

Behavioral Organization Using the Utility Function Method

A Computer Experiment for the Course *Adaptive Algorithms and Intelligent Machines*

Mattias Wahde
2005-04-30

Introduction and background

In **behavior-based robotics** (BBR), the artificial brain of a robot is built from a repertoire of simple behaviors that are combined to generate the overall behavior of the robot. An important problem in BBR is the issue of **behavioral organization** (behavioral selection), i.e. the problem of making sure that a robot will activate the appropriate behavior in all situations. In the field of BBR, many different methods for behavioral organization have been developed. These methods can be divided into two main categories: **arbitration methods** and **cooperative methods**. In cooperative methods, the actions executed by the robot represent a weighted average of the suggestions given by many behaviors. By contrast, in arbitration methods, one (and only one) behavior is active (i.e. controls the robot) at any given time.

Most methods for behavioral organization rely on the ability of the user to set the parameters that determine the activation of different behaviors. Setting parameters by hand is possible for small behavioral repertoires, involving only a few behaviors. However, in more complex cases, it is not feasible to set parameters by hand, since (1) it is more or less impossible to anticipate all the different situations that the robot may encounter during its life time, and (2) even for known situations, the selection of behaviors (for activation) always involves a trade-off, requiring that one be able to compare the relative merits of many different behaviors in the situation at hand, which is a daunting task.

In the **utility function** (UF) **method**, the parameters determining the selection of behaviors are evolved (using an evolutionary algorithm (EA)), rather than set by hand. The method is described in detail on pp. 127-133 in the course book. In the UF method, each behavior is associated with a utility function, and the active behavior is always the one with highest utility. The user must provide a fitness function, but does not need to be concerned with the details of the behavioral selection procedure: The EA will optimize the utility functions so as to maximize the fitness of the robot.

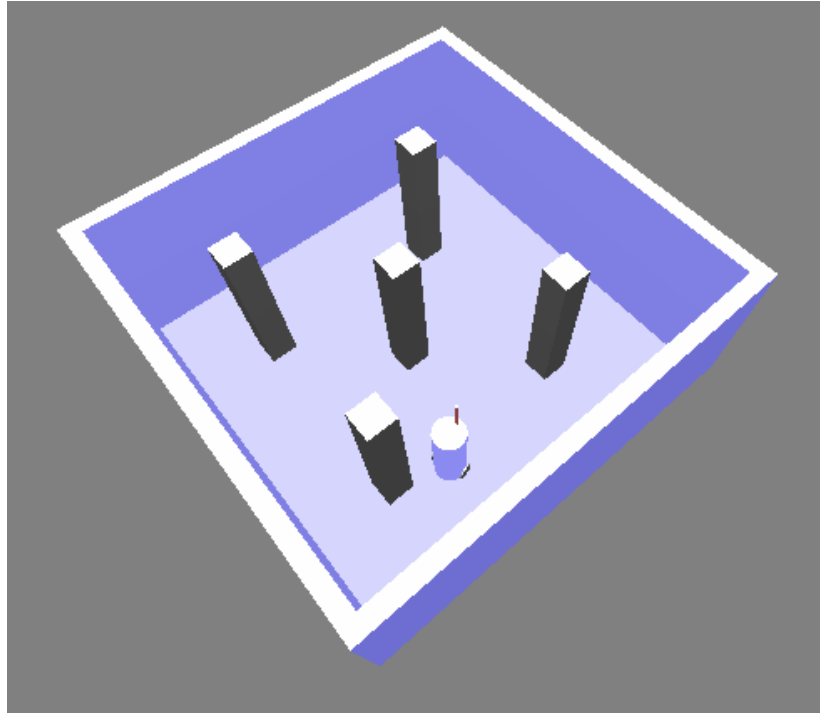


Fig. 1 The robot in its arena, consisting of a room with 5 pillars. The size of the arena is 5 x 5 meters.

Problem description

As indicated above, the UF method is useful in complex cases, involving large behavioral repertoires. In simple problems, involving only a few behaviors, it *is* possible to determine an appropriate selection of behaviors by hand. However, for simplicity and clarity, we shall nevertheless apply the UF method to a very simple case, involving only two behaviors.

Consider a robot moving in the arena shown in Fig. 1. The robot is a differentially steered, two-wheeled robot, equipped with a laser range finder (i.e. a laser-based sensor that can be used for measuring the distance along many (generally thousands) rays, between the robot and the nearest obstacle along the ray), and two DC motors. In addition, the robot is, of course, equipped with a battery.

In the problem considered here, the task of the robot will be to explore the arena as much as possible in a given time. The robot will be equipped with two very simple behaviors, namely *straight-line navigation* (B1), and *obstacle avoidance* (B2). In B1, the brain of the robot sends equal signals to the two motors of the robot, making the robot move in a straight line (after an initial transient, in case the robot was turning at the time of activation of B1). In B2, the robot will send signals of equal magnitude but opposite sign to the two motors, *until* the minimum distance (as measured by the laser range finder) to the nearest obstacle exceeds a certain minimum value (i.e. until the path in front of the robot is without obstacles). Clearly, these behaviors are oversimplified. In a realistic situation, a more advanced navigation behavior than simple, straight-line motion would of course be used. However, for the purposes of illustration, B1 and B2, as defined above, will be sufficient.

Now, since the task of the robot is to explore the arena, only the activation of B1 should contribute to its fitness (in B2, the robot will simply rotate in one place). Thus, for this simple case we can, for example, set the fitness function equal to the amount of time spent executing B1. However, with this simple fitness measure, the robot *may* find itself swapping rapidly between B1 and B2 when it is trying to turn to avoid collisions with obstacles. In order to avoid such unaesthetic dithering, the fitness measure is modified so that the robot obtains a fitness increase *only* if it executes B1 for a given, consecutive time dt (for which the arbitrary value 0.2s was chosen in this experiment).

With this fitness measure, the robot has a strong incentive to execute B1 as much as possible. On the other hand, it is also necessary for it sometimes to activate B2, in order to avoid collisions (the evaluation is terminated if the robot collides with a wall or with some other obstacle). Thus, the robot is faced with a behavioral selection problem, involving a trade-off between carrying out the assigned task (by executing B1) and surviving (by executing B2).

The selection of behaviors is based on utility functions. In the particular case used here, the utility function $U1$ for B1 is a p^{th} degree polynomial in the variables s_1 , s_2 , and s_3 , which are the (inverted) readings of the first, sixth, and 11th rays of a simplified laser range finder, with 11 rays. The inverted reading y of a ray in a laser range finder equals $R-z$, where R is the maximum range (4.0m, in this case), and z is the measured distance to the nearest object along the ray. Thus y will be in the range $[0,R]$. For B2, the utility function depends on two variables: s_{avg} , which equals the average reading of the 11 rays of the laser range finder, and x , which is an internal abstract variable (see the discussion in the course book, p. 128). The latter variable can be interpreted as the value of a hormone whose level is raised if the robot senses fear, e.g. as a result of an imminent collision. The use of the variable x is slightly simplified here: it takes the value 0 if B1 is active and the value 1 if B2 is active.

Thus, we have

$$U1 = U1(s_1, s_2, s_3). \quad (1)$$

$$U2 = U2(s_{\text{avg}}, x). \quad (2)$$

The task of the EA is to determine the coefficients of the polynomials $U1$ and $U2$, so that the robot can maximize its fitness, i.e. the amount of time spent in B1 during an evaluation of maximum length T (which, again, is interrupted prematurely in case of collisions).

Software

The Adaptive Systems research group at Chalmers Univ. of Technology has developed a software library called UFLibrary that implements the UF method, i.e. handles all issues concerning the evolution of utility functions etc. The remaining tasks of the user is to provide (1) the behaviors, in this case B1 and B2, (2) the fitness function, (3) an arena file, in this case the file AAIM2005Arena.txt in the folder ./Data/Arenas/, (4) a robot definition file, in this case divided into two parts, namely the file AAIM2005Robot.txt, specifying the body of the robot (located in the folder ./Data/Robots/) and AAIM2005RoboticBrain.txt, specifying the structure of the robotic brain (located in the folder ./Data/Brains).

The UFLibrary is a software library, rather than a stand-alone application. However, it contains several features (e.g. templates for graphical user interfaces) supporting rapid application development. Thus, once the tasks (1)-(4) above have been completed, the actual executable program can be generated rapidly.

The program for this experiment is available at

www.me.chalmers.se/~mwahde/courses/aaim/2005/AAIM2005_UFExample.zip

After downloading and extracting the program, the folder should look like this:

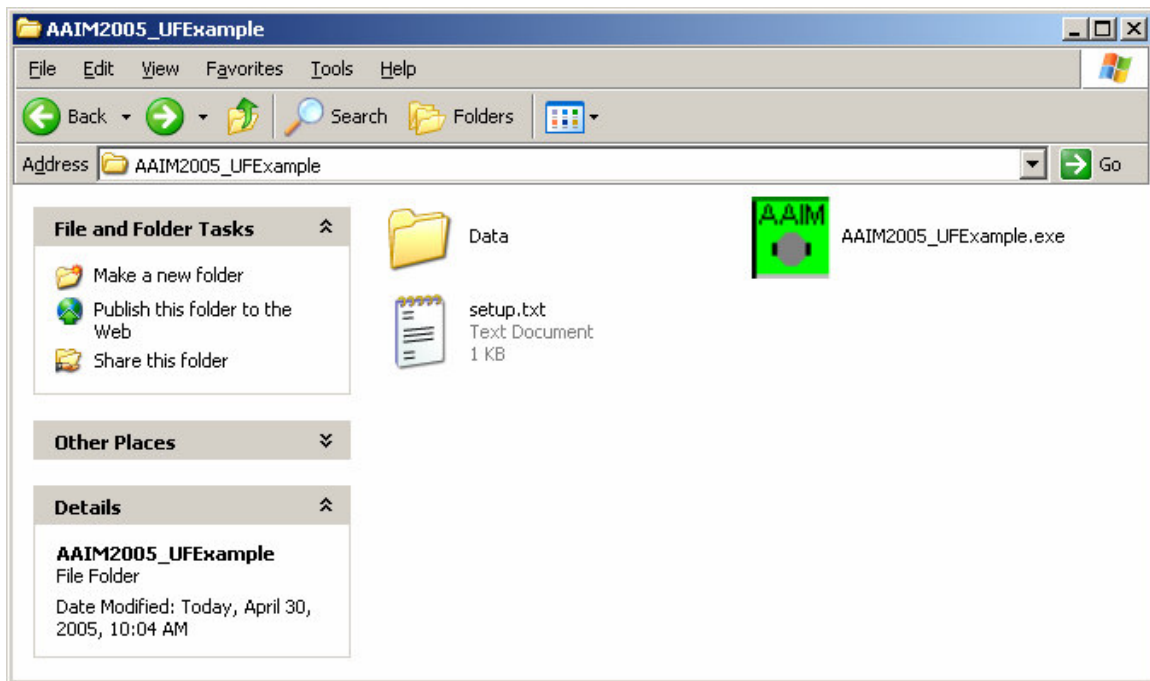


Fig. 2 The program folder for the UF example.

AAIM2005_UFExample is the executable file. setup.txt stores the path to the arena file and the robot definition file. Do *not* edit this file! The Data folder contains three folders, as shown in Fig. 3.

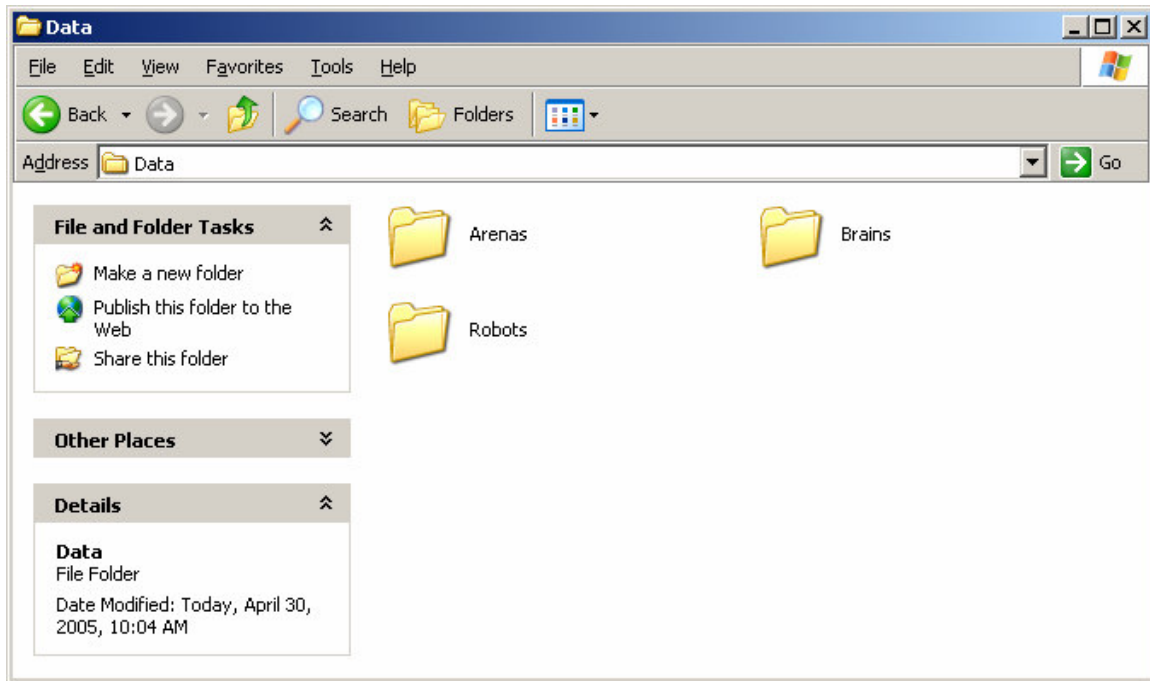


Fig. 3 The Data folder.

These three folders contain the arena file, the robot definition file, and the definition file for the robotic brain.

The arena file

The arena file (AAIM2005Arena.txt in /Arenas/) is a standard text file that can be opened using e.g. Notepad. The beginning of the file is shown in Fig. 4 below. Note that the arena file can be edited: you are welcome to do so, as long as you retain a copy of the original arena file, which should be used for the experiments below

For example, textures can be added to all objects, in order to make the arena look a bit nicer. One texture file, namely marbletiles.jpg has been included in the arena folder. By changing the line (line 9 in the text file) defining the texture for the floor object to read

```
Texture = marbletiles.jpg
```

the program will add this texture to the floor. (Note that extensive use of textures will require a lot of memory!)

The robot file

This file (AAIM2005Robot.txt in /Robots/) contains the definition of the body of the robot. Note that the robot is equipped with a non-discharging battery, appropriate for this simple example: In more realistic cases, the battery will, of course, discharge, forcing the robot to find a charging station and charge it (this requires that a battery charging behavior should be available, which is not the case in this simple example).

In addition to the body and the battery, the robot is equipped with two DC motors and one laser range finder. Again, you are welcome to edit the robot file, as long as the original file is used for the actual experiments below.

The brain file

The robot definition file reads the robotic brain from the file AAIM2005RoboticBrain.txt, located in /Brains/. In the brain definition file, the hormone (“fear”) defining the internal abstract variable used by B2 is introduced. Next, the two behaviors are given. In general, in the UF method, a behavior definition requires specification of the particular parameters used by the behavior (e.g. NavigationMotorOutput in the case of B1), the input variables (which can be used, if needed, by the behavior), and the state variables that are included in the utility functions. Do **not** edit this file.

```
# Generated by ArenaBuilder 20050428 10:42:56 (GMT+1)
object Object_0: TFloor
  Position = 0.000 0.000 0.000
  Velocity = 0.000 0.000 0.000
  Angle = 0.000
  Height = 2.500
  Mass = -1.000
  RGBColor = 0 0 40
# Texture =
  Length = 5.000
  Width = 5.000
  TileLength = 1.000
  TileWidth = 1.000
end
```

...

Fig. 4. The first lines of the arena file, showing the definition of the floor.

Experiments

We will now carry out a few experiments using the AAIM2005_UFExample program. Start the program by double-clicking the program icon. The following window appears:



Fig. 5. The main program window.

Next, select Runs. The window shown in Fig. 6 will appear. Provided that the setup.txt file has not been edited, the program will automatically select the appropriate arena file and robot definition file. It is also possible to browse to the location of each file. The parameters used for the experiment can also be set, namely the EA parameters, the evaluation length T (“Maximum simulation time”), the time step dt , as well as the degree p of the polynomials defining the two utility functions U1 and U2.

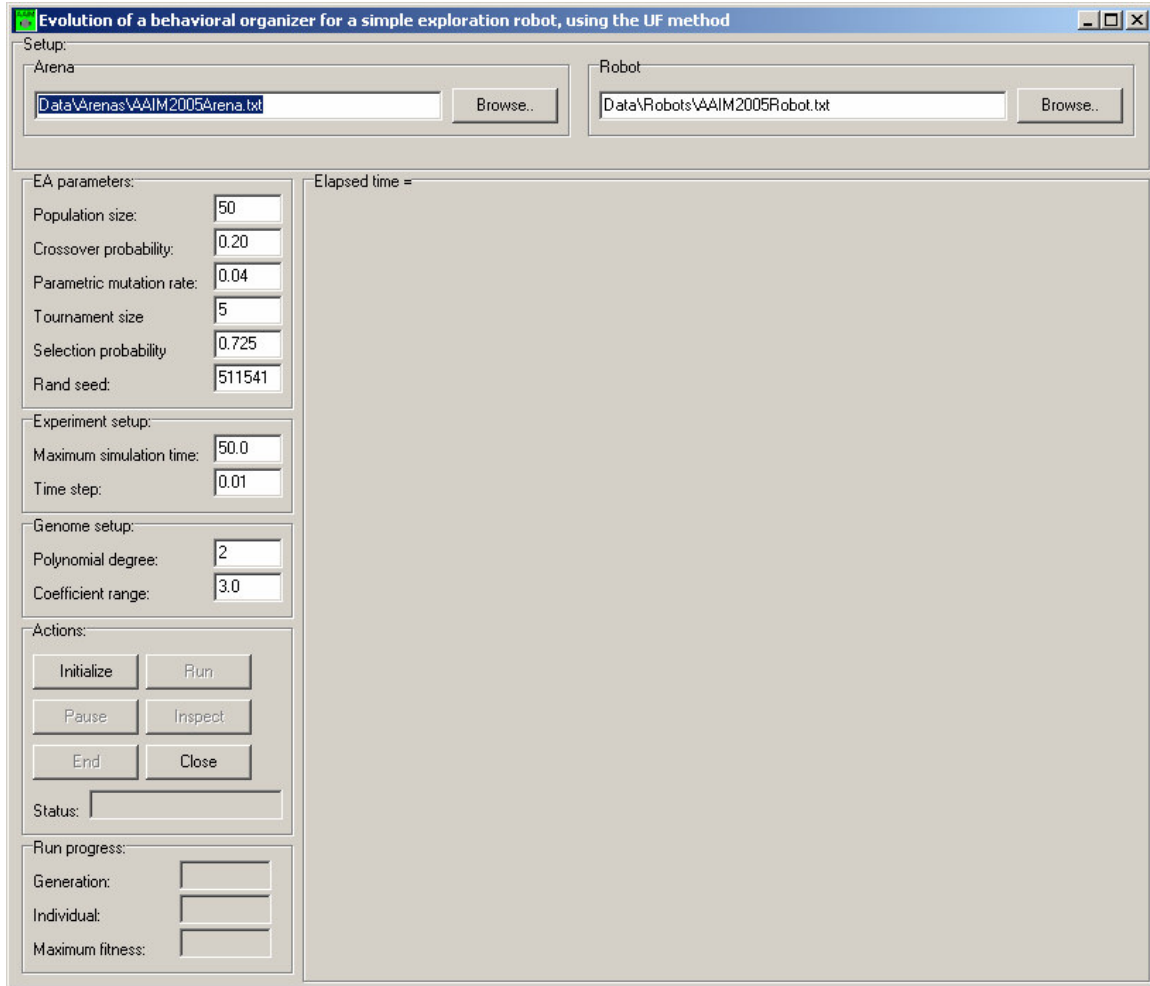


Fig. 6. The experiment window for the AAIM2005_UFExample program.

The Rand Seed parameter sets the seed value for the random number generator, allowing exact reproduction of a run (if the seed value is again set to the same value). Thus, in order to obtain different runs, the Rand Seed value should be changed. Note that the actual behaviors, B1 and B2, are *deterministic*: The choice of the Rand Seed parameter only affects the initialization of the chromosomes used in the EA, as well as the selection and mutation procedures.

When the parameters have been set, the user should press **Initialize**, and then **Run**. In order to inspect the current best individual, press **Pause** and wait for the current generation to be completed. When it is, the **Inspect** button can be pressed. Finally, to end a run, press **End** and then **Close**.

During a run, the program generates some output files. Try running the program for a few generations before pressing **Pause** and then **End**. A new folder, containing the date and time at which it was generated will appear. This folder contains three files: a run summary, the maximum fitness and the average fitness. The fitness values can be plotted using Matlab. However, they are not supplied in standard Matlab format, and therefore a special reading procedure is needed. Start Matlab and move to the correct directory (i.e.

the one containing the three text files just mentioned). Next, use the Matlab command **dlmread**, as shown in Fig. 7. The first parameter used by dlmread is the name of the file, and the second is the delimiter (“ ”, i.e. “space”, in this case). The third parameter is the row at which dlmread should start reading (row 0 contains a comment, so the reading should start at row 1), and the fourth parameter is the column at which dlmread should start reading.

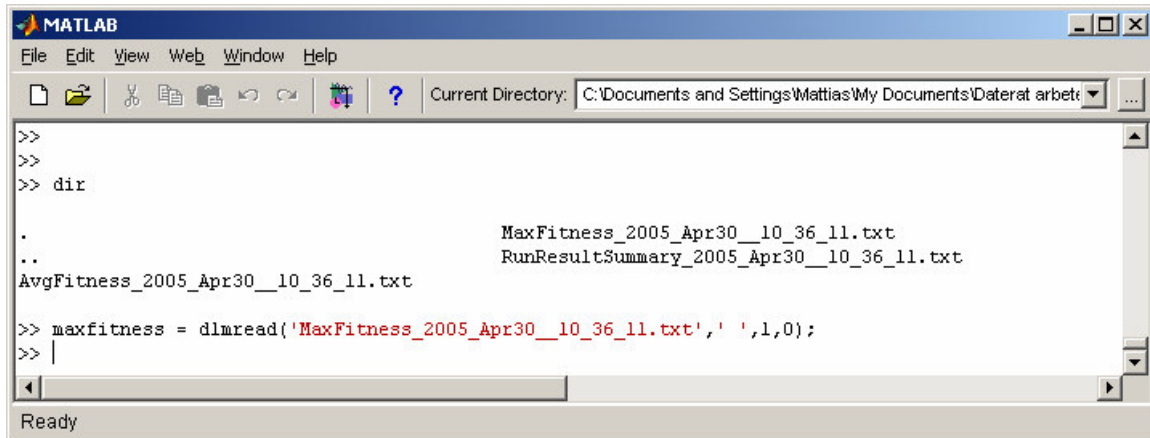


Fig. 7. Using dlmread in Matlab.

Next, the maximum fitness (as a function of generation) can be plotted, e.g. by typing

```
>> plot(maxfitness(:,1), maxfitness(:,2))
```

in the Matlab command window. An example is shown in Fig. 8.

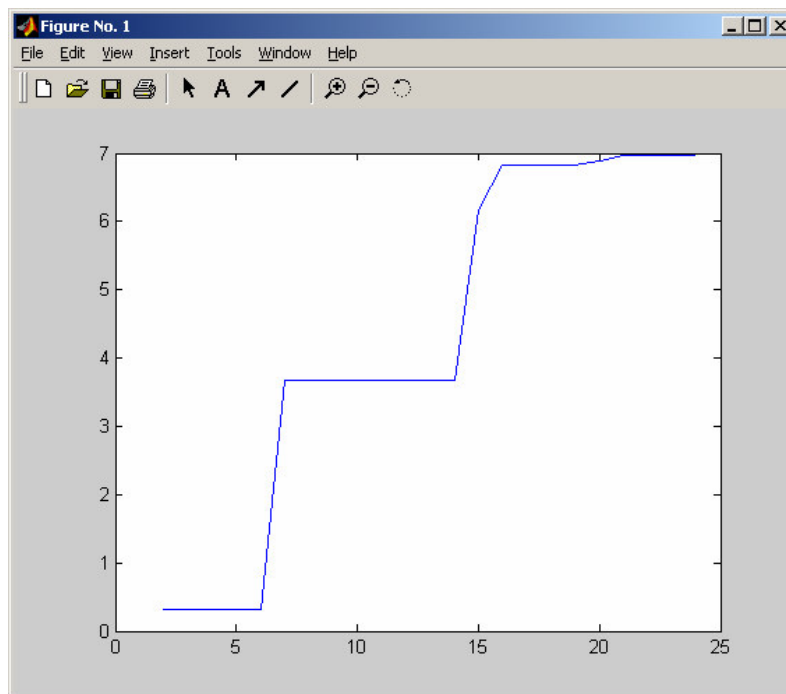


Fig. 8. Plot of the maximum fitness as a function of generation, for a short run (25 generations).

Now, run the program again (by pressing **initialize** and **run**). After a few generations, press **Pause**, and then **Inspect**. An animation of the best robot found so far (by the EA) is shown (see Fig. 9 below). While the animation is running, it is possible to pan and zoom to get a different view of the arena.

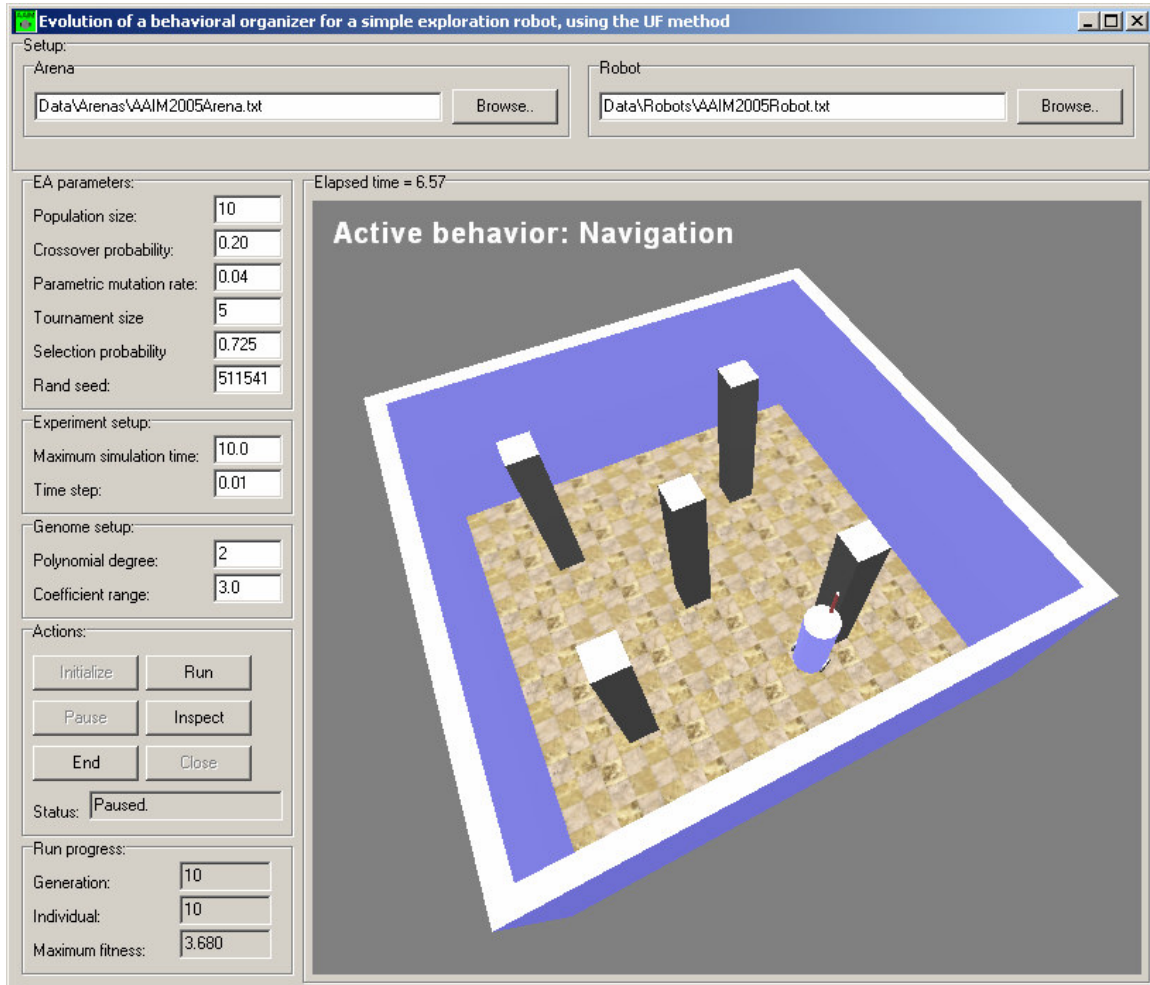


Fig. 9 Inspecting a robot. Note that a texture has been added to the floor.

Now, stop the program by pressing **End**, and open the new output folder generated by the program. In this case, the folder will contain another folder, named *Robot_Gen_N*, where *N* is the generation at which the inspection was carried out. Open this folder: it contains four files. The *Robot_Data* file contains the utility functions for the best robot (note that the variables are all denoted x_i in this file, regardless of their true name (as given in the brain definition file)). The other files contain the utility values, the coordinates (and angle of heading), and battery level of the robot. Note that the battery level remains fixed at 1.00 in the experiments considered here.

Using the utility and coordinate files, the variation (with time) in utility (for both behaviors) and the motion of the robot can be plotted. Start Matlab and move to the correct directory (i.e. the one containing the four files just mentioned). Again, use the `dload` command. Thus, write

```
>> u = dlmread('UtilityValues_Gen9.txt',' ',1,0);
```

(Note replace “9” above with the generation number in your file). Next, plot the two utility functions, e.g. by writing

```
>> plot(u(:,1),u(:,2),'-',u(:,1),u(:,3),':')
```

An example showing the variation of the two utility functions (with time) is shown in Fig. 10.

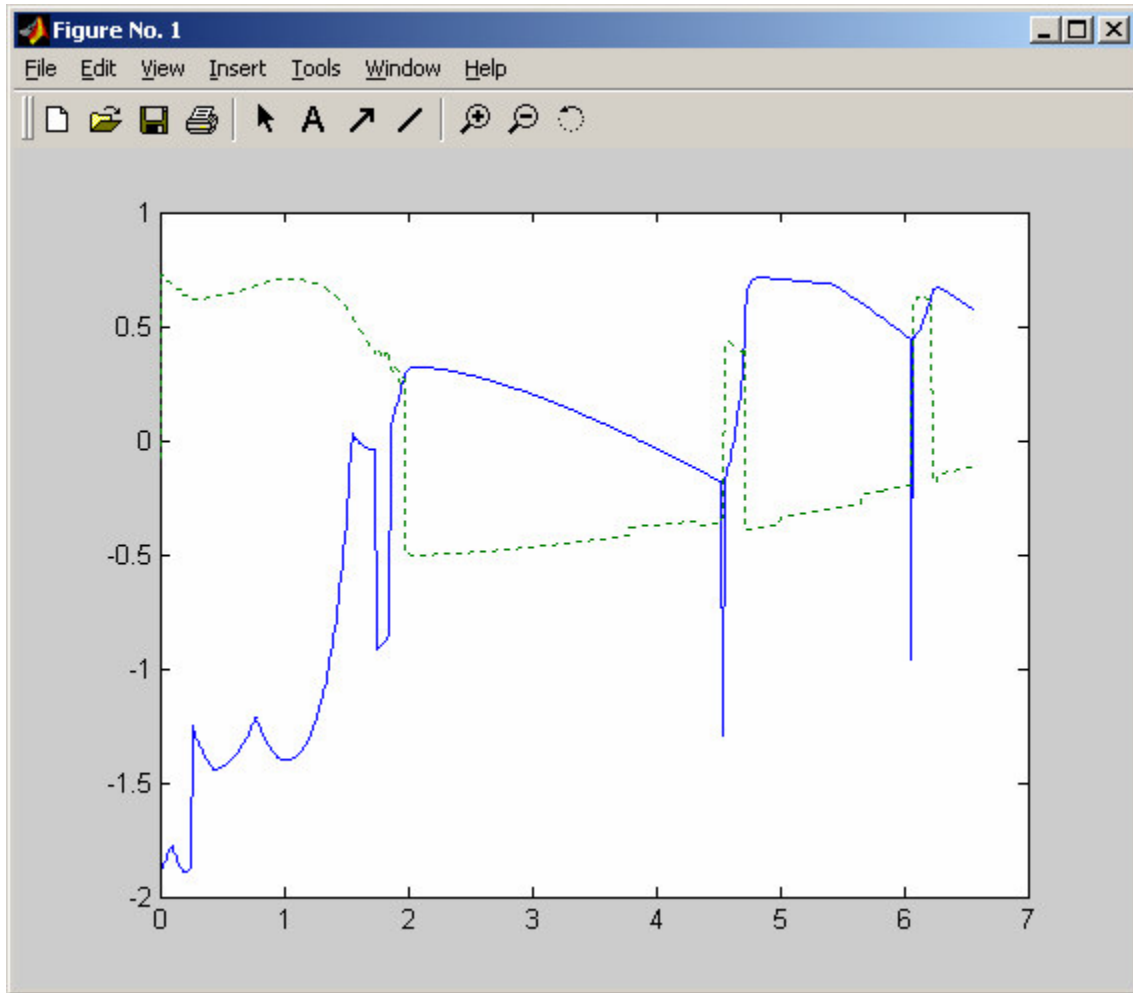


Fig. 10 Variation of the two utility functions. U2 is shown as a green dotted line. In this case, B2 was active for the first 2 seconds, and B1 was active between $t=2$ s and $t = 4.5$ s (approximately), at which point B2 was activated briefly again etc.

Final experiment

Now, having tried the program a bit, you should carry out a few long runs. In these runs, keep the maximum evaluation time and the time step fixed at 50.0, and 0.01, respectively. All other parameters, e.g. the population size, the mutation rate, the polynomial degree p etc. may be varied. Your task is to find the best possible individual. Thus, when you have completed your runs, make a zip file of the output folder from the best run. The compressed file should contain

- (1) The files RunResultsSummary, AvgFitness, and MaxFitness.
- (2) The folder containing the evaluation of the best individual in the final generation of the run (and no other folders!)

(**Note:** Maximum allowed size of the compressed file: 1 Mb). Send the compressed file via e-mail to mattias.wahde@me.chalmers.se, together with your name and ID number.

It is possible to record the motion of the robot (during inspection) as an AVI file. To do so, right-click in the Actions group box after pressing **Pause**, select **Record AVI**, and follow the instructions. Note, however, that (unless compression is used) the resulting file will be HUGE (a 50.0s evaluation, uncompressed, requires > 1Gb storage). Do not send AVI recordings together with the zip file.