijn tastsensoren en lichtsensoren, met 3 aansluitpunten:

```
task main ( ) {
    SetSensor (SENSOR_1,SENSOR_TOUCH); // 0 of 1
    SetSensor (SENSOR_2,SENSOR_LIGHT); // 0..100
    SetSensor (SENSOR_3,SENSOR_TOUCH);
    OnFwd (OUT_A + OUT_C);
    while ( true ) {
        if ( SENSOR_1 == 1 ) ...
        else if ( SENSOR_2 > 40 ) ...
    }//while
}//main
```

Er kunnen ook meerdere sensoren op één input worden gezet, en ze kunnen preciezer (“raw”) worden uitgelezen.

```
task main ( ) {
  SetSensor (SENSOR_1,SENSOR_TOUCH);
  start check_sensors;
  while ( true ) {
    OnFwd (OUT_A + OUT_C); Wait (100);
    OnRev (OUT_C); Wait (85);
  }//while
}//main
task check_sensors ( ) {
  while ( true ) {
    if ( SENSOR_1 == 1 ) {
      OnRev (OUT_A + OUT_C); Wait (50);
      OnFwd (OUT_A); Wait (85); OnFwd (OUT_C);
    }//if
  }//while
}//check_sensors
```

Deze acties interfereren! Oplossing: of stop of semaforen:
until (sem == 0); sem = 1; /*motoractie*/ sem = 0;

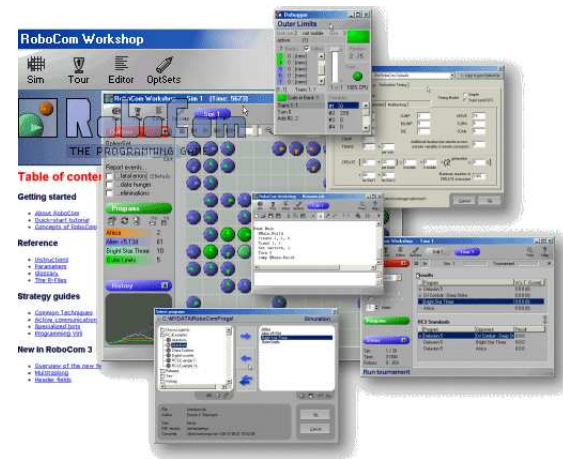
```
int level;
task geefsignaal ( ) {
    while ( true ) { SendMessage (0); Wait (10); }//while
}//geefsignaal
task checksignaal ( ) {
    while ( true ) {
        level = SENSOR_2;
        if ( SENSOR_2 > level + 200 ) { // snelle fluctuatie
            OnRev (OUT_C); Wait (85); OnFwd (OUT_A+OUT_C);
        }//if
    }//while
}//checksignaal
task main ( ) {
    SetSensor (SENSOR_2,SENSOR_TYPE_LIGHT);
    SetSensorMode (SENSOR_2,SENSOR_MODE_RAW); // 0..1023
    OnFwd (OUT_A+OUT_C);
    start geefsignaal;
    start checksignaal;
}//main
```

In werkelijkheid zit de “ruwe” waarde tussen 300 (licht) en 800 (donker).

Er zijn allerlei robot-simulaties.

Vroeger gebruikten we **RoboCom**, een opvolger van Core-War, gemaakt door Dennis Bemann, dat kleine robot-programma's laat vechten in een vierkant stuk computer-geheugen met 20×20 vakjes (= velden).

Software (RoboCom Workshop 3.1),
voor Windows XP en Linux:
<http://robocom.rrobek.de/>.

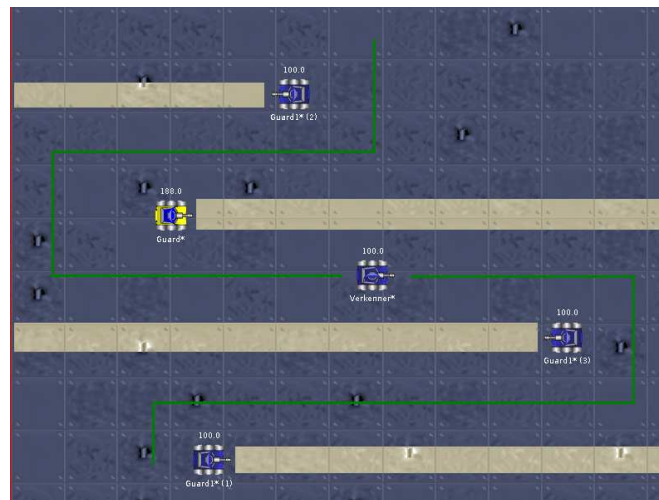


We gebruiken dit jaar **Robocode**, waarin kleine robot-Java-programma's vechten.

Software voor Windows XP en Linux:

<http://robocode.sourceforge.net/>

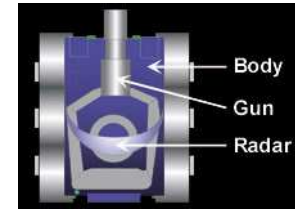
www.liacs.nl/home/kosters/AI/robocode.html



```
package man;
import robocode.*;
public class MyFirstRobot extends Robot {
    public void run() {
        while (true) {
            ahead(100);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }
    public void onScannedRobot(ScannedRobotEvent e) {
        fire(1); // eerst richten?
    }
}
```

Een robot heeft een *body*, een *gun* en een *radar*, die onafhankelijk van elkaar kunnen draaien.

Elke robot heeft een hoeveelheid energie.



In de `run()`-methode staat de hoofdbeweging van de robot. Hier kun je ook de kleuren aangeven.

De andere methoden worden door events “gekieteld”: het geheel is **event-driven**.

Mogelijke acties (= methoden) zijn onder meer:

`ahead(...)`, `back(...)`: ga zoveel pixels voor/achteruit
`turnLeft(...)`: draai robot zoveel graden; ook met `Right`
`turnGunLeft(...)`: idem: gun
`turnRadarLeft(...)`: idem: radar
`getHeading()`: oriëntatie robot (= ik)
`getBearing()`: oriëntatie andere robot
`getDistance()`: afstand tot andere robot
`fire(...)`: schiet met zoveel kracht

Zie verder de Robocode-website!

Mogelijke event(-handler)s zijn onder meer:

- `onScannedRobot(...)`: radar signaleert andere robot
- `onHitRobot(...)`: tegen andere robot gebotst
- `onHitByBullet(...)`: robot zelf geraakt
- `onHitWall(...)`: robot raakt muur
- `onWin(...)`: gewonnen!

Een voorbeeld:

```
public void onHitRobot(HitRobotEvent e) {  
    if (e.isMyFault()) { reverseDirection(); }  
}
```

Voor `TeamRobot`'s geldt dat ze met elkaar kunnen communiceren; zie verder de opgave op de AI-website.

Het huiswerk voor de volgende keer (dinsdag 8 maart 2011): lees **Hoofdstuk 4.1**, p. 120–130 van [RN] door. Kijk ook eens naar de vragen bij dit hoofdstuk.

Lees eens wat over **RoboCup**: voetbalwedstrijden voor robots, zie www.robocup.org.

Denk na over de tweede programmeeropgave; installeer Robocode, en speel met wat robot-programma's.

Deadline: **22 maart 2011**.

Extra vragenuren: maandagen, 13.45-14.30 uur, kamer 156a.