A study of multiple behavior implementations in connection with the utility manifold method for behavioral organization

An investigation of the properties of the utility manifold method for behavioral organization

Hans Sandholt, Mattias Wahde
Department of Machine and Vehicle Systems
Chalmers University of Technology
412 96 Gteborg, Sweden
E-mail: {hans.sandholt, mattias.wahde}@me.chalmers.se

***Abstract*— In this paper, the performance of the utility manifold (UM) method for behavioral organization is investigated. The method is applied to a case involving strongly non-trivial selection between four different behaviors, in order to generate an overall task of navigation for a simulated wheeled robot. The results of the investigation show that the UM method was easily able to achieve the overall navigation task, by generating appropriate selection between the four constituent behaviors.**

A desirable property of any method for behavioral organization is the ability to organize different behaviors regardless of their specific implementation. This property is investigated for the UM method, by testing it against two different versions for each of the four constituent behaviors, i.e. a total of 16 different combinations.

***Index Terms*— Behavior-based robotics, behavioral organization, utility manifold method, evolutionary algorithms**

I. Introduction

Behavioral organization, i.e. the problem of determining when behaviors should be active, is one of the central problems in behavior-based robotics. Several methods have been suggested for solving this problem, starting with the pioneering subsumption method [1].

A drawback with most methods of behavioral organization is that they rely heavily on the ability of the user to set parameters by hand. In all but the simplest cases, it is very difficult to anticipate all situations that the robot may encounter, particularly if it is designed to move in an unstructured environment. Recently, Wahde [2] introduced an alternative method, the utility manifold (UM) method, in which the behavioral organization system is based on utility functions that are evolved rather than designed by hand. In the UM method, the user need only specify fitness functions for so called task behaviors, i.e. behaviors directly related to the task of the robot. For auxiliary behaviors, such as obstacle avoidance and battery charging, no fitness functions need be assigned. The method does have certain limitations: For example, it does not, as yet, handle situations in which an explicit memory is needed (as for example, in cases where the robot is interrupted in a behavior that can only be restarted successfully if the state of the robot is stored (and thus recoverable) at the time of exit. However, work is presently underway to include such behaviors as well. So far, the method has been tested in abstract behavioral combination tasks where, in some cases, the optimal behavioral organization could be derived analytically [2], and in a task involving locomotion of a simulated legged robot [3].

The aim of this paper is to expose the method to more stringent tests and analysis, involving the organization of four behaviors for a wheeled robot. In addition, the generality of the method, i.e. its ability to organize behaviors regardless of their specific implementation, will be investigated.

The outline of the paper is as follows: in Sect. 2, the UM method is briefly reviewed. In Sect. 3 the simulator is presented, and in Sect. 4 are presented the results concerning the generality of the method and its scaling properties. The results are discussed in Sect. 5.

II. The utility manifold method

The utility manifold (UM) method [2] addresses the need for a general, i.e. widely applicable, method for behavioral organization that requires a minimum of parameter-tuning by the user. In the UM method, each behavior is assigned a utility function which contain the desires and beliefs of the robot. The method is an arbitration method, i.e. one in which only a single behavior is active at any given time. The active behavior is simply chosen as the behavior with the highest utility value. Thus, the main problem is to determine the exact shape of the utility functions. In the UM method, whose central concepts will not be outlined briefly, the optimization of utility functions is performed using an evolutionary algorithm. For a more thorough introduction to the UM method, see [2] and [4].

A. Biological background

In the development of the UM method, ethological considerations played a central role. The concept of *utility* provides a common currency for rational agents when they select which behavior to perform [5]. Indeed, the concept of utility maximization follows from the property of *transitivity of choice*, which, in turn, underlies all rational behavior. Thus animals, who are highly adapted to their environment, tend to behave as if they were maximizing a quantity which we may call utility (even though, in most cases, and especially in simpler animals, it is likely that the maximization of utility is something which is performed unwittingly and as a result of evolutionary design). Wahde [2] stresses the importance of considering the highly optimized capacity for behavioral selection in animals when trying to emulate this ability in robotics. The problem of behavioral selection has been studied intensively in ethology [5] and a few authors (e.g. McFarland and Spier [6], McFarland and Bösser [7] have considered the use of utility functions in robotics applications. However, the UM method is the first approach in which utility functions are constructed quantitatively using evolutionary optimization.

B. Behaviors and fitness

The UM method is concerned with behavioral organization, not with the generation of individual behaviors. In fact, the method is intended to be sufficiently general to be able to organize behaviors no matter how they were generated. In the UM method, behaviors are divided into two categories, *task behaviors* which are directly related to the task of the robot and which give it a fitness increase if performed successfully, and *auxiliary behaviors* which may be useful or even essential (and thus associated with high utility), but which give no fitness increase. Thus, the designer of the robot should only be required to provide fitness functions for the behaviors that are related to the task of the robot, and not to its auxiliary behaviors (such as e.g. obstacle avoidance and battery charging), whose activation instead should be determined indirectly through the optimization of the utility functions.

C. State variables and utility functions

With each behavior is associated a utility function (not to be confused with the fitness functions provided for task behaviors, see the example below), which depends on (some of) the state variables. State variables, in turn, are divided into three categories: external variables (e.g. readings of IR or visual sensors) that measure anything the robot can derive directly from the environment, internal physical variables (e.g. battery levels) that measure physical properties such as temperature or energy levels within the robot itself, and, finally, internal abstract variables, which are used in the behavioral selection (see Sect. VI below), and which roughly correspond to signaling substances (e.g. hormones) in biological systems [2].

D. Evolutionary optimization

In the UM method, the optimization of utility functions is normally performed using evolutionary algorithms (EAs). In general, the utility functions depend on several state variables, and should provide appropriate utility values for any combination of the relevant inputs. Thus, determining the exact shape of the utility functions is a formidable task, and one for which EAs are very well suited. In principle, genetic programming (GP) can be used, in which case any function of the state variables can be evolved. However, it is often sufficient

to make an ansatz for the functional form of each utility function, and then implement the EA as a standard genetic algorithm (GA) for the optimization of the parameters in the utility function. The latter approach will be used in the paper. The ansatz for each utility function is given in Subsect. V-C.

Thus, once the state variables have been identified, and an ansatz has been made for each utility function, the EA can begin the process of shaping the utility functions in such a way that the choice of behaviors becomes as good as possible.

E. A simple example

The UM method will now be illustrated by means of a simple example. For a more thorough introduction see [2]. Consider the simple example of a cleaning robot equipped with two behaviors: one task behavior *sweep floor* (B1) and one auxiliary behavior *charge batteries*[1] (B2). Clearly, from a user's or owner's point of view, the floor sweeping behavior is the relevant one, i.e. the behavior one would wish the robot to perform continuously if it were possible. Thus, a fitness function (f_1) is assigned to this behavior. For example, the robot could be given an additional fitness point for each square meter of (dirty) floor that it cleans. B2 gives not fitness, i.e. $f_2 \equiv 0$. Furthermore, each behavior is associated with a utility function, denoted U1 and U2 for B1 and B2, respectively. The utility functions depend on the state variables of the robot, which in the case of the cleaning robot may include readings of IR sensors (collision detectors), the battery energy level, some internal abstract variables[2] etc. When the behavioral organization system of this robot is generated, the two utility functions are evolved, either from completely random functions of the state variables or from some ansatz. The feedback signal to the optimization procedure is the fitness, which, as described above, only is given for B1. How, then, can B2 be activated? Consider the case in which B2 is *not* activated at all. In such a case, the robot would sweep the floor until it ran out of battery energy, and would then be unable to continue (and also unable to gain additional fitness). However, the activation of behaviors is governed by the utility functions. Thus, if instead the evolutionary algorithm designs the utility functions such that U2 sometimes exceeds U1 (e.g. when the battery level is low and the robot is near a charging station), the robot would charge its batteries and thus be able to resume its floor-sweeping activities after some time. Thus, while the fitness f_1 is the optimization measure, and B2 gives no fitness increase, the evolutionary optimization method will nevertheless design the utility functions so that B2 is sometimes activated (at least if the evaluation time exceeds the time that the robot can operate on a single battery charge).

[1] A battery charging behavior would normally also have to include a sub-behavior for *finding* a charging station. However, for simplicity, such complications are neglected in this example, which is only intended to describe the basic concepts of the method.

[2] The use of internal abstract variables is described further in Subsect. V-C.

F. Procedure

The main step in the use of the UM method is the generation of the behavioral organization system through the definition of utility functions for each behavior.

The constituent behaviors used by the UM method can be generated by any means desired. For example, behaviors can be either hand-coded or evolved.

In this paper, both the generation of individual behaviors and the subsequent evolution of the behavioral organization system are done in a simulated environment, which will now be described in detail. Thereafter, the detailed properties of the various constituent behaviors are described, as well as the simulation procedure used in connection with the UM method.

III. Simulator

Mathematical models and simulations are mostly a simplification of the real-world still they gives a good indication of the expected outcome of the simulated properties. Using simulations instead of real-world tests lead to lower cost and faster evaluations of the investigated properties.

When applying a solution from a simulation to a real-world situation there is commonly a problem with discrepancies between the models and reality, called the reality gap [8]. However, this problem will not be further investigated in this paper.

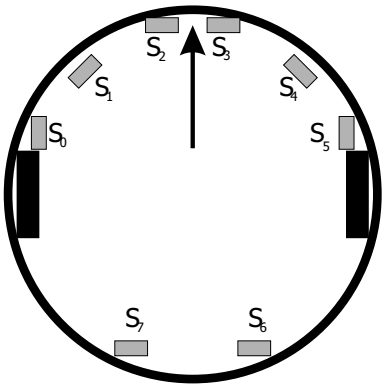

Fig. 1. Sensor and motor layout of the modelled Khepera robot. Sensors are shown as gray rectangles and motors as black.

A good simulator should be based on accurate descriptions of the used objects (e.g. robots, sensors, motors) and proper description of the environment and interactions during the simulation. These descriptions are now addressed.

A. Simulated robot

A mathematical model of a Khepera robot[3] shown in Fig. 1, equipped with two speed-controlled wheels, and eight IR sensors, six in the front and two in the rear is used. The IR sensors are used as proximity sensors.

The motion of the differentially steered robot is governed by the equations

$$M\dot{v} + \alpha v = A(\tau_L + \tau_R) \quad (1)$$

$$I\ddot{\varphi} + \beta\dot{\varphi} = B(-\tau_L + \tau_R) \quad (2)$$

where v and $\dot{\varphi}$ are the curvilinear and rotational speeds of the robot, respectively. τ_L and τ_R are the torques acting on the left and right wheel, respectively, and M and I are the mass and moment of inertia of the robot, respectively. A and B are scale factors depending on the geometrical properties of the robot.

1) Motors: A simplified DC-motor model (no induction modelled) was developed. The torque acting on a wheel is modelled as

$$\tau_i = k_m(\frac{u_i}{R} - \omega_i k_a) \quad (3)$$

where R, k_m, k_a are the estimated resistance, torque-, and the back-emk constants for the motor, respectively and τ_i, u_i, and ω_i are the torque on the wheel, applied potential, and angular speed of motor i, respectively.

The model also includes two speed-controllers (PI-type), one for each wheel. The PI-controller is described as

$$u_i = k_p\omega_i + k_I I_i \quad (4)$$

$$I_i = \int(\omega_i - r_i)\delta t,$$

in discrete form becomes

$$I_i(t+\delta t) = I_i(t) + (\omega_i(t) - r_i(t))\delta t \quad (5)$$

where k_p and k_I are the proportional and integral constants of the controller, respectively and I_i and r_i are the integral state and motor-speed reference value, respectively. The integral state I_i is updated for each time-step.

The speed-controller parameters are set so that the simulated step-response is similar to the real step-response, as measured by Byung et al. [9].

In the simulation the range of the wheel-speeds are normalized to $[-1, 1]$.

2) Sensors: For a relative simple but accurate proximity sensor model is an obstacle within the sensor range divided into N parts. The number N depends on the complexity of the obstacle. Contributions from each obstacle-part is summed according to

$$S_i = \sum_{j=0}^{N} \frac{\alpha_{i,j}}{\Theta_i}\left(1 - \frac{d_{i,j}}{D_i}\right)^2 \quad (6)$$

where $\alpha_{i,j}$ is the j:th part of an obstacle angular coverage of sensor i, Θ_i the aperture of sensor i, both in radians, $d_{i,j}$ is the distance from the sensor to the center of obstacle-part j, and D is the sensor range. In Fig. 2 are these variables visually described. The square-term in the expression is due to a quadratic fall-off with distance of the sensor reading for this sensor type. Note that the far-end of the sensor range is simplified to a straight line instead of a cordial boundary.

The range of the sensor values are normalized to $[0, 1]$.

3) Battery: The range of the battery level is normalized to $[0, 1]$ and the level is updated each time-step according to

$$E(t+\delta t) = E(t) - c_E\delta t, E \in [0, 1] \quad (7)$$

where c_E is the discharge-rate. This equation apply when the battery is not charging.

[3]The Khepera robot is manufactured by K-team, www.k-team.com

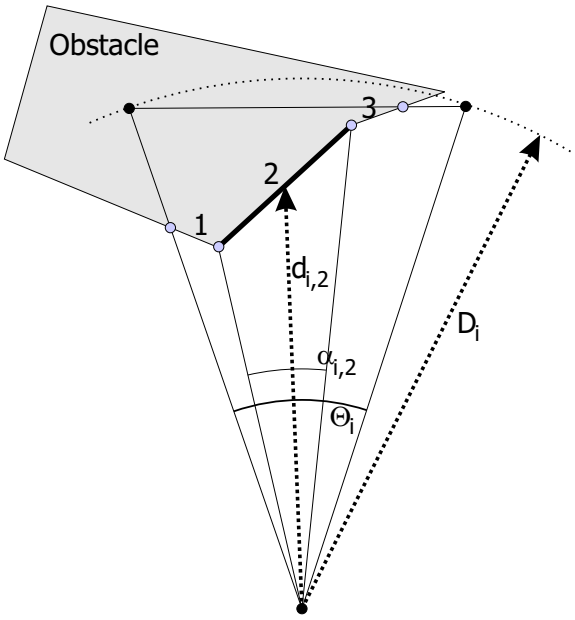

Fig. 2. Proximity sensor variables. The sensor range is a triangle, and the obstacle is divided into three parts due to its complexity. The figure shows the parameter case for obstacle part 2.

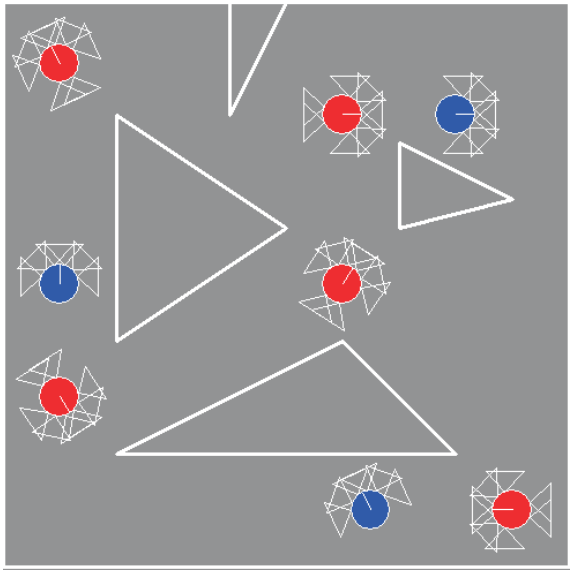

Fig. 3. Simulation arena with fixed and moving obstacles. The robots are equipped with two rear sensors while the moving obstacles are not.

B. Simulated environment

In the simulation environment, shown in Fig. 3, robots as well as fixed and moving obstacles are defined. The environment is equipped with walls along each edge, i.e. periodic boundary conditions are *not* used. The moving obstacles are equipped with front sensors, giving them the possibility to avoid stationary obstacles.

In all simulations described below, the stationary obstacles were placed as in Fig. 3.

Usually, during simulations, several robots were active simultaneously in the environment. The exact simulation procedure will be described in Sect. V below.

IV. Constituent behaviors

The UM method sets no restriction on the implementation details and manner of generation of individual behaviors. Thus, in order to illustrate the ability of the UM method to organize many different sorts of behaviors, different implementations of the behaviors will be used.

The task for this evolved robotic brain is to drive the robot as far as possible in the environment without colliding with any obstacles or running out of battery energy. Therefore four different behaviors are identified and implemented.

The behaviors defined for the simulation are *wandering*, *obstacle avoidance*, *battery charging*, and *robot scouting*. The *wandering* behavior is the task behavior while the others are auxiliary behaviors.

No sensor input is used by any behaviors, except *obstacle avoidance*, i.e. the motors are set disregarding any sensor readings since the behavior organizer selects appropriate behavior.

The description of the behaviors are as follows:

a) Wandering: The *wandering* behavior is implemented in the simplest possible way, simply moving the robot in piece-wise straight paths. Two different implementations were used, namely *straight-line wandering* (denoted B1.1), in which the motors are set to the same speed making the robot move in a straight line, and *drunkard's walk* (denoted B1.2), in which the robot moves in a straight line for a random period of time, and then turns to a new random direction starting a new straight-line motion etc.

b) Obstacle Avoidance: The behavior is responsible for navigating the robot away from any obstacle in the close vicinity that compromises a safe passage.

Three implementations are provided for this behavior, namely *rotate away and stop* (denoted B2.1) where the robot rotates away from an obstacle and stops the motors when the front-sensor values fall below a certain value, *rotate*

away and recede (denoted B2.2) is similar to the *rotate away and stop* implementation but allows the robot to set full speed backward or forward if an obstacle is present in front of or behind the robot, respectively, and *grazing robot behavior* (denoted B2.3) a behavior used in this paper by the moving obstacles. This last behavior is a wander and obstacle avoidance behavior where normally the left and right motors at set to slow forward speeds resulting in a slightly curved path. If an obstacle is close the motors are set to mainly rotate away from it resulting in a forward "wiggling" motion.

c) Battery Charging: This behavior is responsible for recharging the batteries. The robot is considered to be equipped with foldable solar cells, thus is the robot standing still during charging. Two implementations are used for this behavior, namely *exponential charging* (denoted B3.1) where recharging is controlled by $E(t+\delta t) = E(t)(1+c_e\delta t)$, where c_e is the charge-rate for this implementation and, *delayed linear charging* (denoted B3.2) where recharging is delayed a few time-steps before proceeding at a linear rate, $E(t+\delta t) = E(t) + c_e\delta t$, where c_d charge-rate for this implementation.

d) Robot scouting: : This behavior makes the robot search its closest vicinity for other robots that might collide with it from a blind angle. Two implementations are provided for this behavior, namely, *rotational scouting* (denoted B4.1) where the motors are set to fixed speed with opposite signs to make the robot executing a pure clockwise rotation while scanning for other robots, and *seeking scouting* (denoted B4.2) using a hand-coded function describing a simple search pattern where the motor speeds are set according to $\{\omega_1, \omega_2\} = \{\sin c_{s1}t_i, -\cos c_{s2}t_i\}$, where t_i is the behavior time, and c_{s1}, c_{s2} are constants determining the search pattern.

V. Simulation procedure

The main part of the simulation procedure used in this paper consists of evolving behavioral organizers using the UM method. In addition, the simulator allows evaluation of evolved behavioral organizers. The UM method is based on an EA (see e.g. [10] for a thorough description of EAs), the basic flow of which is shown in Fig. 4. As all EAs, the optimization procedure acts on a population of individuals that are assigned fitness values based on their performance during evalutions, the flow of which are shown in Fig. 4. In the following subsections, the EA and the evaluation procedure for individuals will be described briefly. Next, the specific form used for the utility functions will be discussed, and the section is concluded with a description of the fitness measure used in the EA.

A. Evolutionary algorithm

The chromosomes appearing in an EA used in connection with the UM method encode the utility functions on which behavioral selection is based. In the investigation reported here, a specific ansatz for the utility functions, described in Subsect. V-C, is used. Thus, when decoded, the information contained in the genes of a chromosome is used for determining the exact shape of the utility functions for the individual in question.

At the start of an EA run, a population of N chromosomes in initialized randomly. Once a chromosomes has been decoded, the corresponding individual is tested and a assigned fitnes values (see below) based on its performance. When all individuals have been tested, the next generation is formed using the procedures of tournament selection, crossover, and mutation. Elitism is used, i.e. a single exact copy is made of the best chromosome, and it is transferred unchanged to the next generation. The EA runs for a maximum of G generations, unless a pre-specified fitness threshold is reached, in which case the simulation is terminated.

B. Evaluation of individuals

During evaluation, the chromosome is decoded, forming the brain of an individual (i.e. a simulated robot) that is allowed to move in the environment, performing various actions based on the implemented behaviors.

While the stationary part of the environment (e.g. the walls and the stationary obstacles) remain the same in all runs, see Fig. 3, the environment also contains moving obstacles. At initialization, the moving obstacles are always placed at pre-specified initial positions, and with a pre-specified heading. However,

Parameter	Value used
N	100
G	500
N_e	5
T	2,000
δt	20 ms

TABLE I
THE SETTINGS USED FOR THE SIMULATION PARAMETERS INTRODUCED IN SUBSECTS. V-A AND V-B.

the wandering behavior executed by moving obstacles (B2.3) contains a random element, meaning that their motion will never be the same for different tests.

Thus, in order to reduce the effects of stochastic noise caused by the non-deterministic character of the motion of the moving obstacles, the evaluation of an individual is made by using several (N_e) copies of the individual. In principle, it would be possible e.g. to run the same individual N_e times, using different starting conditions. Here, however, a slightly different procedure has been used, in which N_e exact copies of the *same* individual are evaluated at once, in the same environment but with different starting position and heading for each copy. Both the starting position and the heading are pre-specified and identical for all individuals. Furthermore, the N_e robots that are evaluated simultaneously are *not* able to see each other, and collisions between robots are also turned off, even though the robots can, of course, collide with moving obstacles.

A given evaluation lasts for T time steps of length δt, unless a robot collides with an obstacle or a wall, or runs out of battery energy, in which the evaluation of that particular robot is terminated, while the other $N_e - 1$ robots are allowed to continue.

Robots consume energy according to Eq. 7 in all but the battery charging behaviors (B3.x). The discharge rate is set so that the battery is depleted in T_B seconds.

The fitness of an individual is calculated based on the distance travelled while executing the task behavior (B1.1 or B1.2), as described in Subsect. V-D below.

The settings of the various parameters introduced above are given in Table V-B.

C. Utility function implementation

Once the individual behaviors have been defined, the evolution of the behavioral organizer can begin. The aim of the UM method is to find correct utility functions, i.e. function parameters such that the behavioral selection provides the desired result. In principle, any functional form can be allowed for the utility functions. However, in practice (and as will be shown below), it is often sufficient to make an *ansatz* for the utility functions and to limit the search to parametric optimization.

In this study, the utility functions are defined by polynomial functions of degree P. As an example, consider a utility function U_i that depends on a sensor value S, the battery energy E, and an internal abstract variable x. For e.g. $P = 2$, the ansatz becomes

$$\begin{aligned} U_i &= a_{i,000} + a_{i,100}S + a_{i,010}E + a_{i,001}x + \\ &+ a_{i,200}S^2 + a_{i,020}E^2 + a_{i,002}x^2 + \\ &+ a_{i,110}SE + a_{i,101}Sx + a_{i,011}Ex, \end{aligned} \quad (8)$$

where the $a_{i,jkl}$ are constants that are to be determined by the EA.

In this study the external physical variables are taken as the readings $S_0, S_1, \ldots, S_7$ of the sensors, and the battery level E is the only internal physical variable.

One abstract variable is defined for all behaviors B1, B2, B4, and B4. The variation of the internal abstract variable must also be specified. In principle, the variable can be any function of sensor variables and behavior-time. However, for the purposes of this paper, the ansatz

$$x_i = \begin{cases} b_{i,1} + b_{i,2}e^{-|b_{i,3}|t_i} & \text{If } B_i \text{ is active,} \\ 0 & \text{Otherwise} \end{cases} \quad (9)$$

where $b_{i,j}$ are constants will be used for each abstract variable x_i. The behavior-time t_i increases linearly with (global) time if behavior i is active, and is zero otherwise. Furthermore the abstract variable x_i is exactly zero when the associated behavior is inactive.

The constants $a_{i,jkl}$ and $b_{i,j}$ are encoded in the chromosomes used by the EA, using real-number encoding, i.e. with one gene per variable.

Since the behavioral selection is based on the highest utility value, it implies no significant restriction to set the utility function for one behavior identically equal to zero. In this case,

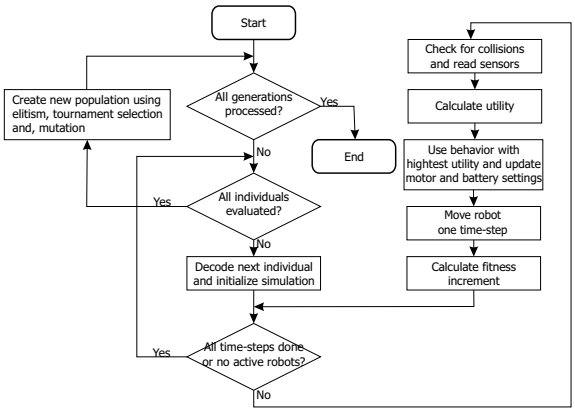

Fig. 4. Evolution and simulation flow chart.

the battery charging behavior was chosen as the behavior with zero utility (note that the utilities of other behaviors can take both negative and positive values), so that behaviors B3.1 and B3.2 do not have an abstract variable defined.

The polynomials used in this paper are of second degree, i.e. $P = 2$ leading to an effective genome length of 95 genes per utility function, of which 92 genes determines the polynomial coefficients, and 3 determine the variation of the internal abstract variable.

In most runs described below, only a single version of each behavior is available to the behavioral organizer, resulting in an effective genome length of $n_a - 1 \times 95$, where n_a is the number of available behaviors, and the negative term corresponds to behavior B3, for which no utility function is defined.

D. Fitness calculation procedure

The fitness of an individual is based on a combination of the fitness values f_i obtained for several (N_e) evaluated robots with identical brains (see above). The combination can be made in a variety of ways, of which two have been tried in this paper, namely the *average fitness* (fitness measure I), according to

$$f^{\mathrm{I}} = \sum_{i=1}^{N_e} f_i, \tag{10}$$

and the minimum fitness (fitness measure II), according to

$$f^{\mathrm{II}} = min_i f_i. \tag{11}$$

The motivation for the second fitness measure [11] is that it forces the EA to avoid even occasional failures.

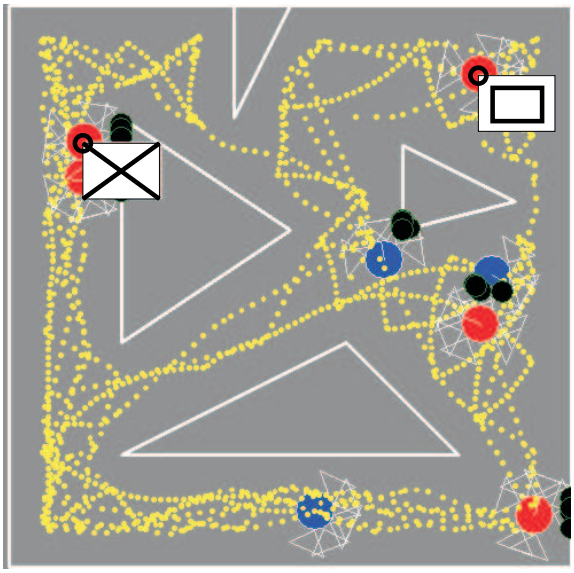

Fig. 5. Snap shot of the arena where the simulation and evaluation of a behavior coordinator is done. Small dots indicates a trail a robots has taken. The square and cross icons indicates a depleted battery and collision, respectively.

VI. Behavioral organization

As described previously is the aim of the robotic brain to select behaviors adequately during its execution in such a way that the robot completes the simulation with highest possible fitness. When investigating the properties of the UM method it is of great importance first to ascertain the ability of the method to find satisfactory solutions to the problem at hand. Thus, the results from basic simulations, involving a specific quartet of behaviors (namely B1.1, B2.1, B3.1, and B4.1) are presented first, followed by a detailed analysis of the evolved behavioral organizer. Next, the generality of the method is investigated, by evolving the same overall task, using all 16 distinct combinations of the four behaviors (for each of which two different implementations are provided, as discussed in Sect. IV.

The simulations were carried out on two standard DELL Dimension computers, equipped with 2.25 GHZ and 2.53 GHz Pentium 4 processors, respectively.

The overall goal for the robot is to execute the task behavior in a relatively unstructured environment with stationary and moving obstacles. Collision with an obstacle or depletion of a battery result in a premature termination of the robot.

Striving towards this goal, four behaviors,

B1.1, B2.1, B3.1, and B4.1 are used in this first investigation. Only the use of the task behavior, B1.1, rewards the robot during the evaluation. The auxiliary behaviors (B2.1, B3.1, and B4.1) are instruments to help the robot execute the task behavior in an efficient and continuous manner.

Several runs were performed using this configuration.

RESULTS SECTION TO BE COMPLETED BY 20041115.

A. Generality properties

Here, all the remaining 15 combinations of task and support behaviors were investigated. The evolution setup is the same as the previous investigation, and all runs lasted for a total of 200 generations.

RESULTS SECTION TO BE COMPLETED BY 20041115.

B. Conclusion

CONCLUSION SECTION TO BE COMPLETED BY 20041115.

REFERENCES

[1] R. A. Brooks, "A robust layered control system for a mobile robot," *Robotics and Autonomous Systems*, vol. 2, March 1986.

[2] M. Wahde, "A method for behavioural organization for autonomous robots based on evolutionary optimization of utility functions," *J. Systems and Control Engineering*, vol. 217, pp. 249–258, september 2003. Part I.

[3] J. Pettersson and M. Wahde, "Application of the utility manifold method for behavioral organization in a locomotion task," *IEEE trans. Ev. Comp., Submitted*, 2044.

[4] M. Wahde, *An Introduction to Adaptive Algorithms and Intelligent Machines, 2:nd ed.* Chalmers, 2004.

[5] D. McFarland, *Animal Behavior*. Addison Wesley Longman, 1993.

[6] D. McFarland and E. Spier, "Basic cycles, utility, and opportunism in self-sufficient robots," *Robotics and Autonomous Systems*, vol. 20, pp. 179–190, 1997.

[7] D. McFarland and T. Bösser, *Intelligent Behavior in Aninals and Robots*. MIT Press, 1993.

[8] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," *Lecture Notes in Computer Science*, vol. 929, pp. 704–720, 1995.

[9] B. Kim and P. Tsiotras, "Controller for unicycle-type wheeled robots: Theoretical results and experimental validation," *IEEE Tansactions on Robotics and Automation*, vol. 18, June 2003.

[10] T. Back, D. B. Fogel, and Z. Michalewicz, eds., *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University, 1997.

[11] J. Savage, E. Marquez, J. Pettersson, N. Trygg, A. Petersson, and M. Wahde, "Optimization of waypoint-guided potential field navigation using evolutionary algorithms," in *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2004)*, 2004.